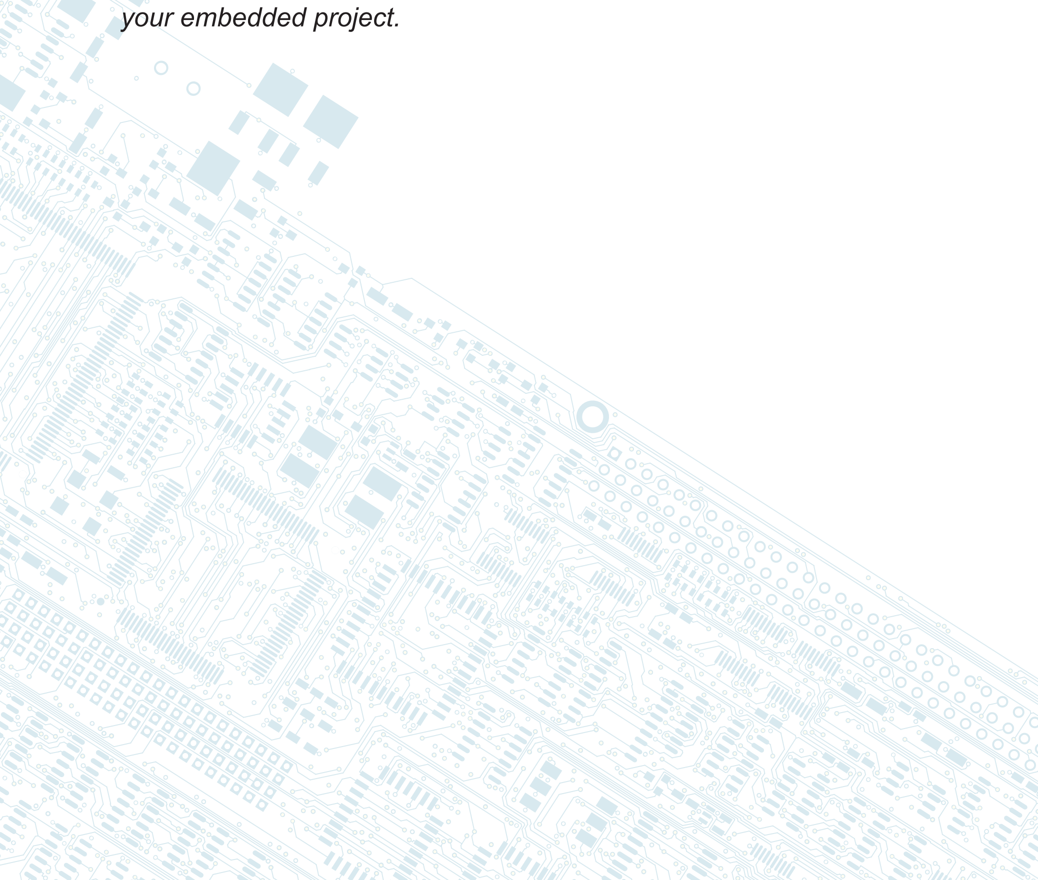
mikroBasic PRO for dsPIC™

Manual

mikroBasic PRO for dsPIC30/33 and PIC24 is a full-featured Basic compiler for dsPIC30, dsPIC33 and PIC24 MCUs from Microchip. It is designed for developing, building and debugging dsPIC30/33 and PIC24-based embedded applications. This development environment has a wide range of features such as: easy-to-use IDE, very compact and efficient code, many hardware and software libraries, comprehensive documentation, software simulator, COFF file generation, SSA optimization (up to 30% code reduction) and many more. Numerous ready-to-use and well-explained examples will give a good start for your embedded project.

Compiler



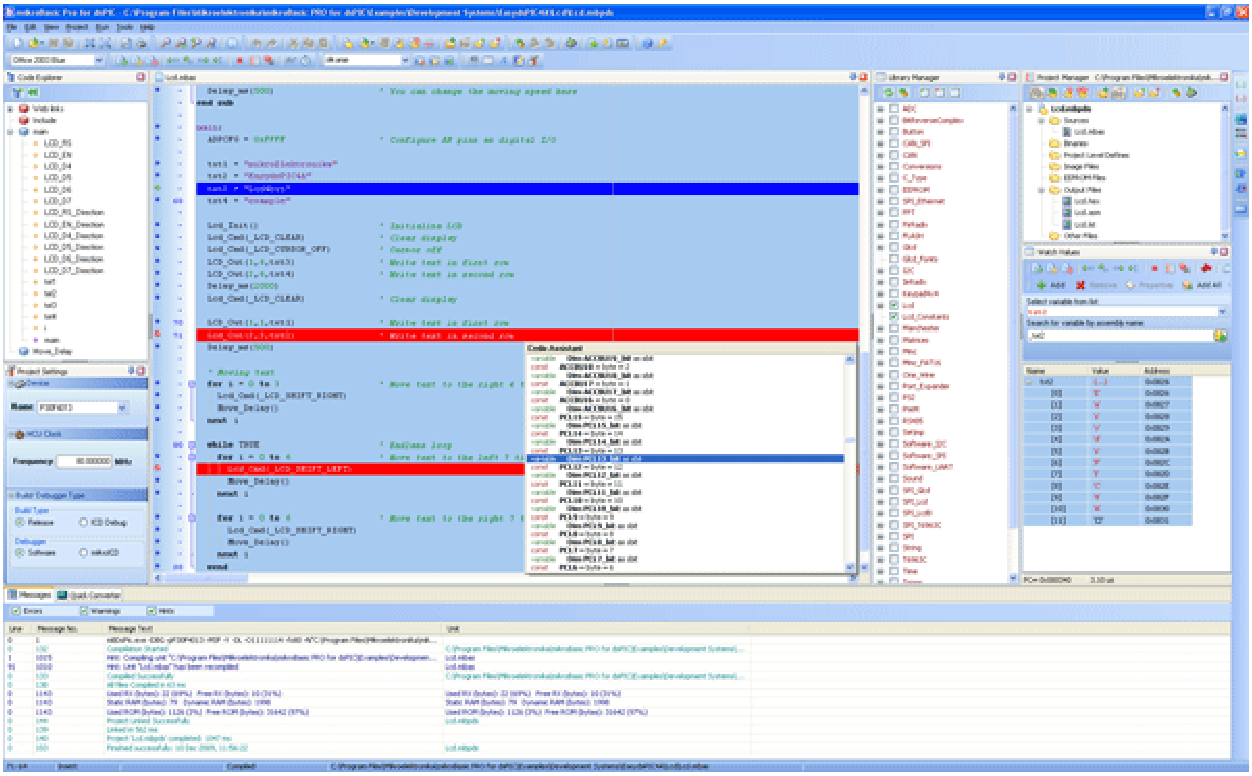
MikroElektronika
SOFTWARE AND HARDWARE SOLUTIONS FOR EMBEDDED WORLD ...making it simple

Table of Contents

CHAPTER 1

INTRODUCTION

mikroBasic PRO for dsPIC30/33 and PIC24 is a powerful, feature-rich development tool for dsPIC30/33 and PIC24 microcontrollers. It is designed to provide the programmer with the easiest possible solution to developing applications for embedded systems, without compromising performance or control.



mikroBasic PRO for dsPIC30/33 and PIC24 IDE

Introduction to mikroBasic PRO for dsPIC30/33 and PIC24

dsPIC30/33 and PIC24 and mikroBasic PRO for dsPIC30/33 and PIC24 fit together well: dsPIC is designed as a PIC with digital signal processing capabilities. These are Microchip's first inherent 16-bit (data) microcontrollers. They build on the PIC's existing strengths by offering hardware MAC (multiply-accumulate), barrel shifting, bit reversal, (16x16)-bit multiplication and other digital signal processing operations. Having a wide range of application and being also prized for efficiency, the dsPIC30/33 and PIC24 MCUs are a natural choice for developing embedded systems. mikroBasic PRO for dsPIC30/33 and PIC24 provides a successful match featuring highly advanced IDE, broad set of hardware libraries, comprehensive documentation, and plenty of ready-to-run examples.

Features

mikroBasic PRO for dsPIC30/33 and PIC24 allows you to quickly develop and deploy complex applications:

- Write your source code using the built-in Code Editor (Code and Parameter Assistants, Code Folding, Syntax Highlighting, Auto Correct, Code Templates, and more.)
- Use included mikroBasic PRO for dsPIC30/33 and PIC24 libraries to dramatically speed up the development: data acquisition, memory, displays, conversions, communication etc.
- Monitor your program structure, variables, and functions in the Code Explorer.
- Generate commented, human-readable assembly, and standard HEX compatible with all programmers.
- Use the integrated mikroICD (In-Circuit Debugger) Real-Time debugging tool to monitor program execution on the hardware level.
- Inspect program flow and debug executable logic with the integrated Software Simulator.
- Generate COFF(Common Object File Format) file for software and hardware debugging under Microchip's MPLAB software.
- Use Single Static Assingment optimization to shrink your code to even smaller size.
- Get detailed reports and graphs: RAM and ROM map, code statistics, assembly listing, calling tree, and more.
- Active Comments enable you to make your comments alive and interactive.
- mikroBasic PRO for dsPIC30/33 and PIC24 provides plenty of examples to expand, develop, and use as building bricks in your projects. Copy them entirely if you deem fit – that's why we included them with the compiler.

Where to Start

- In case that you're a beginner in programming the dsPIC30/33 and PIC24 microcontrollers, read carefully the dsPIC Specifics chapter. It might give you some useful information on the dsPIC30/33 and PIC24 constraints, code portability, and good programming practices.
- If you are experienced in Basic programming, you will probably want to consult the mikroBasic PRO for dsPIC30/33 and PIC24 Specifics first. For language issues, you can always refer to the comprehensive Language Reference. A complete list of included libraries is available in the mikroBasic PRO for dsPIC30/33 and PIC24 Libraries.
- If you are not very experienced in Basic programming, don't panic! mikroBasic PRO for dsPIC30/33 and PIC24 provides plenty of examples making it easy for you to go quickly through it . We suggest you to consult Projects and Source Files first, and then start browsing the examples that you're the most interested in.

What's new in mikroBasic PRO for dsPIC30/33 and PIC24

IDE build 4.60
Command line build 4.60

New features and enhancements in the following areas will boost your productivity by helping you complete many tasks more easily and in less time.

For a complete version history of mikroBasic PRO for dsPIC30/33 and PIC24 2010, visit the following link:
http://www.mikroe.com/download/eng/documents/compilers/mikrobasic/pro/dspic/version_history

- Compiler Changes
- IDE Changes

Compiler Changes

Fixed:

- Optimization issues in specific cases when destination variable is in Rx space.

IDE Changes

Fixed:

- Compiler version is not visible in caption if no projects are open.
- Parameter assistant ignores commas when switching to another parameter.
- Occasional lost of configuration flags when swithing between projets.
- Improper display of RAM memory usage in statistics.

Improved:

- Communication to programmer concerning supported chips.
- License Key Request form.

Software License Agreement

mikroElektronika Associates License Statement and Limited Warranty

IMPORTANT - READ CAREFULLY

This license statement and limited warranty constitute a legal agreement ("License Agreement") between you (either as an individual or a single entity) and mikroElektronika ("mikroElektronika Associates") for software product ("Software") identified above, including any software, media, and accompanying on-line or printed documentation.

BY INSTALLING, COPYING, OR OTHERWISE USING SOFTWARE, YOU AGREE TO BE BOUND BY ALL TERMS AND CONDITIONS OF THE LICENSE AGREEMENT.

Upon your acceptance of the terms and conditions of the License Agreement, mikroElektronika Associates grants you the right to use Software in a way provided below.

This Software is owned by mikroElektronika Associates and is protected by copyright law and international copyright treaty. Therefore, you must treat this Software like any other copyright material (e.g., a book).

You may transfer Software and documentation on a permanent basis provided. You retain no copies and the recipient agrees to the terms of the License Agreement. Except as provided in the License Agreement, you may not transfer, rent, lease, lend, copy, modify, translate, sublicense, time-share or electronically transmit or receive Software, media or documentation. You acknowledge that Software in the source code form remains a confidential trade secret of mikroElektronika Associates and therefore you agree not to modify Software or attempt to reverse engineer, decompile, or disassemble it, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

If you have purchased an upgrade version of Software, it constitutes a single product with the mikroElektronika Associates software that you upgraded. You may use the upgrade version of Software only in accordance with the License Agreement.

LIMITED WARRANTY

Respectfully excepting the Redistributables, which are provided "as is", without warranty of any kind, mikroElektronika Associates warrants that Software, once updated and properly used, will perform substantially in accordance with the accompanying documentation, and Software media will be free from defects in materials and workmanship, for a period of ninety (90) days from the date of receipt. Any implied warranties on Software are limited to ninety (90) days.

mikroElektronika Associates' and its suppliers' entire liability and your exclusive remedy shall be, at mikroElektronika Associates' option, either (a) return of the price paid, or (b) repair or replacement of Software that does not meet mikroElektronika Associates' Limited Warranty and which is returned to mikroElektronika Associates with a copy of your receipt. DO NOT RETURN ANY PRODUCT UNTIL YOU HAVE CALLED MIKROELEKTRONIKA ASSOCIATES FIRST AND OBTAINED A RETURN AUTHORIZATION NUMBER. This Limited Warranty is void if failure of Software has resulted from an accident, abuse, or misapplication. Any replacement of Software will be warranted for the rest of the original warranty period or thirty (30) days, whichever is longer.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, MIKROELEKTRONIKA ASSOCIATES AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESSED OR IMPLIED, INCLUDED, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, WITH REGARD TO SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES.

IN NO EVENT SHALL MIKROELEKTRONIKA ASSOCIATES OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS AND BUSINESS INFORMATION, BUSINESS INTERRUPTION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF MIKROELEKTRONIKA ASSOCIATES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, MIKROELEKTRONIKA ASSOCIATES' ENTIRE LIABILITY UNDER ANY PROVISION OF THIS LICENSE AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR SOFTWARE PRODUCT PROVIDED, HOWEVER, IF YOU HAVE ENTERED INTO A MIKROELEKTRONIKA ASSOCIATES SUPPORT SERVICES AGREEMENT, MIKROELEKTRONIKA ASSOCIATES' ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT.

HIGH RISK ACTIVITIES

Software is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). mikroElektronika Associates and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

GENERAL PROVISIONS

This statement may only be modified in writing signed by you and an authorised officer of mikroElektronika Associates. If any provision of this statement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided is determined to have failed for its essential purpose, all limitations of liability and exclusions of damages set forth in the Limited Warranty shall remain in effect.

This statement gives you specific legal rights; you may have others, which vary, from country to country. mikroElektronika Associates reserves all rights not specifically granted in this statement.

mikroElektronika
Visegradska 1A,
11000 Belgrade,
Europe.

Phone: + 381 11 36 28 830
Fax: +381 11 36 28 831
Web: www.mikroe.com
E-mail: office@mikroe.com

Technical Support

The latest software can be downloaded free of charge via Internet (you might want to bookmark the page so you could check news, patches, and upgrades later on): www.mikroe.com/en/compilers/mikrobasic PRO/dspic/download.htm .

In case you encounter any problem, you are welcome to our support forums at www.mikroe.com/forum/. Here, you may also find helpful information, hardware tips, and practical code snippets. Your comments and suggestions on future development of the mikroBasic PRO for dsPIC30/33 and PIC24 are always appreciated — feel free to drop a note or two on our Wishlist.

In our Knowledge Base www.mikroe.com/en/kb/ you can find the answers to Frequently Asked Questions and solutions to known problems. If you can not find the solution to your problem in Knowledge Base then report it to Support Desk www.mikroe.com/en/support/. In this way, we can record and track down bugs more efficiently, which is in our mutual interest. We respond to every bug report and question in a suitable manner, ever improving our technical support.

How to Register

The latest version of the mikroBasic PRO for dsPIC30/33 and PIC24 is always available for downloading from our website. It is a fully functional software with the mikroICD(in-circuit Debugger), all the libraries, examples, and comprehensive help included.

The only limitation of the free version is that it cannot generate hex output over 2K of program words. Although it might sound restrictive, this margin allows you to develop practical, working applications with no thinking of demo limit. If you intend to develop really complex projects in the mikroBasic PRO for dsPIC30/33 and PIC24, then you should consider the possibility of purchasing the license key.

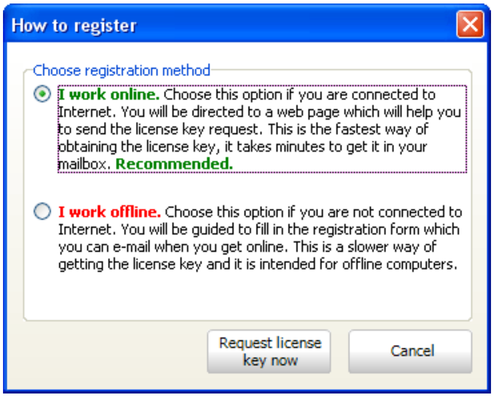
Who Gets the License Key

Buyers of the mikroBasic PRO for dsPIC30/33 and PIC24 are entitled to the license key. After you have completed the payment procedure, you have an option of registering your mikroBasic PRO for dsPIC30/33 and PIC24. In this way you can generate hex output without any limitations.

How to Get License Key

After you have completed the payment procedure, start the program. Select **Help › How to Register** from the drop-down menu or click the How To Register Icon 🔑 .

You can choose between two registering methods, **I work online** or **I work offline**, based on your current internet connection and click **Request license key now** button:



● **I work online.** Choose this option if you are connected to Internet. You will be directed to a web page which will help you to send the license key request. This is the fastest way of obtaining the license key, it takes minutes to get it in your mailbox. **Recommended.**

○ **I work offline.** Choose this option if you are not connected to Internet. You will be guided to fill in the registration form which you can e-mail when you get online. This is a slower way of getting the license key and it is intended for offline computers.

If you choose **I work online** registering method, following page will be opened in your default browser:

Software Activation

In order to get activation key please fill in required fields. Upon receiving and verifying your request, we will send the license key to the e-mail address you specified in the form.

Product: mikroBasic PRO for dsPIC30/33 and PIC24
Name* : John Smith
Address:
Invoice:
2CO Number:
Email* : jsmith@example.com
Re-enter email* : jsmith@example.com
Company:
Product ID: 3F47-546774-7F6A73-5552F7
Comment:
Distributor* : MikroElektronika

If you do not specify 2CO Number or invoice number then the license key request must be processed manually which can take longer time.

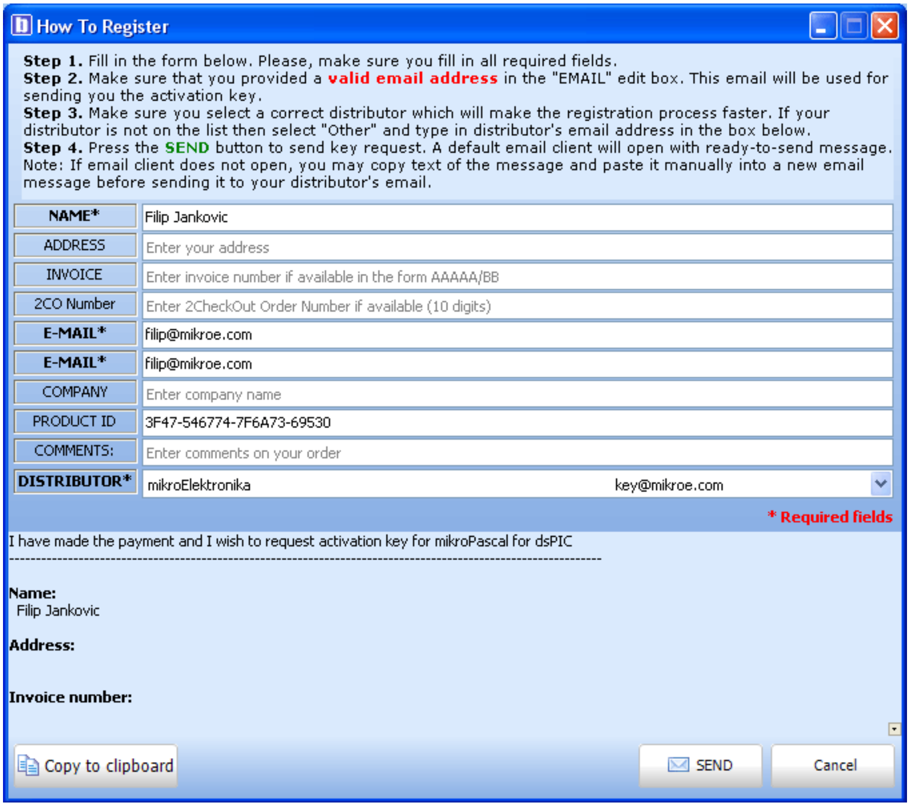
Type the two words: progrant I744

Related Links: Products News Forums Distributors About MikroElektronika Legal Information and Privacy Policy Product Archive Contact Us

Copyright © 1998-2010. MikroElektronika. All rights reserved. All trade and/or services marks mentioned are the property of their respective owners.

Fill out the registration form, select your distributor, and click the **Submit** button.

If you choose **I work offline** registering method, following window will be opened:



Step 1. Fill in the form below. Please, make sure you fill in all required fields.
Step 2. Make sure that you provided a **valid email address** in the "EMAIL" edit box. This email will be used for sending you the activation key.
Step 3. Make sure you select a correct distributor which will make the registration process faster. If your distributor is not on the list then select "Other" and type in distributor's email address in the box below.
Step 4. Press the **SEND** button to send key request. A default email client will open with ready-to-send message.
Note: If email client does not open, you may copy text of the message and paste it manually into a new email message before sending it to your distributor's email.

NAME*	Filip Jankovic
ADDRESS	Enter your address
INVOICE	Enter invoice number if available in the form AAAAA/BB
2CO Number	Enter 2CheckOut Order Number if available (10 digits)
E-MAIL*	filip@mikroe.com
E-MAIL*	filip@mikroe.com
COMPANY	Enter company name
PRODUCT ID	3F47-546774-7F6A73-69530
COMMENTS:	Enter comments on your order
DISTRIBUTOR*	mikroElektronika key@mikroe.com

I have made the payment and I wish to request activation key for mikroPascal for dsPIC

Name:
Filip Jankovic

Address:

Invoice number:

Fill out the registration form, select your distributor, and click the **Submit** button.

This will start your e-mail client with message ready for sending. Review the information you have entered, and add the comment if you deem it necessary. Please, do not modify the subject line.

Upon receiving and verifying your request, we will send the license key to the e-mail address you specified in the form.

After Receving the License Key

The license key comes as a small autoextracting file – just start it anywhere on your computer in order to activate your copy of compiler and remove the demo limit. You do not need to restart your computer or install any additional components. Also, there is no need to run mikroBasic PRO for dsPIC30/33 and PIC24 at the time of activation.

Important:

- The license key is valid until you format your hard disk. In case you need to format the hard disk, you should request a new activation key.
- Please keep the activation program in a safe place. Every time you upgrade the compiler you should start this program again in order to reactivate the license.

CHAPTER 2

mikroBasic PRO for dsPIC30/33 and PIC24 Environment

Main Menu Options

Available Main Menu options are:


File


Edit


View


Project


Build


Run


Tools

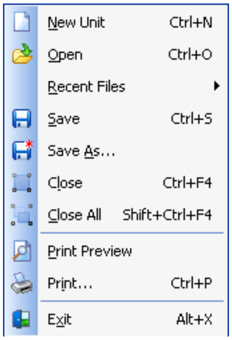

Help

Related topics: Keyboard shortcuts, Toolbars

File

File Menu Options

The File menu is the main entry point for manipulation with the source files.



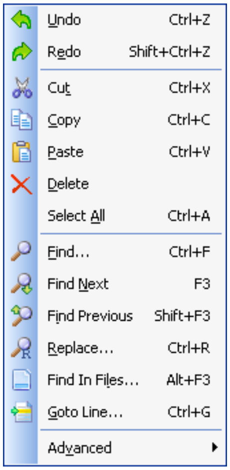
File	Description
New Unit Ctrl+N	Open a new editor window.
Open Ctrl+O	Open source file for editing or image file for viewing.
Recent Files	Reopen recently used file.
Save Ctrl+S	Save changes for active editor.
Save As...	Save the active source file with the different name or change the file type.
Close Ctrl+F4	Close active source file.
Close All Shift+Ctrl+F4	Close all opened files.
Print Preview	Print Preview.
Print... Ctrl+P	Print.
Exit Alt+X	Exit IDE.

Related topics: Keyboard shortcuts, File Toolbar, Managing Source Files

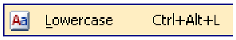
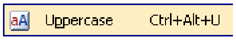
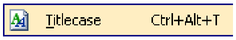
Edit

Edit Menu Options

The Edit Menu contains commands for editing the contents of the current document.

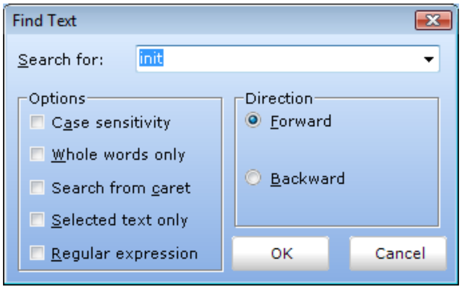


Edit	Description
Undo Ctrl+Z	Undo last change.
Redo Shift+Ctrl+Z	Redo last change.
Cut Ctrl+X	Cut selected text to clipboard.
Copy Ctrl+C	Copy selected text to clipboard.
Paste Ctrl+V	Paste text from clipboard.
Delete	Delete selected text.
Select All Ctrl+A	Select all text in active editor.
Find... Ctrl+F	Find text in active editor.
Find Next F3	Find next occurence of text in active editor.
Find Previous Shift+F3	Find previous occurence of text in active editor.
Replace... Ctrl+R	Replace text in active editor.
Find In Files... Alt+F3	Find text in current file, in all opened files, or in files from desired folder.
Goto Line... Ctrl+G	Go to line to the desired line in active editor.
Advanced	Advanced Code Editor options

Advanced »	Description
 Comment Shift+Ctrl+.	Comment selected code or put single line comment if there is no selection.
 Uncomment Shift+Ctrl+,	Uncomment selected code or remove single line comment if there is no selection.
 Indent Shift+Ctrl+I	Indent selected code.
 Outdent Shift+Ctrl+U	Outdent selected code.
 Lowercase Ctrl+Alt+L	Changes selected text case to lowercase.
 Uppercase Ctrl+Alt+U	Changes selected text case to uppercase.
 Titlecase Ctrl+Alt+T	Changes selected text case to titlercase.

Find Text

Dialog box for searching the document for the specified text. The search is performed in the direction specified. If the string is not found a message is displayed.



Find Text

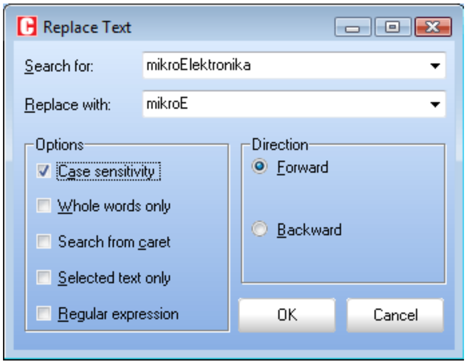
Search for: init

Options
☐ Case sensitivity
☐ Whole words only
☐ Search from caret
☐ Selected text only
☐ Regular expression

Direction
● Forward
○ Backward

Replace Text

Dialog box for searching for a text string in file and replacing it with another text string.



Replace Text

Search for: mikroElektronika

Replace with: mikroE

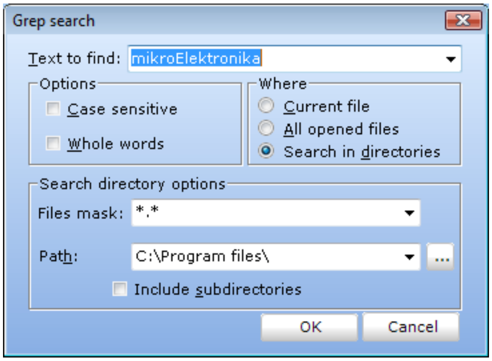
Options
☑ Case sensitivity
☐ Whole words only
☐ Search from caret
☐ Selected text only
☐ Regular expression

Direction
● Forward
○ Backward

Find In Files

Dialog box for searching for a text string in current file, all opened files, or in files on a disk.

The string to search for is specified in the **Text to find** field. If Search in directories option is selected, The files to search are specified in the **Files mask** and **Path** fields.



Text to find: mikroElektronika

☐ Case sensitive
☐ Whole words

○ Current file
○ All opened files
● Search in directories

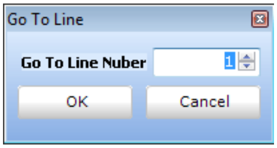
Files mask: *.*

Path: C:\Program files\

☐ Include subdirectories

Go To Line

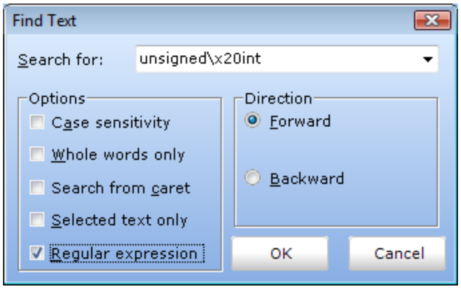
Dialog box that allows the user to specify the line number at which the cursor should be positioned.



Go To Line Nuber 1

Regular expressions option

By checking this box, you will be able to advance your search, through Regular expressions.



Search for: unsigned\x20int

☐ Case sensitivity
☐ Whole words only
☐ Search from caret
☐ Selected text only
☑ Regular expression

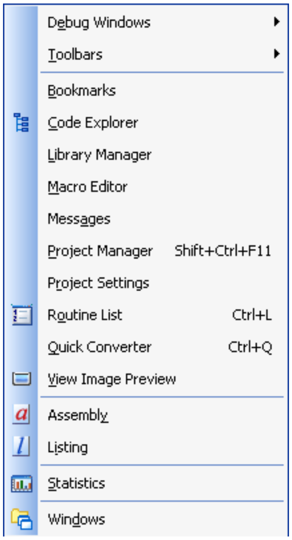
● Forward
○ Backward

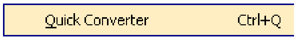
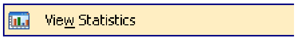
Related topics: Keyboard shortcuts, Edit Toolbar, Advanced Edit Toolbar

View

View Menu Options

View Menu contains commands for controlling the on-screen display of the current project.



View	Description
 Debug Windows	Show/Hide Software Simulator / mikroICD (In-Circuit Debugger) debug windows.
 Toolbars	Show/Hide Toolbars.
 Bookmarks	Show/Hide Bookmarks window.
 Code Explorer	Show/Hide Code Explorer window.
 Library Manager	Show/Hide Library Manager window.
 Macro Editor	Show/Hide Macro Editor window.
 Messages	Show/Hide Messages window.
 Project Manager Shift+Ctrl+F11	Show/Hide Project Manager window.
 Project Settings	Show/Hide Project Settings window.
 Routine List Ctrl+L	Show/Hide Routine List in active editor.
 Quick Converter Ctrl+Q	Show/Hide Quick Converter window.
 View Image Preview	Show/Hide View Image Preview window.
 View Assembly	View Assembly.
 View Listing	View Listing.
 View Statistics	View Statistics.
 Windows	Show Window List window.

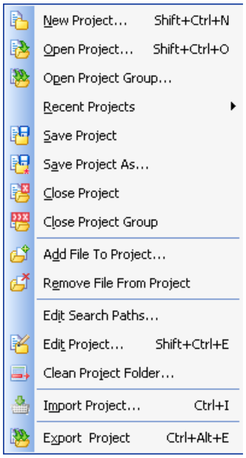
The Tools toolbar can easily be customized by adding new tools in Options(F12) window.

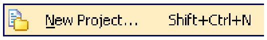
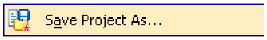
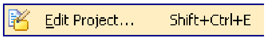
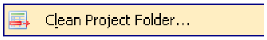
Related topics: Keyboard shortcuts, Integrated Tools, Software Simulator

Project

Project Menu Options

Project Menu allows the user to easily manipulate current project.



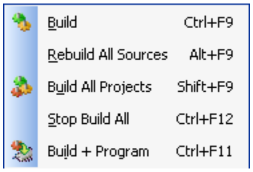
Project	Description
 New Project... Shift+Ctrl+N	Open New Project Wizard
 Open Project... Shift+Ctrl+O	Open existing project.
 Open Project Group...	Open project group.
 Recent Projects	Open recently used project or project group.
 Save Project	Save current project.
 Save Project As...	Save active project file with the different name.
 Close Project	Close active project.
 Close Project Group	Close project group.
 Add File To Project...	Add file to project.
 Remove File From Project	Remove file from project.
 Edit Search Paths...	Edit search paths.
 Edit Project... Shift+Ctrl+E	Edit project settings
 Clean Project Folder...	Clean Project Folder
 Import Project... Ctrl+I	Import projects created in previous versions of mikroBasic.
 Export Project Ctrl+Alt+E	Export Project.

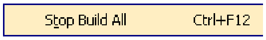
Related topics: Keyboard shortcuts, Project Toolbar, Creating New Project, Project Manager, Project Settings

Build

Build Menu Options

Build Menu allows the user to easily manage building and compiling process.



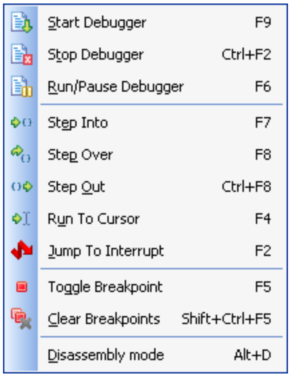
Build	Description
 Build Ctrl+F9	Build active project.
 Rebuild All Sources Alt+F9	Rebuild all sources in acrive project.
 Build All Projects Shift+F9	Build all projects.
 Stop Build All Ctrl+F12	Stop building of all projects.
 Build + Program Ctrl+F11	Build and program active project.

Related topics: Keyboard shortcuts, Project Toolbar, Creating New Project, Project Manager, Project Settings

Run

Run Menu Options

Run Menu is used to debug and test compiled code on a software or harware level.



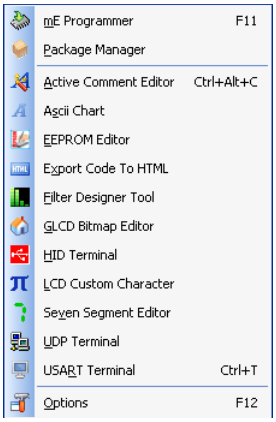
Run	Description
Start Debugger F9	Start Software Simulator or mikroICD (In-Circuit Debugger).
Stop Debugger Ctrl+F2	Stop debugger.
Run/Pause Debugger F6	Run/Pause Debugger.
Step Into F7	Step Into.
Step Over F8	Step Over.
Step Out Ctrl+F8	Step Out.
Run To Cursor F4	Run To Cursor.
Jump To Interrupt F2	Jump to interrupt in current project.
Toggle Breakpoint F5	Toggle Breakpoint.
Clear Breakpoints Shift+Ctrl+F5	Clear Breakpoints.
Disassembly mode Alt+D	Toggle between source and disassembly.

Related topics: Keyboard shortcuts, Debug Toolbar

Tools

Tools Menu Options

Tools Menu contains a number of applications designed to ease the use of compiler and included library routines.



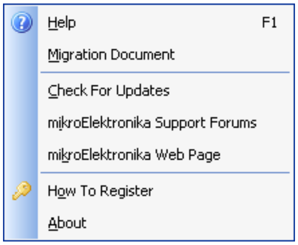
mE Programmer F11
Package Manager
Active Comment Editor Ctrl+Alt+C
Ascii Chart
EEPROM Editor
Export Code To HTML
Filter Designer Tool
GLCD Bitmap Editor
HID Terminal
LCD Custom Character
Seven Segment Editor
UDP Terminal
USART Terminal Ctrl+T
Options F12

Tools	Description
mE Programmer F11	Run mikroElektronika Programmer.
Package Manager	Run Package Manager.
Active Comment Editor Ctrl+Alt+C	Show/Hide Active Comment Editor window.
Ascii Chart	Run ASCII Chart
EEPROM Editor	Run EEPROM Editor
Export Code To HTML	Generate HTML code suitable for publishing source code on the web.
Filter Designer Tool	Run Filter Designer Tool.
GLCD Bitmap Editor	Run Glcd bitmap editor
HID Terminal	Run HID Terminal
LCD Custom Character	Run Lcd custom character
Seven Segment Editor	Run Seven Segment Editor
UDP Terminal	Run UDP communication terminal
USART Terminal Ctrl+T	Run USART Terminal
Options F12	Open Options window

Related topics: Keyboard shortcuts, Tools Toolbar

Help

Help Menu Options



Help	Description
Help F1	Open Help File.
Migration Document	Open Code Migration Document.
Check For Updates	Check if new compiler version is available.
mikroElektronika Support Forums	Open mikroElektronika Support Forums in a default browser.
mikroElektronika Web Page	Open mikroElektronika Web Page in a default browser.
How To Register	Information on how to register
About	Open About window.

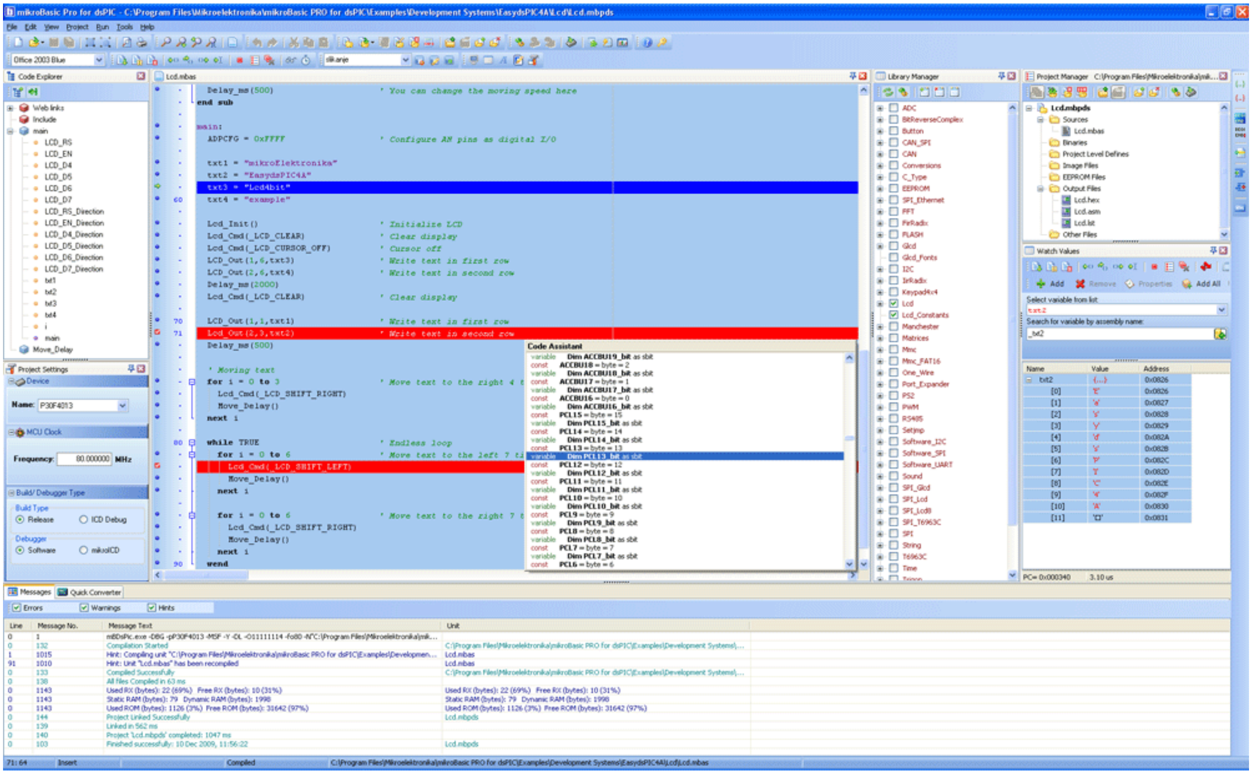
Related topics: Keyboard shortcuts, Help Toolbar

mikroBasic PRO for dsPIC30/33 and PIC24 IDE

IDE Overview

The mikroBasic PRO for dsPIC30/33 and PIC24 is an user-friendly and intuitive environment.

For a detailed information on a certain part of IDE, simply click on it (hovering a mouse cursor above a desired IDE part will pop-up its name):



```
  Delay_ms(500)                        ' You can change the moving speed here
end sub

main:
  ADPCFG = 0xFFFF                      ' Configure AN pins as digital I/O

  txt1 = "mikroElektronika"
  txt2 = "EasydsPIC4A"
  txt3 = "Lcd4bit"
  txt4 = "example"

  Lcd_Init()                           ' Initialize LCD
  Lcd_Cmd(_LCD_CLEAR)                  ' Clear display
  Lcd_Cmd(_LCD_CURSOR_OFF)             ' Cursor off
  LCD_Out(1,6,txt3)                    ' Write text in first row
  LCD_Out(2,6,txt4)                    ' Write text in second row
  Delay_ms(2000)
  Lcd_Cmd(_LCD_CLEAR)                  ' Clear display

  LCD_Out(1,1,txt1)                    ' Write text in first row
  Lcd_Out(2,3,txt2)                    ' Write text in second row
  Delay_ms(500)

  ' Moving text
  for i = 0 to 3                       ' Move text to the right 4 t
    Lcd_Cmd(_LCD_SHIFT_RIGHT)
    Move_Delay()
  next i

  while TRUE                           ' Endless loop
    for i = 0 to 6                     ' Move text to the left 7 ti
      Lcd_Cmd(_LCD_SHIFT_LEFT)
      Move_Delay()
    next i

    for i = 0 to 6                     ' Move text to the right 7 t
      Lcd_Cmd(_LCD_SHIFT_RIGHT)
      Move_Delay()
    next i
  wend
```

Name: P30F4013

Frequency: 80.000000 MHz

Build Type: ● Release ○ ICD Debug

Debugger: ● Software ○ mikroICD

```
Line  Message No.  Message Text                                                                    Unit
0     1            mBDsPic.exe -DBG -pP30F4013 -MSF -Y -DL -O11111114 -fo80 -N"C:\Program Files\Mikroelektronika\mik...
0     132          Compilation Started                                                             C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Examples\Development Systems\...
1     1015         Hint: Compiling unit "C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Examples\Developmen...   Lcd.mbas
91    1010         Hint: Unit "Lcd.mbas" has been recompiled                                       Lcd.mbas
0     133          Compiled Successfully                                                           C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Examples\Development Systems\...
0     138          All files Compiled in 63 ms
0     1143         Used RX (bytes): 22 (69%)  Free RX (bytes): 10 (31%)                            Used RX (bytes): 22 (69%)  Free RX (bytes): 10 (31%)
0     1143         Static RAM (bytes): 79  Dynamic RAM (bytes): 1998                               Static RAM (bytes): 79  Dynamic RAM (bytes): 1998
0     1143         Used ROM (bytes): 1126 (3%)  Free ROM (bytes): 31642 (97%)                      Used ROM (bytes): 1126 (3%)  Free ROM (bytes): 31642 (97%)
0     144          Project Linked Successfully                                                     Lcd.mbpds
0     139          Linked in 562 ms
0     140          Project 'Lcd.mbpds' completed: 1047 ms
0     103          Finished successfully: 10 Dec 2009, 11:56:22                                    Lcd.mbpds
```

- The Code Editor features adjustable Syntax Highlighting, Code Folding, Code Assistant, Parameters Assistant, Spell Checker, Auto Correct for common typos and Code Templates (Auto Complete).
- The Code Explorer is at your disposal for easier project management.
- The Project Manager alows multiple project management
- General project settings can be made in the Project Settings window
- Library manager enables simple handling libraries being used in a project
- The Messages Window displays all information, messages and errors detected during compiling and linking.
- The source-level Software Simulator lets you debug executable logic step-by-step by watching the program flow.
- The New Project Wizard is a fast, reliable, and easy way to create a project.
- Help files are syntax and context sensitive.
- Like in any modern Windows application, you may customize the layout of mikroBasic PRO for dsPIC30/33 and PIC24 to suit your needs best.
- Spell checker underlines identifiers which are unknown to the project. In this way it helps the programmer to spot potential problems early, much before the project is compiled.
- Spell checker can be disabled by choosing the option in the Preferences dialog (F12).

Code Editor

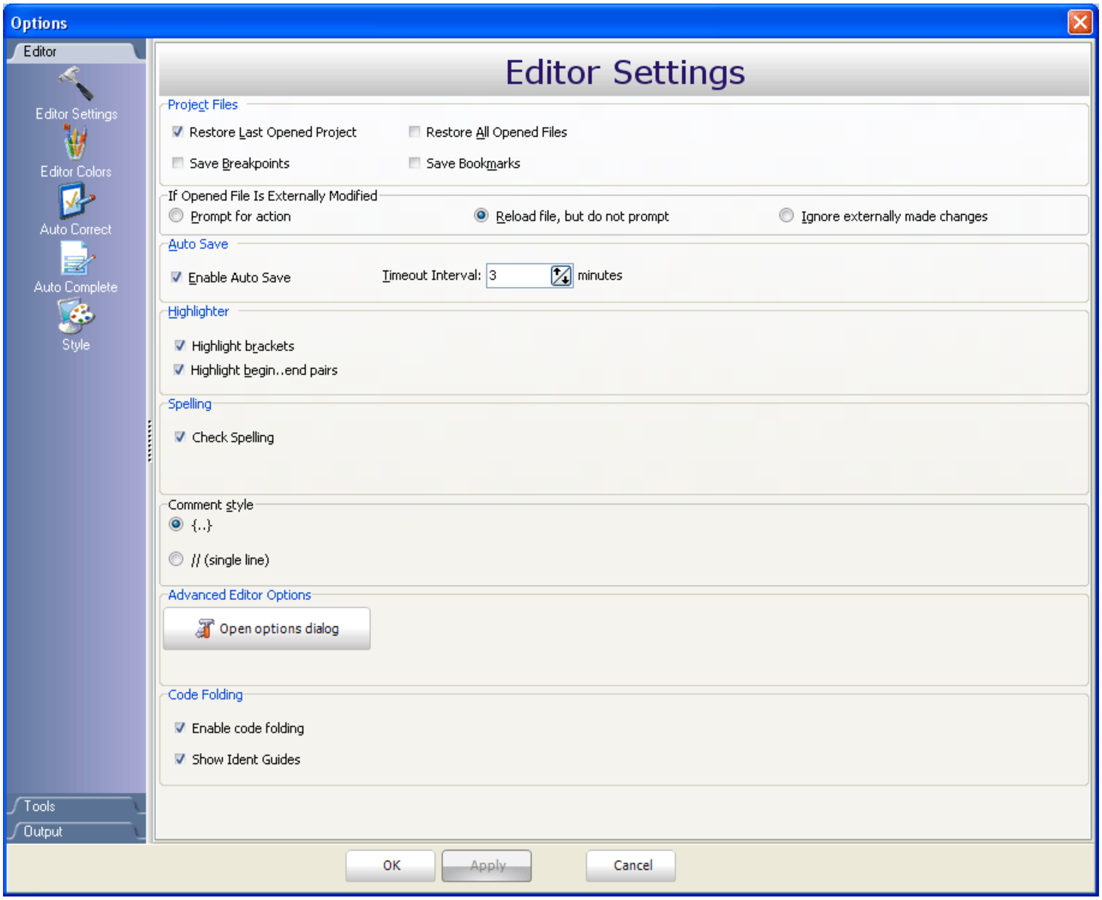
The Code Editor is advanced text editor fashioned to satisfy needs of professionals. General code editing is the same as working with any standard text-editor, including familiar Copy, Paste and Undo actions, common for Windows environment.

Available Code Editor options are: Editor Settings, Editor Colors, Auto Correct, Auto Complete and Style.

Editor Settings

Main Editor Settings Features are:

- Auto Save
- Highlighter
- Spelling
- Comment Style
- Code Folding
- Code Assistant
- Parameter Assistant
- Bookmarks and Go to Line



Project Files
☑ Restore Last Opened Project ☐ Restore All Opened Files
☐ Save Breakpoints ☐ Save Bookmarks

If Opened File Is Externally Modified
○ Prompt for action ● Reload file, but do not prompt ○ Ignore externally made changes

Auto Save
☑ Enable Auto Save Timeout Interval: 3 minutes

Highlighter
☑ Highlight brackets
☑ Highlight begin..end pairs

Spelling
☑ Check Spelling

Comment style
● {..}
○ // (single line)

Advanced Editor Options

Code Folding
☑ Enable code folding
☑ Show Ident Guides

Auto Save

Auto Save is a function which saves an opened project automatically, helping to reduce the risk of data loss in case of a crash or freeze. Autosaving is done in time intervals defined by the user.

Highlighter

Highlighting is a convenient feature for spotting brackets which notate begin or end of a routine, by making them visually distinct.

Spelling

The Spell Checker underlines unknown objects in the code, so they can be easily noticed and corrected before compiling your project.

Select **Tools › Options** from the drop-down menu, or click the Show Options Icon and then select the Spell Checker Tab.

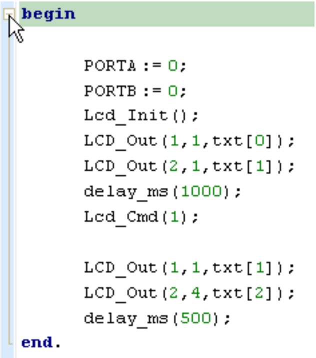
Comment Style

Code Editor has a feature to change the comment style to either single-line or multi-line. Commenting or uncommenting the selected code is done by a simple click of a mouse, using the Comment Icon {..} and Uncomment Icon {..} from the Advanced Edit Toolbar.

Code Folding

Code folding is IDE feature which allows users to selectively hide and display sections of a source file. In this way it is easier to manage large regions of code within one window, while still viewing only those subsections of the code that are relevant during a particular editing session.

While typing, the code folding symbols (⊟ and ⊞) appear automatically. Use the folding symbols to hide/unhide the code subsections.

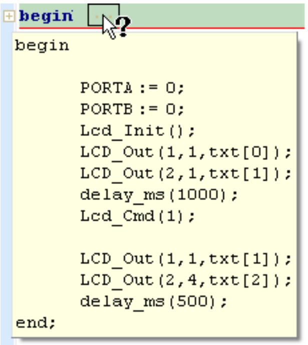


```
begin

        PORTA := 0;
        PORTB := 0;
        Lcd_Init();
        LCD_Out(1,1,txt[0]);
        LCD_Out(2,1,txt[1]);
        delay_ms(1000);
        Lcd_Cmd(1);

        LCD_Out(1,1,txt[1]);
        LCD_Out(2,4,txt[2]);
        delay_ms(500);
end.
```

```
begin ...
```

Another way of folding/unfolding code subsections is by using Alt+← and Alt+→.

If you place a mouse cursor over the tooltip box, the collapsed text will be shown in a tooltip style box.



```
begin
begin

      PORTA := 0;
      PORTB := 0;
      Lcd_Init();
      LCD_Out(1,1,txt[0]);
      LCD_Out(2,1,txt[1]);
      delay_ms(1000);
      Lcd_Cmd(1);

      LCD_Out(1,1,txt[1]);
      LCD_Out(2,4,txt[2]);
      delay_ms(500);
end;
```

Code Assistant

If you type the first few letters of a word and then press Ctrl+Space, all valid identifiers matching the letters you have typed will be prompted in a floating panel (see the image below). Now you can keep typing to narrow the choice, or you can select one from the list using the keyboard arrows and Enter.

```
sp
variable    sfr  SP: byte
variable    sfr  SPDR: byte
variable    sfr  SPSR: byte
variable    sfr  SPCR: byte
```

Parameter Assistant

The Parameter Assistant will be automatically invoked when you open parenthesis "(" or press Shift+Ctrl+Space. If the name of a valid function precedes the parenthesis, then the expected parameters will be displayed in a floating panel. As you type the actual parameter, the next expected parameter will become bold.

```
ADC_Read(
```

Bookmarks

Bookmarks make navigation through a large code easier. To set a bookmark, use Ctrl+Shift+*number*. The same princliple applies to the removal of the bookmarks. To jump to a bookmark, use Ctrl+*number*.

Go to Line

The Go to Line option makes navigation through a large code easier. Use the shortcut Ctrl+G to activate this option.

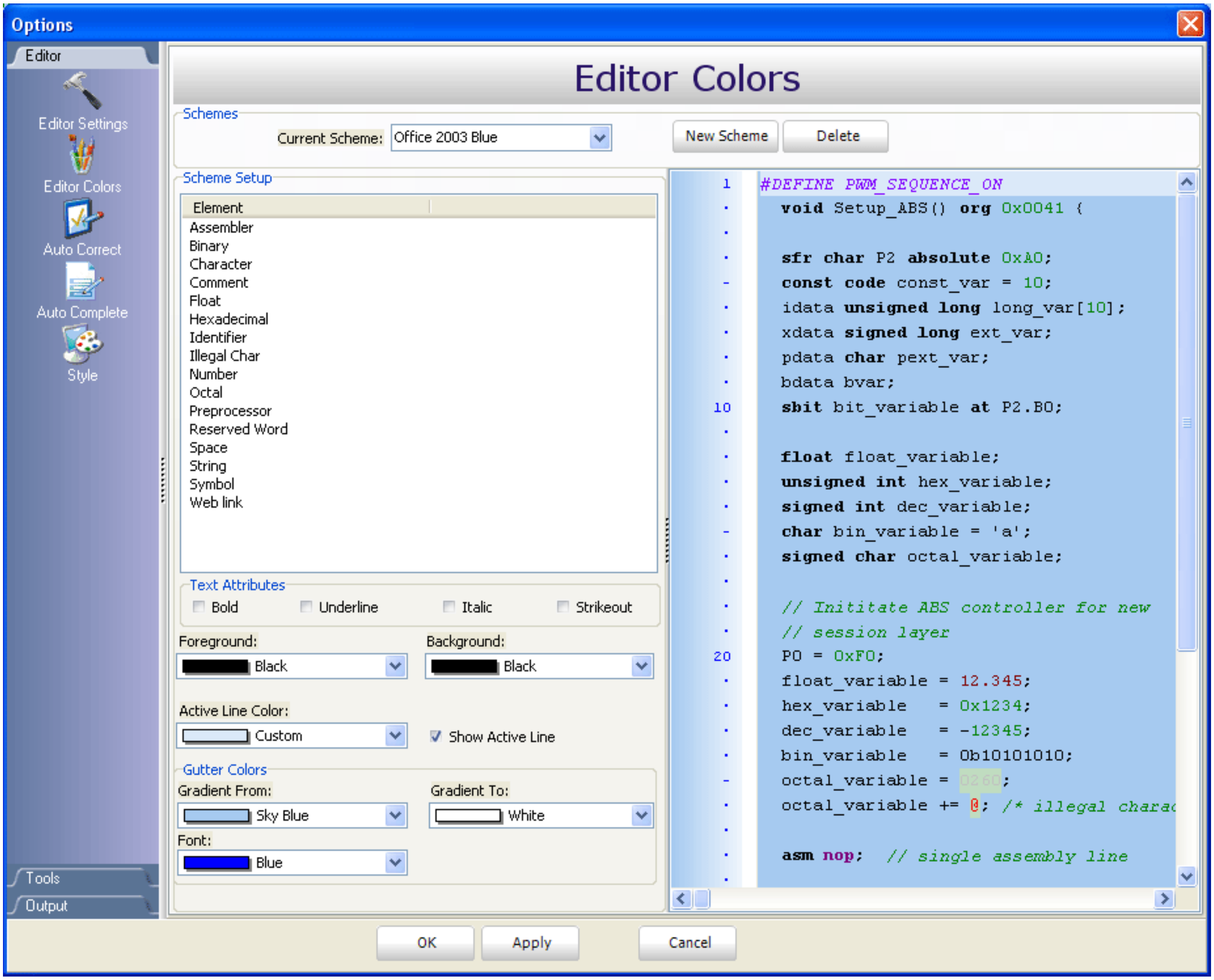
Column Select Mode

This mode changes the operation of the editor for selecting text. When column select mode is used, highlighted text is based on the character column position of the first character selected to the column of the last character of text selected.
Text selected in this mode does not automatically include all text between the start and end position, but includes all text in the columns between the first and last character selected.

Column mode editing is sometimes referred to as *block mode editing* as the act of selecting text forms a rectangle.

To enter this mode, press Alt + Left mouse button, drag the mouse towards the desired direction thus selecting the text.

Editor Colors



Current Scheme: Office 2003 Blue

Element
Assembler
Binary
Character
Comment
Float
Hexadecimal
Identifier
Illegal Char
Number
Octal
Preprocessor
Reserved Word
Space
String
Symbol
Web link

☐ Bold ☐ Underline ☐ Italic ☐ Strikeout

Foreground: Black

Background: Black

Active Line Color: Custom

☑ Show Active Line

Gradient From: Sky Blue

Gradient To: White

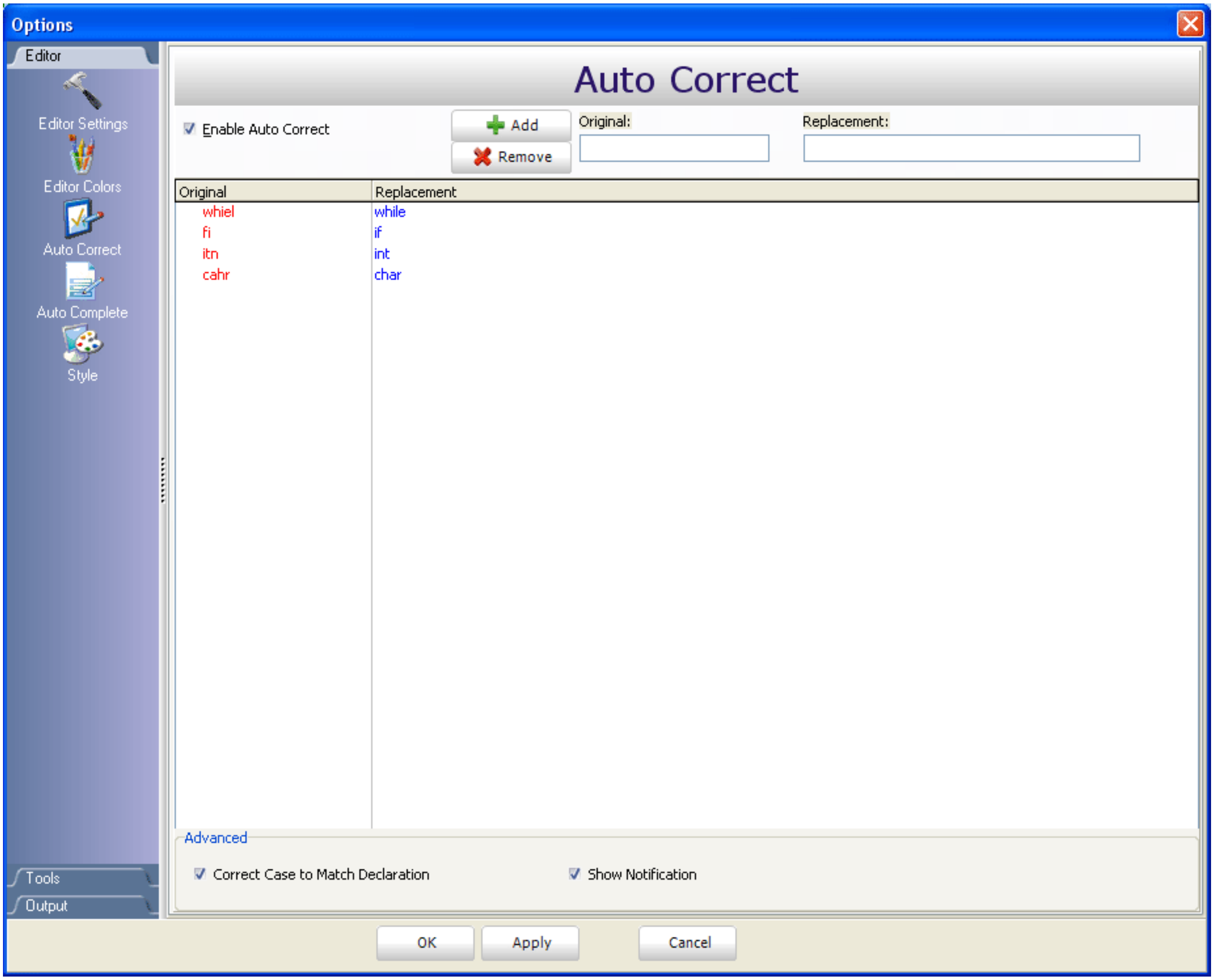
Font: Blue

```
#DEFINE PWM_SEQUENCE_ON
  void Setup_ABS() org 0x0041 {

  sfr char P2 absolute 0xA0;
  const code const_var = 10;
  idata unsigned long long_var[10];
  xdata signed long ext_var;
  pdata char pext_var;
  bdata bvar;
  sbit bit_variable at P2.B0;

  float float_variable;
  unsigned int hex_variable;
  signed int dec_variable;
  char bin_variable = 'a';
  signed char octal_variable;

  // Inititate ABS controller for new
  // session layer
  P0 = 0xF0;
  float_variable = 12.345;
  hex_variable   = 0x1234;
  dec_variable   = -12345;
  bin_variable   = 0b10101010;
  octal_variable = 0260;
  octal_variable += @; /* illegal charac

  asm nop;  // single assembly line
```

Editor Colors option allows user to set, change and save text and color settings organized in schemes. Schemes represent custom graphical appearance that can be applied to GUI (Graphical User Interface) to satisfy tastes of different users.

Auto Correct

Auto Correct option facilitates the user in such a fashion that it automatically corrects common typing or spelling errors as it types.



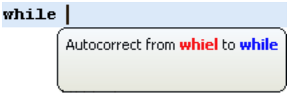
☑ Enable Auto Correct

Original	Replacement
whiel	while
fi	if
itn	int
cahr	char

☑ Correct Case to Match Declaration

☑ Show Notification

This option is already set up to automatically correct some words. For example, if you type `whiel`, it will be corrected to `while` when you press the spacebar:



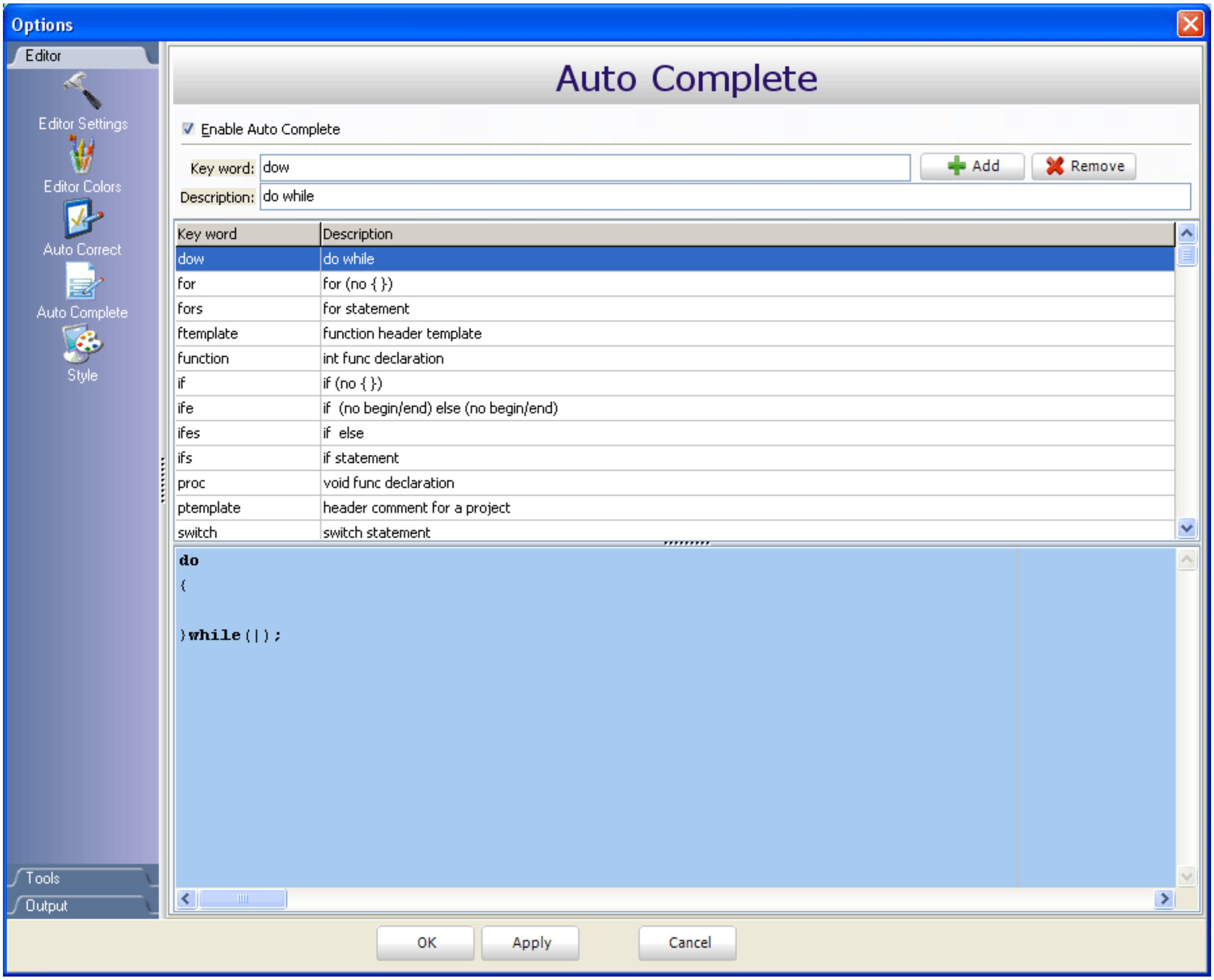
```
while |
```

Autocorrect from **whiel** to **while**

The user can easily add its common typos by entering original typo, for example `btye`, to the Original box, and replacement, `byte`, to the Replacement box, and just click "Add" button.
Next time when the typo occurs, it will be automatically corrected.

Auto Complete (Code Templates)

Auto Complete option saves lots of keystrokes for commonly used phrases by automatically completing user's typing.



☑ Enable Auto Complete

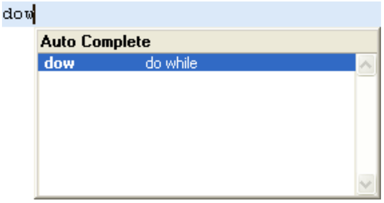
Key word: dow

Description: do while

Key word	Description
dow	do while
for	for (no { })
fors	for statement
ftemplate	function header template
function	int func declaration
if	if (no { })
ife	if (no begin/end) else (no begin/end)
ifes	if else
ifs	if statement
proc	void func declaration
ptemplate	header comment for a project
switch	switch statement

```
do
{

}while(|);
```

The user can insert the Code Template by typing the name of the template (for instance, `dow`), then press Ctrl+J and the Code Editor will automatically generate a code:



dow

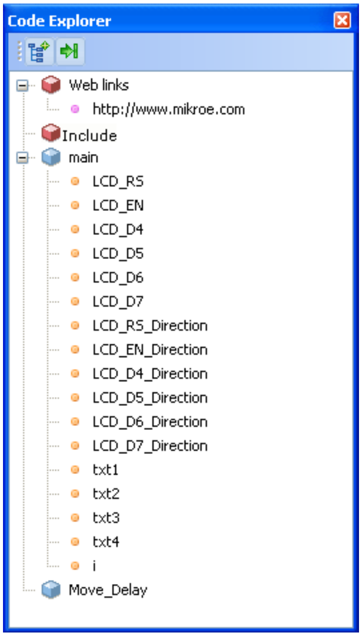
You can add your own templates to the list by entering the desired keyword, description and code of your template in appropriate boxes.
Autocomplete macros can retreive system and project information:

- `%DATE%` - current system date
- `%TIME%` - current system time
- `%DEVICE%` - device (MCU) name as specified in project settings
- `%DEVICE_CLOCK%` - clock as specified in project settings
- `%COMPILER%` - current compiler version

These macros can be used in template code, see template `ptemplate` provided with mikroBasic PRO for dsPIC30/33 and PIC24 installation.

Code Explorer

The Code Explorer gives clear view of each item declared inside the source code. You can jump to a declaration of any item by double clicking it, or pressing the Enter button. Also, besides the list of defined and declared objects, code explorer displays message about the first error and it's location in code.



Code Explorer

- Web links
  - http://www.mikroe.com
- Include
- main
  - LCD_RS
  - LCD_EN
  - LCD_D4
  - LCD_D5
  - LCD_D6
  - LCD_D7
  - LCD_RS_Direction
  - LCD_EN_Direction
  - LCD_D4_Direction
  - LCD_D5_Direction
  - LCD_D6_Direction
  - LCD_D7_Direction
  - txt1
  - txt2
  - txt3
  - txt4
  - i
- Move_Delay

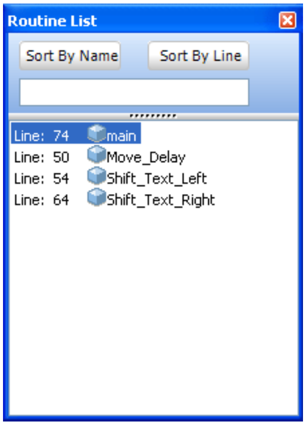
The following options are available in the Code Explorer:

Icon	Description
⊞	Expand/Collapse all nodes in tree.
➡	Locate declaration in code.

Routine List

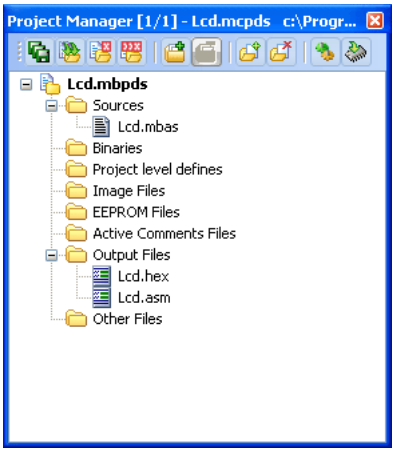
Routine list diplays list of routines, and enables filtering routines by name. Routine list window can be accessed by pressing Ctrl+L.

You can jump to a desired routine by double clicking on it, or pressing the Enter button. Also, you can sort routines by size or by address.



Project Manager

Project Manager is IDE feature which allows the users to manage multiple projects. Several projects which together make project group may be open at the same time. Only one of them may be active at the moment.
Setting project in **active** mode is performed by **double clicking** the desired project in the Project Manager, which will result in bolding the project's name.
Also, the name of the currently active project will be diplayed in the Program Manager window title, alongside with the number of projects in project group.



Following options are available in the Project Manager:

Icon	Description
	Save project Group.
	Open project group.
	Close the active project.
	Close project group.
	Add project to the project group.
	Remove project from the project group.
	Add file to the active project.
	Remove selected file from the project.
	Build the active project.
	Run mikroElektronika's Flash programmer.

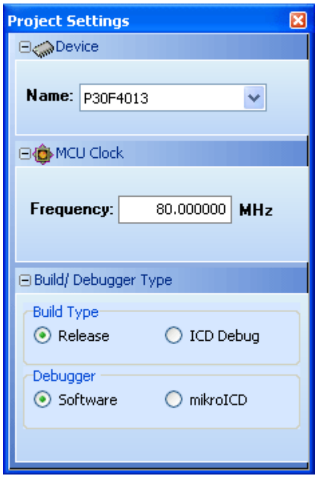
For details about adding and removing files from project see Add/Remove Files from Project.

Related topics: Project Settings, Project Menu Options, File Menu Options, Project Toolbar, Build Toolbar, Add/Remove Files from Project

Project Settings

The following options are available in the Project Settings window:

- Device - select the appropriate device from the device drop-down list.
- MCU Clock - enter the clock frequency value.
- Build/Debugger Type - choose debugger type.



Project Settings

Device

Name: P30F4013

MCU Clock

Frequency: 80.000000 **MHz**

Build/ Debugger Type

Build Type

● Release ○ ICD Debug

Debugger

● Software ○ mikroICD

Related topics: Edit Project, Customizing Projects, Project Manager

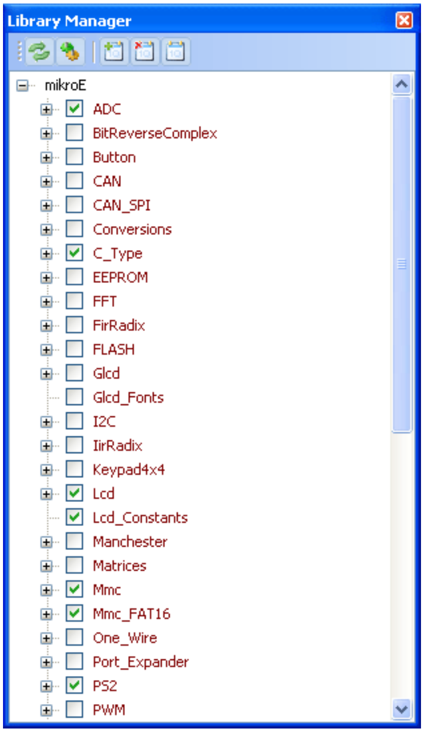
Library Manager

Library Manager enables simple handling libraries being used in a project. Library Manager window lists all libraries (extension `.mcl`) which are instantly stored in the compiler *Uses* folder. The desirable library is added to the project by selecting check box next to the library name.

In order to have all library functions accessible, simply press the button **Check All** and all libraries will be selected.

In case none library is needed in a project, press the button **Clear All** and all libraries will be cleared from the project.

Only the selected libraries will be linked.



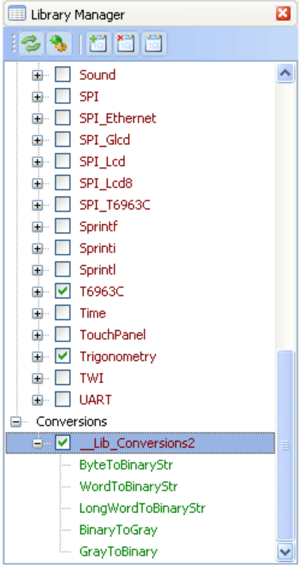
- mikroE
  - ☑ ADC
  - ☐ BitReverseComplex
  - ☐ Button
  - ☐ CAN
  - ☐ CAN_SPI
  - ☐ Conversions
  - ☑ C_Type
  - ☐ EEPROM
  - ☐ FFT
  - ☐ FirRadix
  - ☐ FLASH
  - ☐ Glcd
  - ☐ Glcd_Fonts
  - ☐ I2C
  - ☐ IirRadix
  - ☐ Keypad4x4
  - ☑ Lcd
  - ☑ Lcd_Constants
  - ☐ Manchester
  - ☐ Matrices
  - ☑ Mmc
  - ☑ Mmc_FAT16
  - ☐ One_Wire
  - ☐ Port_Expander
  - ☑ PS2
  - ☐ PWM

Icon	Description
⟳	Refresh Library by scanning files in “Uses” folder.Useful when new libraries are added by copying files to “Uses” folder.
⚒	Rebuild all available libraries. Useful when library sources are available and need refreshing.
+	Include all available libraries in current project.
✕	No libraries from the list will be included in current project.
↺	Restore library to the state just before last project saving.

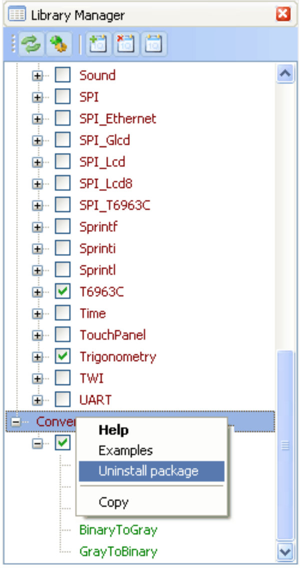
Managing libraries using Package Manager

The Package Manager is a tool which enables users to easily install their own libraries in the mikroIDE. Libraries are distributed in the form of a package, which is an archive composed of one or more files, containing libraries. For more information on Package Manager, visit our website.

Upon package installation, a new node with the package name will be created in the Library Manager. For example:



From the Library Manager, the user can also uninstall the desired package by right clicking the the appropriate node, and from the drop-down menu choose Uninstall package:

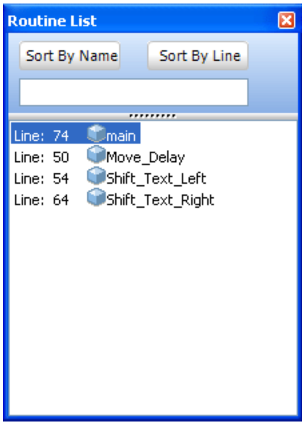


Related topics: mikroBasic PRO for PIC Libraries, Creating New Library

Routine List

Routine list diplays list of routines, and enables filtering routines by name. Routine list window can be accessed by pressing Ctrl+L.

You can jump to a desired routine by double clicking on it, or pressing the Enter button. Also, you can sort routines by size or by address.



```
Line: 74  main
Line: 50  Move_Delay
Line: 54  Shift_Text_Left
Line: 64  Shift_Text_Right
```

Statistics

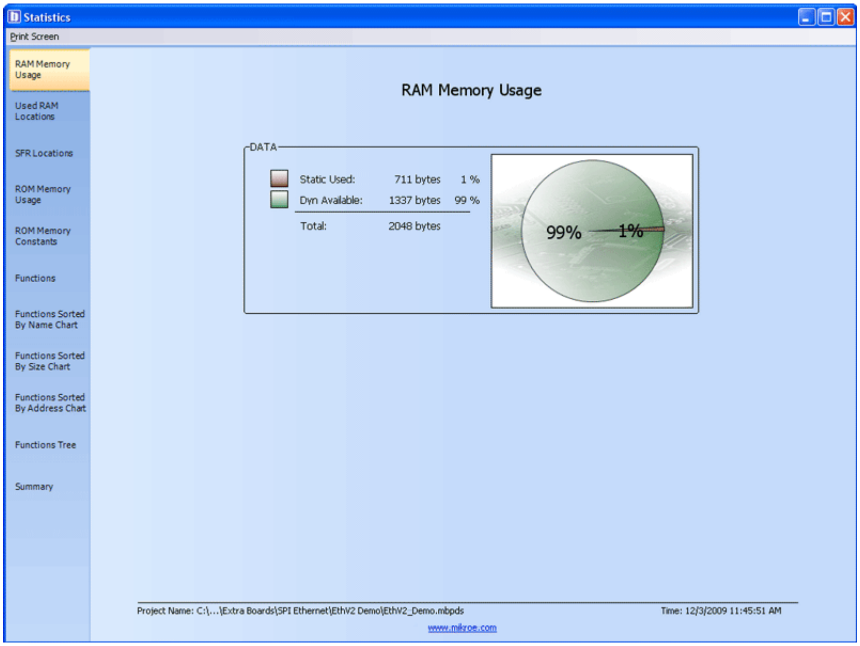
After successful compilation, you can review statistics of your code. Click the Statistics Icon .

Memory Usage Windows

Provides overview of RAM and ROM usage in the various forms.

RAM Memory Usage

Displays RAM memory usage in a pie-like form.



RAM Memory Usage

Static Used:	711 bytes	1 %
Dyn Available:	1337 bytes	99 %
Total:	2048 bytes	

Project Name: C:\...\Extra Boards\SPI Ethernet\EthV2 Demo\EthV2_Demo.mbpds

Time: 12/3/2009 11:45:51 AM

www.mikroe.com

Used RAM Locations

Displays used RAM memory locations and their names.

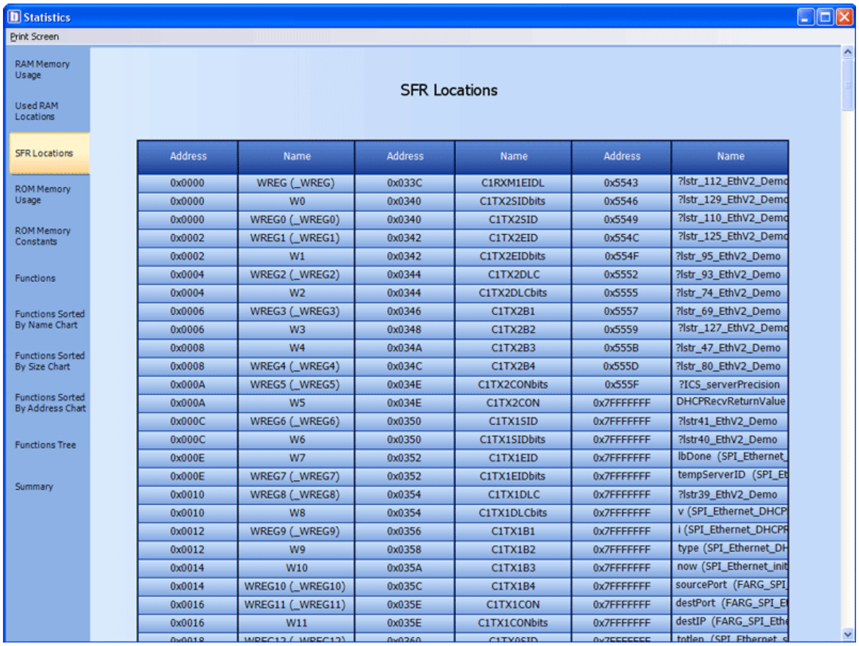
Used RAM Locations

* Click on Name column header to toggle between Name and Unique Assembler Name

Address	Name	Address	Name	Address	Name
0x0000	WREG	0x7FFFFFFF	dl	0x7FFFFFFF	c
0x0000	W0	0x7FFFFFFF	type	0x7FFFFFFF	?lstr130_EthV2_Demo
0x0000	WREG0	0x7FFFFFFF	a	0x7FFFFFFF	?FLOC__SPI_Ethernet_UserUDP
0x0002	WREG1	0x7FFFFFFF	f	0x7FFFFFFF	buffer
0x0002	W1	0x7FFFFFFF	q	0x7FFFFFFF	?lstr132_EthV2_Demo
0x0004	WREG2	0x7FFFFFFF	align	0x7FFFFFFF	?lstr131_EthV2_Demo
0x0004	W2	0x7FFFFFFF	len	0x7FFFFFFF	loc_word
0x0006	WREG3	0x7FFFFFFF	tx	0x7FFFFFFF	reqLength
0x0006	W3	0x7FFFFFFF	tcpFlags	0x7FFFFFFF	out_char
0x0008	W4	0x7FFFFFFF	port	0x7FFFFFFF	i
0x0008	WREG4	0x7FFFFFFF	remotePort	0x7FFFFFFF	tts
0x000A	WREG5	0x7FFFFFFF	swap	0x7FFFFFFF	in
0x000A	W5	0x7FFFFFFF	syn	0x7FFFFFFF	jd
0x000C	WREG6	0x7FFFFFFF	transmit	0x7FFFFFFF	I
0x000C	W6	0x7FFFFFFF	datalen	0x7FFFFFFF	ts
0x000E	W7	0x7FFFFFFF	replen	0x7FFFFFFF	out
0x000E	WREG7	0x7FFFFFFF	start	0x7FFFFFFF	e
0x0010	WREG8	0x7FFFFFFF	ipHeaderLen	0x7FFFFFFF	N
0x0010	W8	0x7FFFFFFF	m	0x7FFFFFFF	?FLOC__Time_epochToDate
0x0012	WREG9	0x7FFFFFFF	?lstr3___Lib_EthEnc28j60	0x7FFFFFFF	L
0x0012	W9	0x7FFFFFFF	?lstr4___Lib_EthEnc28j60	0x7FFFFFFF	J
0x0014	W10	0x7FFFFFFF	align	0x7FFFFFFF	K
0x0014	WREG10	0x7FFFFFFF	packetEndAddr	0x7FFFFFFF	found
0x0016	WREG11	0x7FFFFFFF	l	0x7FFFFFFF	s1
0x0016	W11	0x7FFFFFFF	payloadAddr	0x7FFFFFFF	character

SFR Locations

Displays list of used SFR locations.

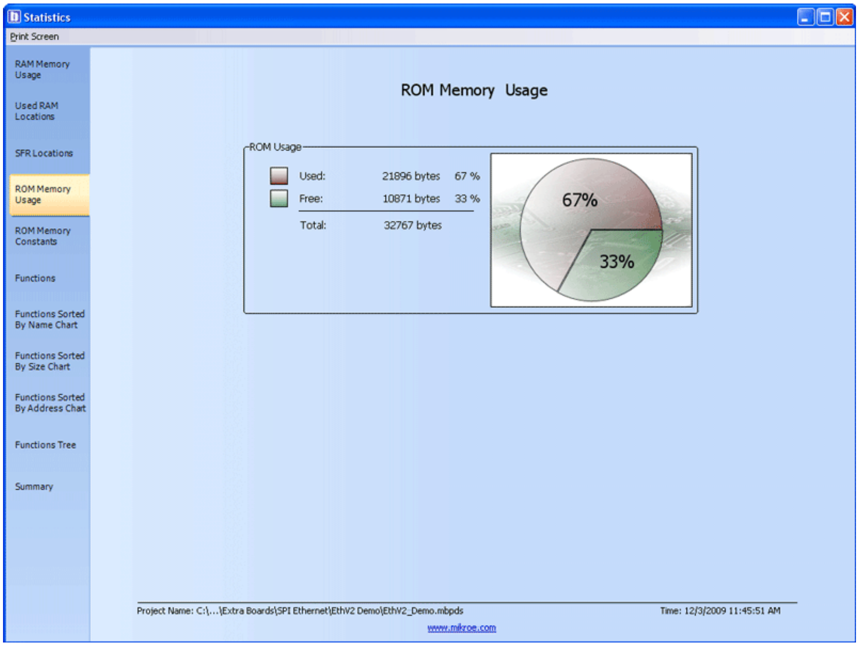


SFR Locations

Address	Name	Address	Name	Address	Name
0x0000	WREG (_WREG)	0x033C	C1RXM1EIDL	0x5543	?lstr_112_EthV2_Demo
0x0000	W0	0x0340	C1TX2SIDbits	0x5546	?lstr_129_EthV2_Demo
0x0000	WREG0 (_WREG0)	0x0340	C1TX2SID	0x5549	?lstr_110_EthV2_Demo
0x0002	WREG1 (_WREG1)	0x0342	C1TX2EID	0x554C	?lstr_125_EthV2_Demo
0x0002	W1	0x0342	C1TX2EIDbits	0x554F	?lstr_95_EthV2_Demo
0x0004	WREG2 (_WREG2)	0x0344	C1TX2DLC	0x5552	?lstr_93_EthV2_Demo
0x0004	W2	0x0344	C1TX2DLCbits	0x5555	?lstr_74_EthV2_Demo
0x0006	WREG3 (_WREG3)	0x0346	C1TX2B1	0x5557	?lstr_69_EthV2_Demo
0x0006	W3	0x0348	C1TX2B2	0x5559	?lstr_127_EthV2_Demo
0x0008	W4	0x034A	C1TX2B3	0x555B	?lstr_47_EthV2_Demo
0x0008	WREG4 (_WREG4)	0x034C	C1TX2B4	0x555D	?lstr_80_EthV2_Demo
0x000A	WREG5 (_WREG5)	0x034E	C1TX2CONbits	0x555F	?ICS_serverPrecision
0x000A	W5	0x034E	C1TX2CON	0x7FFFFFFF	DHCPRecvReturnValue
0x000C	WREG6 (_WREG6)	0x0350	C1TX1SID	0x7FFFFFFF	?lstr41_EthV2_Demo
0x000C	W6	0x0350	C1TX1SIDbits	0x7FFFFFFF	?lstr40_EthV2_Demo
0x000E	W7	0x0352	C1TX1EID	0x7FFFFFFF	lbDone (SPI_Ethernet_
0x000E	WREG7 (_WREG7)	0x0352	C1TX1EIDbits	0x7FFFFFFF	tempServerID (SPI_Et
0x0010	WREG8 (_WREG8)	0x0354	C1TX1DLC	0x7FFFFFFF	?lstr39_EthV2_Demo
0x0010	W8	0x0354	C1TX1DLCbits	0x7FFFFFFF	v (SPI_Ethernet_DHCP
0x0012	WREG9 (_WREG9)	0x0356	C1TX1B1	0x7FFFFFFF	i (SPI_Ethernet_DHCP
0x0012	W9	0x0358	C1TX1B2	0x7FFFFFFF	type (SPI_Ethernet_DH
0x0014	W10	0x035A	C1TX1B3	0x7FFFFFFF	now (SPI_Ethernet_init
0x0014	WREG10 (_WREG10)	0x035C	C1TX1B4	0x7FFFFFFF	sourcePort (FARG_SPI_
0x0016	WREG11 (_WREG11)	0x035E	C1TX1CON	0x7FFFFFFF	destPort (FARG_SPI_E
0x0016	W11	0x035E	C1TX1CONbits	0x7FFFFFFF	destIP (FARG_SPI_Ethe
0x0018	WREG12 (_WREG12)	0x0360	C1TX0SID	0x7FFFFFFF	totlen (SPI_Ethernet_s

ROM Memory Usage

Displays ROM memory space usage in a pie-like form.



ROM Memory Usage

ROM Usage

🟥	Used:	21896 bytes	67 %
🟩	Free:	10871 bytes	33 %
	Total:	32767 bytes	

Project Name: C:\...\Extra Boards\SPI Ethernet\EthV2 Demo\EthV2_Demo.mbpds

Time: 12/3/2009 11:45:51 AM

www.mikroe.com

ROM Memory Constants

Displays ROM memory constants and their addresses.

ROM Memory Constants

Address	Name
0x5483	?ICS?lstr1___Lib_EthEnc28j60
0x529C	?ICS?lstr1_EthV2_Demo
0x5310	?ICS?lstr1_httpUtils
0x51ED	?ICS?lstr10_EthV2_Demo
0x4C19	?ICS?lstr101_EthV2_Demo
0x4C22	?ICS?lstr102_EthV2_Demo
0x4C33	?ICS?lstr103_EthV2_Demo
0x4C45	?ICS?lstr104_EthV2_Demo
0x4C4C	?ICS?lstr105_EthV2_Demo
0x4C53	?ICS?lstr106_EthV2_Demo
0x4C5D	?ICS?lstr107_EthV2_Demo
0x4C7E	?ICS?lstr108_EthV2_Demo
0x51E9	?ICS?lstr11_EthV2_Demo
0x521C	?ICS?lstr12_EthV2_Demo
0x520B	?ICS?lstr13_EthV2_Demo
0x5259	?ICS?lstr130_EthV2_Demo
0x526A	?ICS?lstr131_EthV2_Demo
0x527B	?ICS?lstr132_EthV2_Demo
0x5214	?ICS?lstr14_EthV2_Demo
0x5220	?ICS?lstr15_EthV2_Demo
0x51E5	?ICS?lstr16_EthV2_Demo
0x4E4B	?ICS?lstr17_EthV2_Demo
0x4E54	?ICS?lstr18_EthV2_Demo
0x4E50	?ICS?lstr19_EthV2_Demo
0x5356	?ICS?lstr2___Lib_EthEnc28j60
0x4E58	?ICS?lstr2_EthV2_Demo

Functions

Sorts and displays functions in various ways.

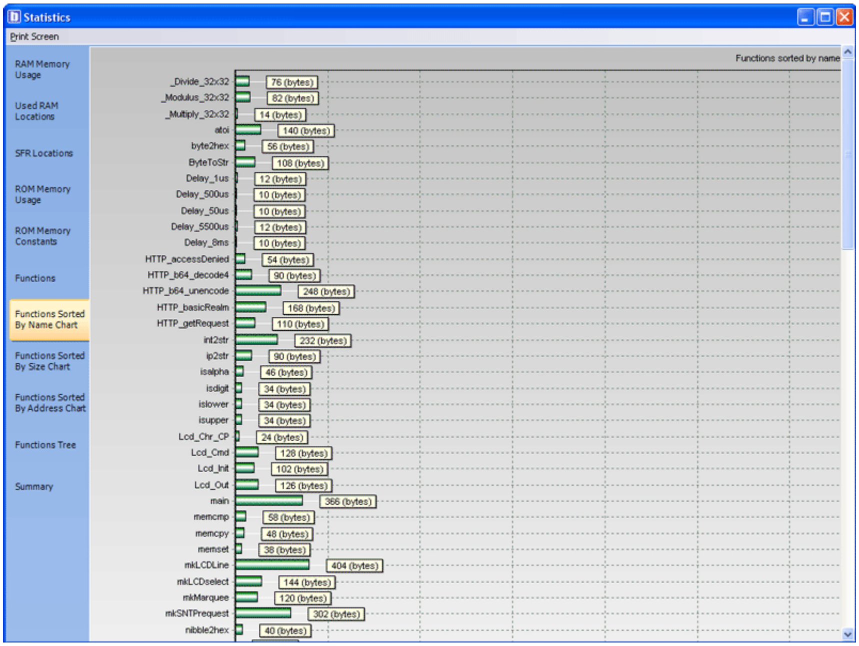
Functions Sorted By Size

* Click on column header to sort table by Address, Name, Unigue Assembler Name or Size

Address Asc	Name	Unique Assembler Name	Size (bytes)
0x084C	saveConf	_saveConf	2
0x3BF6	SPI_Ethernet_getIpAddress	_SPI_Ethernet_getIpAddress	4
0x3D4E	SPI_Ethernet_getIpMask	_SPI_Ethernet_getIpMask	4
0x3CCC	SPI_Ethernet_getDnsIpAddress	_SPI_Ethernet_getDnsIpAddress	4
0x3CC8	SPI_Ethernet_getGwIpAddress	_SPI_Ethernet_getGwIpAddress	4
0x340E	Delay_50us	_Delay_50us	10
0x319E	Strobe	__Lib_Lcd_Strobe	10
0x2A7A	Delay_500us	_Delay_500us	10
0x33D6	Delay_8ms	_Delay_8ms	10
0x2A6E	Delay_1us	_Delay_1us	12
0x3418	Delay_5500us	_Delay_5500us	12
0x0358	_Multiply_32x32	__Multiply_32x32	14
0x31F6	Lcd_Chr_CP	_Lcd_Chr_CP	24
0x3184	SPI1_Read	_SPI1_Read	26
0x3424	SPI_Ethernet_delay	_SPI_Ethernet_delay	30
0x0FE6	SPI_Ethernet_writeMemory	_SPI_Ethernet_writeMemory	30
0x0E8C	SPI_Ethernet_setRxReadAddress	_SPI_Ethernet_setRxReadAddress	32
0x0366	strlen	_strlen	32
0x323E	strchr	_strchr	34
0x2ACC	islower	_islower	34
0x2A84	isupper	_isupper	34
0x0386	isdigit	_isdigit	34
0x043A	strcpy	_strcpy	36
0x104A	SPI_Ethernet_MACswap	_SPI_Ethernet_MACswap	38

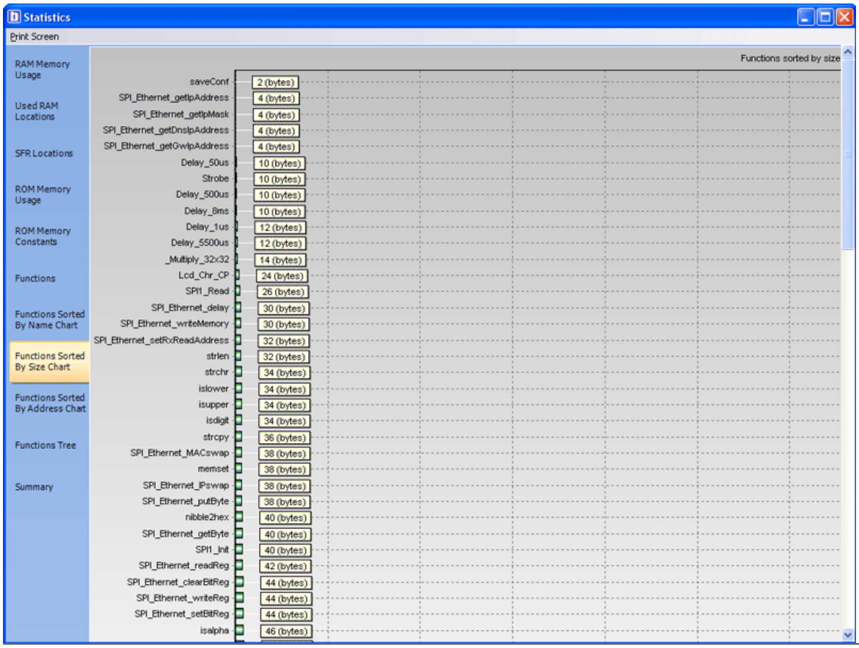
Functions Sorted By Name Chart

Sorts and displays functions by their name, in the ascending order.



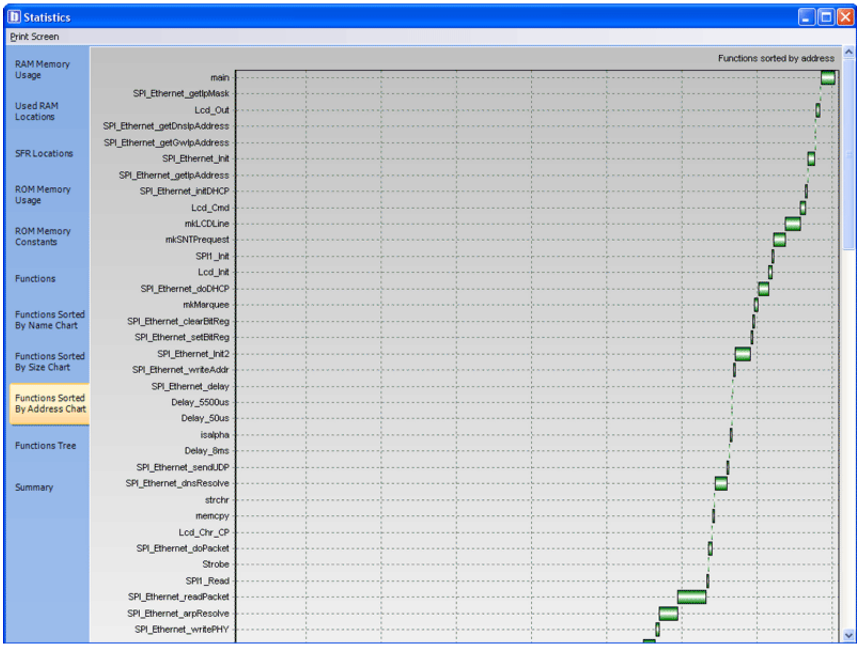
Functions Sorted By Size Chart

Sorts and displays functions by their sizes in a chart-like form.



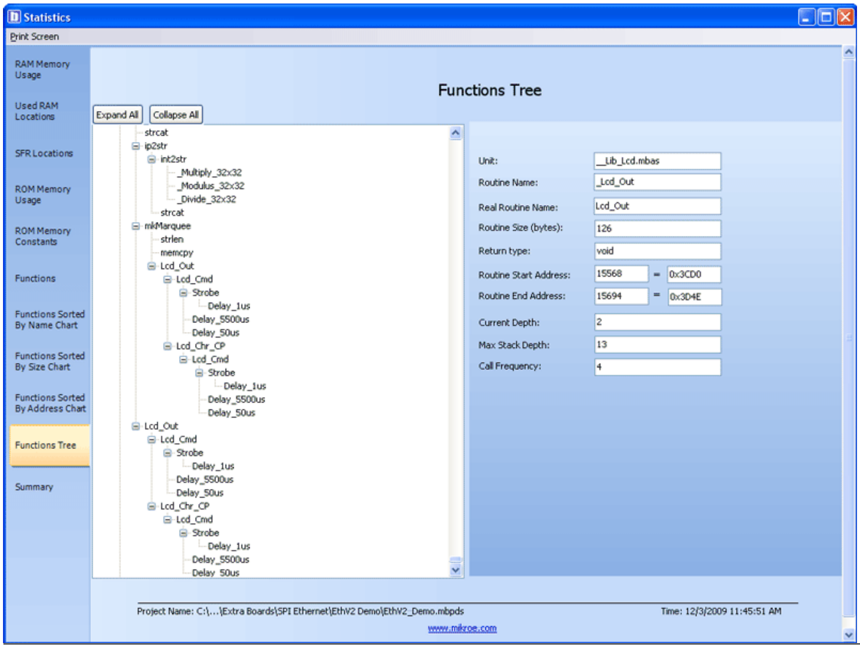
Functions Sorted By Addresses

Sorts and displays functions by their addresses, in the ascending order.



Function Tree

Displays Function Tree with the relevant data for each function.



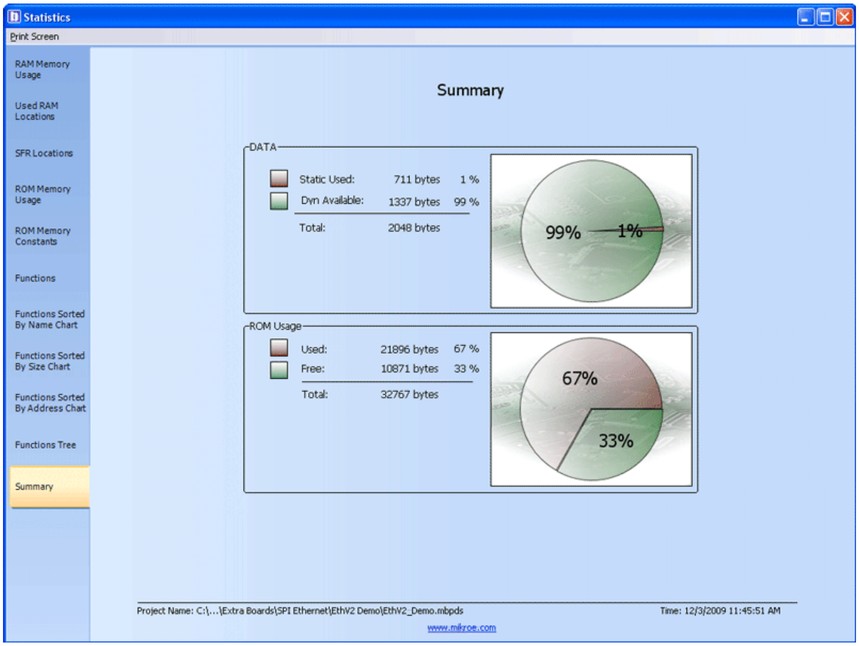
```
  strcat
⊟ ip2str
  ⊟ int2str
      _Multiply_32x32
      _Modulus_32x32
      _Divide_32x32
    strcat
⊟ mkMarquee
    strlen
    memcpy
  ⊟ Lcd_Out
    ⊟ Lcd_Cmd
      ⊟ Strobe
          Delay_1us
        Delay_5500us
        Delay_50us
    ⊟ Lcd_Chr_CP
      ⊟ Lcd_Cmd
        ⊟ Strobe
            Delay_1us
          Delay_5500us
          Delay_50us
⊟ Lcd_Out
  ⊟ Lcd_Cmd
    ⊟ Strobe
        Delay_1us
      Delay_5500us
      Delay_50us
  ⊟ Lcd_Chr_CP
    ⊟ Lcd_Cmd
      ⊟ Strobe
          Delay_1us
        Delay_5500us
        Delay_50us
```

Unit: __Lib_Lcd.mbas
Routine Name: _Lcd_Out
Real Routine Name: Lcd_Out
Routine Size (bytes): 126
Return type: void
Routine Start Address: 15568 = 0x3CD0
Routine End Address: 15694 = 0x3D4E
Current Depth: 2
Max Stack Depth: 13
Call Frequency: 4

Project Name: C:\...\Extra Boards\SPI Ethernet\EthV2 Demo\EthV2_Demo.mbpds
Time: 12/3/2009 11:45:51 AM
www.mikroe.com

Memory Summary

Displays summary of RAM and ROM memory in a pie-like form.



Summary

DATA

🟥	Static Used:	711 bytes	1 %
🟩	Dyn Available:	1337 bytes	99 %
	Total:	2048 bytes	

ROM Usage

🟥	Used:	21896 bytes	67 %
🟩	Free:	10871 bytes	33 %
	Total:	32767 bytes	

Project Name: C:\...\Extra Boards\SPI Ethernet\EthV2 Demo\EthV2_Demo.mbpds

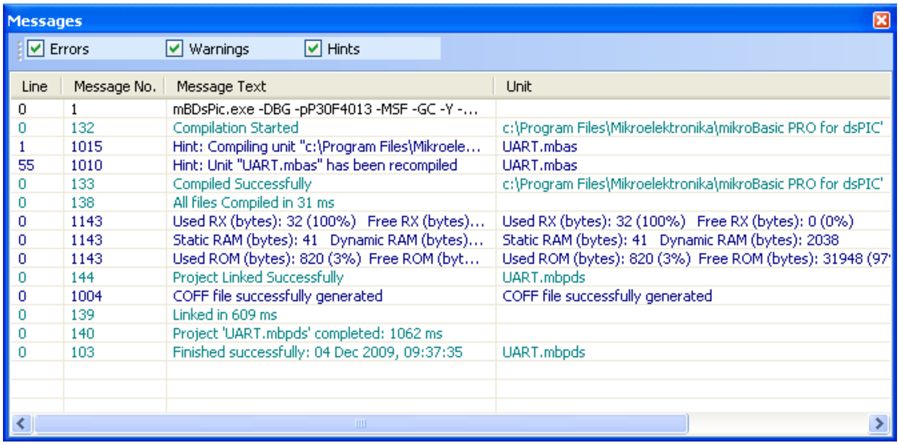
Time: 12/3/2009 11:45:51 AM

Messages Window

Messages Window displays various informations and notifications about the compilation process.

It reports for example, time needed for preprocessing, compilation and linking; used RAM and ROM space, generated baud rate with error percentage, etc.
The user can filter which notifications will Messages Window display by checking Errors, Warning and Hints box.

In case that errors were encountered during compiling, the compiler will report them and won't generate a hex file. The Messages Window will display errros at the bottom of the window by default.
The compiler also reports warnings, but these do not affect the output; only errors can interefere with the generation of hex.

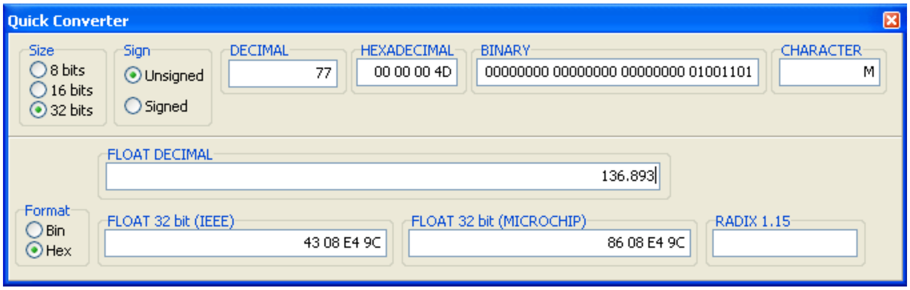


Line	Message No.	Message Text	Unit
0	1	mBDsPic.exe -DBG -pP30F4013 -MSF -GC -Y -...	
0	132	Compilation Started	c:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC'
1	1015	Hint: Compiling unit "c:\Program Files\Mikroele...	UART.mbas
55	1010	Hint: Unit "UART.mbas" has been recompiled	UART.mbas
0	133	Compiled Successfully	c:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC'
0	138	All files Compiled in 31 ms	
0	1143	Used RX (bytes): 32 (100%) Free RX (bytes)...	Used RX (bytes): 32 (100%) Free RX (bytes): 0 (0%)
0	1143	Static RAM (bytes): 41 Dynamic RAM (bytes)...	Static RAM (bytes): 41 Dynamic RAM (bytes): 2038
0	1143	Used ROM (bytes): 820 (3%) Free ROM (byt...	Used ROM (bytes): 820 (3%) Free ROM (bytes): 31948 (97'
0	144	Project Linked Successfully	UART.mbpds
0	1004	COFF file successfully generated	COFF file successfully generated
0	139	Linked in 609 ms	
0	140	Project 'UART.mbpds' completed: 1062 ms	
0	103	Finished successfully: 04 Dec 2009, 09:37:35	UART.mbpds

Double click the message line in the Message Window to highlight the line where the error was encountered.

Quick Converter

Quick Converter enables the user to easily transform numbers from one base to another.

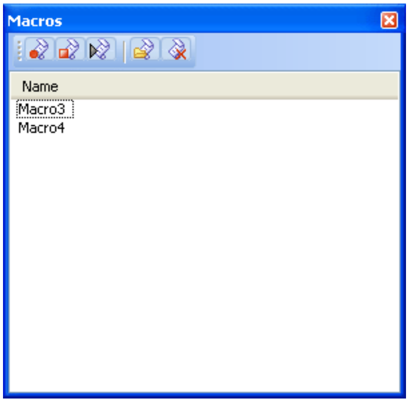


Size: ○ 8 bits ○ 16 bits ● 32 bits
Sign: ● Unsigned ○ Signed
DECIMAL: 77
HEXADECIMAL: 00 00 00 4D
BINARY: 00000000 00000000 00000000 01001101
CHARACTER: M
FLOAT DECIMAL: 136.893
Format: ○ Bin ● Hex
FLOAT 32 bit (IEEE): 43 08 E4 9C
FLOAT 32 bit (MICROCHIP): 86 08 E4 9C
RADIX 1.15:

The user can convert integers of various sizes (8, 16 or 32 bits), signed and unsigned, using different representation (decimal, hexadecimal, binary and character).
Also, Quick Converter features float point numbers conversion from/to Float Decimal, Float 32bit (IEEE), Float 32bit (Microchip) and Radix 1.15 for dsPIC family of MCUs.

Macro Editor

A macro is a series of keystrokes that have been 'recorded' in the order performed. A macro allows you to 'record' a series of keystrokes and then 'playback', or repeat, the recorded keystrokes.



Macro3
Macro4

The Macro offers the following commands:

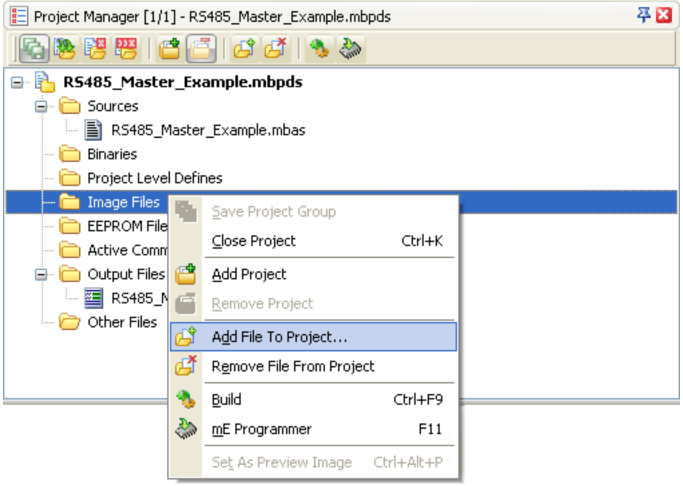
Icon	Description
●	Starts ‘recording’ keystrokes for later playback.
■	Stops capturing keystrokes that was started when the Start Recording command was selected.
▶	Allows a macro that has been recorded to be replayed.
📂	New macro.
✕	Delete macro.

Related topics: Code Editor, Code Templates

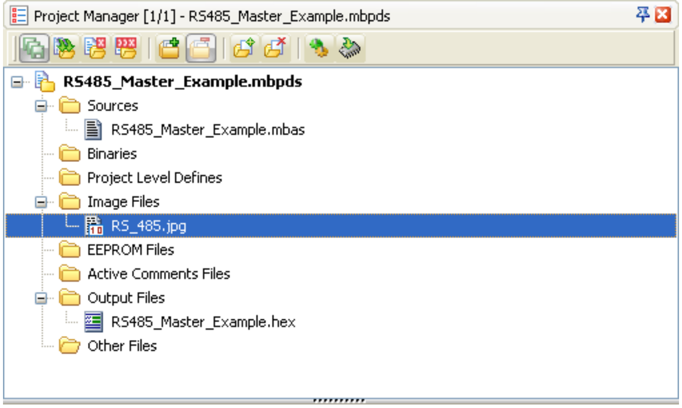
Image Preview

There are a lot of occassions in which the user besides the code, must look at the appropriate schematics in order to succesfully write the desired program.
The mikroBasic PRO for dsPIC30/33 and PIC24 provides this possibility through the **Image Preview Window**.

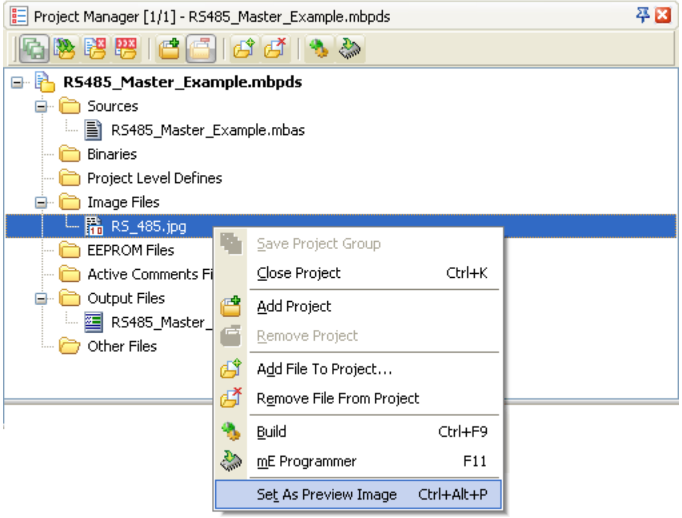
To add an image to the **Image Preview Window**, right click the **Image Files** node in the **Project Manager**:



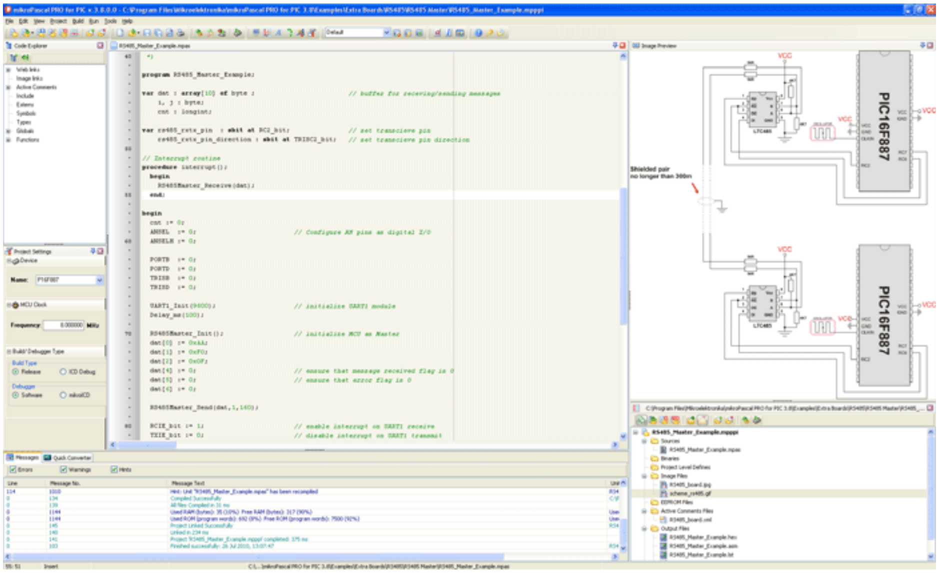
Now, navigate to the desired image file, and simply add it:



Next, right click the added file, and choose **Set As Preview Image**:



Once you have added the image, it will appear in the **Image Preview Window**:



```
*)

program RS485_Master_Example;

var dat : array[10] of byte ;                    // buffer for receving/sending messages
    i, j : byte;
    cnt : longint;

var rs485_rxtx_pin  : sbit at RC2_bit;           // set transciever pin
    rs485_rxtx_pin_direction : sbit at TRISC2_bit;   // set transciever pin direction

// Interrupt routine
procedure interrupt();
  begin
    RS485Master_Receive(dat);
  end;

begin
  cnt := 0;
  ANSEL  := 0;                    // Configure AN pins as digital I/O
  ANSELH := 0;

  PORTB  := 0;
  PORTD  := 0;
  TRISB  := 0;
  TRISD  := 0;

  UART1_Init(9600);               // initialize UART1 module
  Delay_ms(100);

  RS485Master_Init();             // initialize MCU as Master
  dat[0] := 0xAA;
  dat[1] := 0xF0;
  dat[2] := 0x0F;
  dat[4] := 0;                    // ensure that message received flag is 0
  dat[5] := 0;                    // ensure that error flag is 0
  dat[6] := 0;

  RS485Master_Send(dat,1,160);

  RCIE_bit := 1;                  // enable interrupt on UART1 receive
  TXIE_bit := 0;                  // disable interrupt on UART1 transmit
```

Also, you can add multiple images to the **Image Files** node, but only the one that is set will be automatically displayed in the **Image Preview Window** upon opening the project.

By changing the **Image Preview Window** size, displayed image will be fit by its height in such a way that its proportions will remain intact.

Toolbars

This section provides an overview of the toolbars available in mikroBasic PRO for dsPIC30/33 and PIC24 Help:

- File Toolbar
- Edit Toolbar
- Advanced Edit Toolbar
- Find Toolbar
- Project Toolbar
- Build Toolbar
- Debug Toolbar
- Styles Toolbar
- Tools Toolbar
- View Toolbar
- Layout Toolbar
- Help Toolbar

File Toolbar

File Toolbar is a standard toolbar with the following options:

Icon	Description
🗋	Opens a new editor window.
📂	Open source file for editing or image file for viewing.
💾	Save changes for active window.
💾	Save changes in all opened windows.
🔍	Print Preview.
🖨	Print.

Edit Toolbar

Edit Toolbar is a standard toolbar with the following options:

Icon	Description
↶	Undo last change.
↷	Redo last change.
✂	Cut selected text to clipboard.
⎘	Copy selected text to clipboard.
📋	Paste text from clipboard.

Advanced Edit Toolbar

Advanced Edit Toolbar comes with the following options:

Icon	Description
{..}	Comment selected code or put a single line comment if there is no selection
{..}	Uncomment selected code or remove single line comment if there is no selection.
BEGI END	Select text from starting delimiter to ending delimiter.
BEGI END	Go to ending delimiter.
⇒	Go to line.
⇥	Indent selected code lines.
⇤	Outdent selected code lines.
HTML	Generate HTML code suitable for publishing current source code on the web.

Find/Replace Toolbar

Find/Replace Toolbar is a standard toolbar with the following options:

Icon	Description
🔍	Find text in current editor.
🔍	Find next occurence.
🔍	Find previous occurence.
R	Replace text.
🗋	Find text in files.

Project Toolbar

Project Toolbar comes with the following options:

Icon	Description
	New project.
	Open Project
	Save Project
	Edit project settings.
	Close current project.
	Clean project folder.
	Add File To Project
	Remove File From Project

Build Toolbar

Build Toolbar comes with the following options:

Icon	Description
	Build current project.
	Build all opened projects.
	Build and program active project.
	Start programmer and load current HEX file.

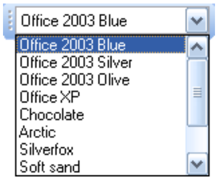
Debug Toolbar

Debug Toolbar comes with the following options:

Icon	Description
	Start Software Simulator or mikroICD (In-Circuit Debugger).
	Run/Pause Debugger.
	Stop Debugger.
	Step Into.
	Step Over.
	Step Out.
	Run To Cursor.
	Toggle Breakpoint.
	View Breakpoints Window
	Clear Breakpoints.
	View Watch Window
	View Stopwatch Window

Styles Toolbar

Styles toolbar allows you to easily change colors of your workspace.



Tools Toolbar

Tools Toolbar comes with the following default options:

Icon	Description
🖥	Run USART Terminal
	EEPROM
A	ASCII Chart
	Seven Segment Editor.
	Open Active Comment editor.
	Options menu

Tip : The Tools toolbar can easily be customized by adding new tools in Options menu window.

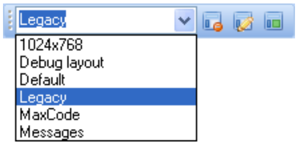
View Toolbar

View Toolbar provides access to assembly code, listing file and statistics windows.

Icon	Description
a	Open assembly code in editor.
l	Open listing file in editor.
	View statistics for current project.

Layout Toolbar

Styles toolbar allows you to easily customize workspace through a number of different IDE layouts.



Legacy

1024x768
Debug layout
Default
Legacy
MaxCode
Messages

Icon	Description
	Delete the selected layout.
	Save the current layout.
	Set the selected layout.

Help Toolbar

Help Toolbar provides access to information on using and registering compilers:

Icon	Description
?	Open Help file.
🔑	How To Register.

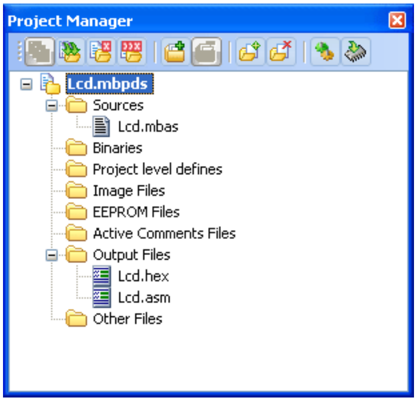
Related topics: Keyboard shortcuts, Integrated Tools

Customizing IDE Layout

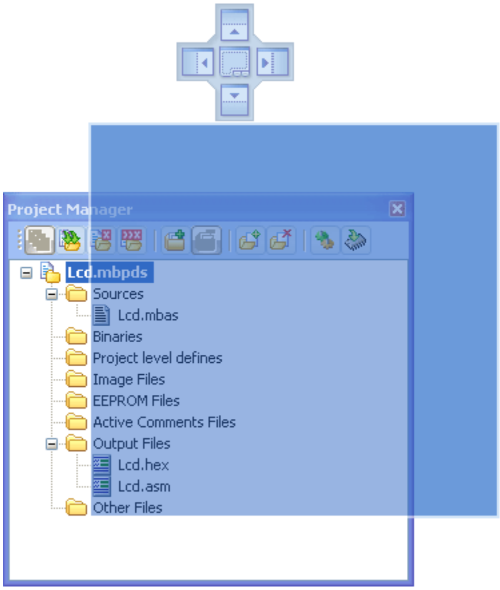
Docking Windows

You can increase the viewing and editing space for code, depending on how you arrange the windows in the IDE.

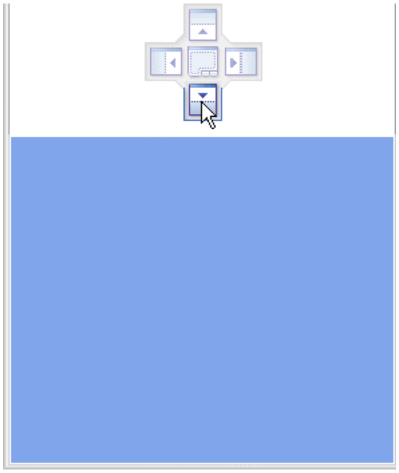
Step 1: Click the window you want to dock, to give it focus.



Step 2: Drag the tool window from its current location. A guide diamond appears. The four arrows of the diamond point towards the four edges of the IDE.



Step 3: Move the pointer over the corresponding portion of the guide diamond. An outline of the window appears in the designated area.



Step 4: To dock the window in the position indicated, release the mouse button.

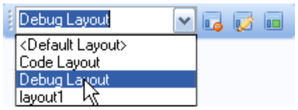
Tip : To move a dockable window without snapping it into place, press CTRL while dragging it.

Saving Layout

Once you have a window layout that you like, you can save the layout by typing the name for the layout and pressing the Save Layout Icon .

To set the layout select the desired layout from the layout drop-down list and click the Set Layout Icon .

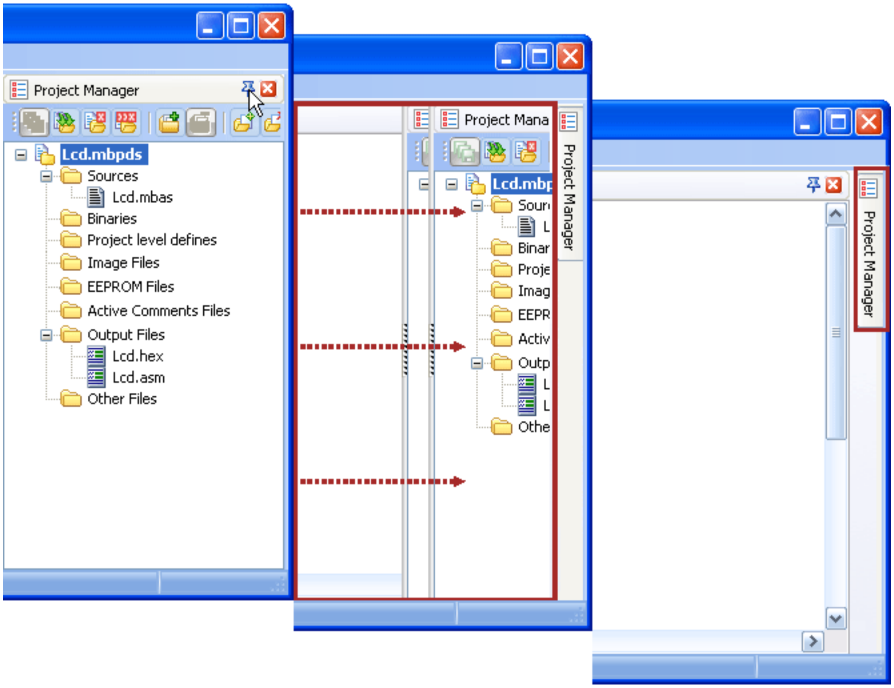
To remove the layout from the drop-down list, select the desired layout from the list and click the Delete Layout Icon .



Auto Hide

Auto Hide enables you to see more of your code at one time by minimizing tool windows along the edges of the IDE when not in use.

- Click the window you want to keep visible to give it focus.
- Click the Pushpin Icon on the title bar of the window.



When an auto-hidden window loses focus, it automatically slides back to its tab on the edge of the IDE. While a window is auto-hidden, its name and icon are visible on a tab at the edge of the IDE. To display an auto-hidden window, move your pointer over the tab. The window slides back into view and is ready for use.

Options

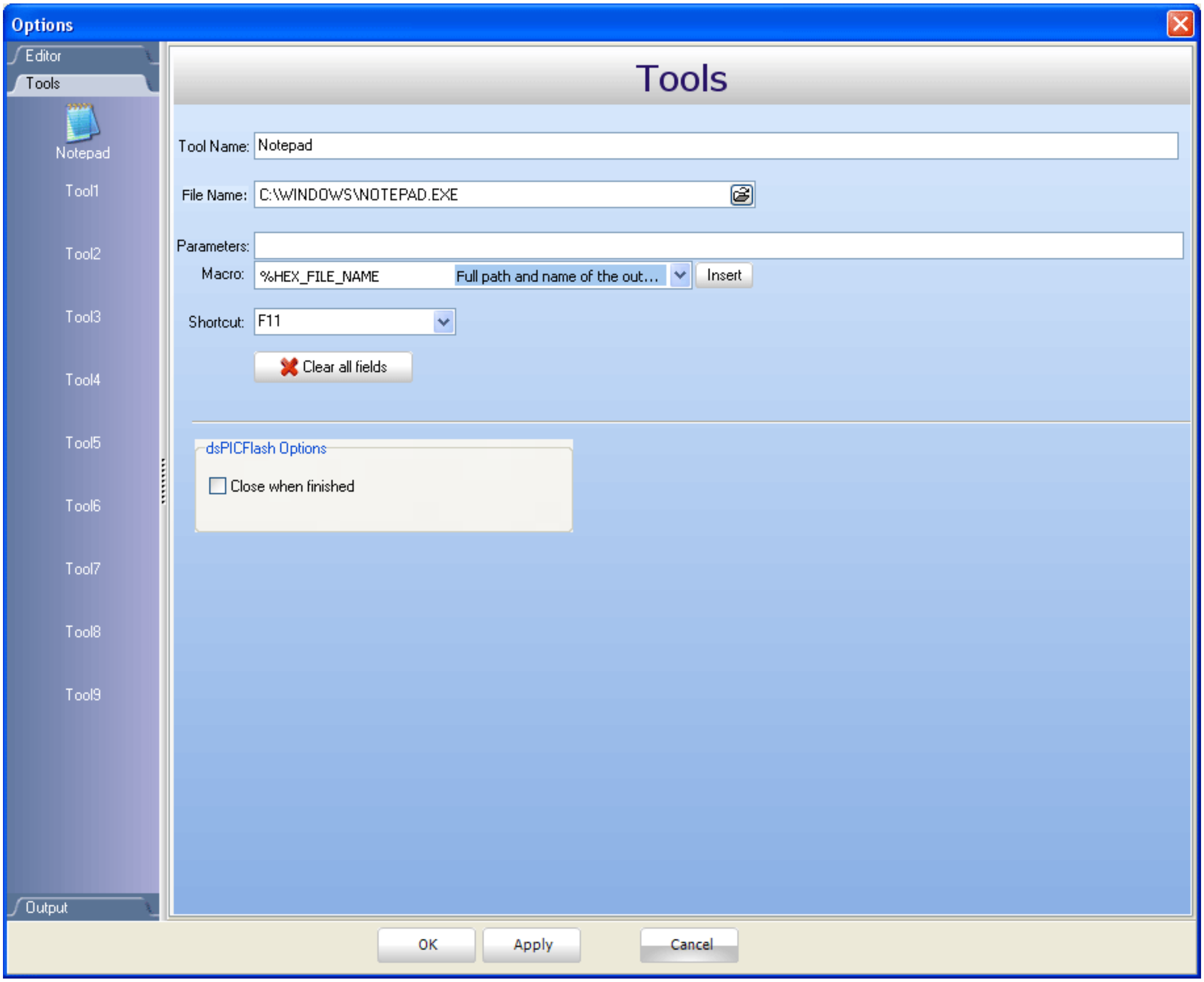
Options menu consists of three tabs: Code Editor, Tools and Output settings

Code editor

The Code Editor is advanced text editor fashioned to satisfy needs of professionals.

Tools

The mikroBasic PRO for dsPIC30/33 and PIC24 includes the Tools tab, which enables the use of shortcuts to external programs, like Calculator or Notepad.
You can set up to 10 different shortcuts, by editing Tool0 - Tool9.



Tool Name: Notepad

File Name: C:\WINDOWS\NOTEPAD.EXE

Parameters:

Macro: %HEX_FILE_NAME Full path and name of the out...

Shortcut: F11

dsPICFlash Options

☐ Close when finished

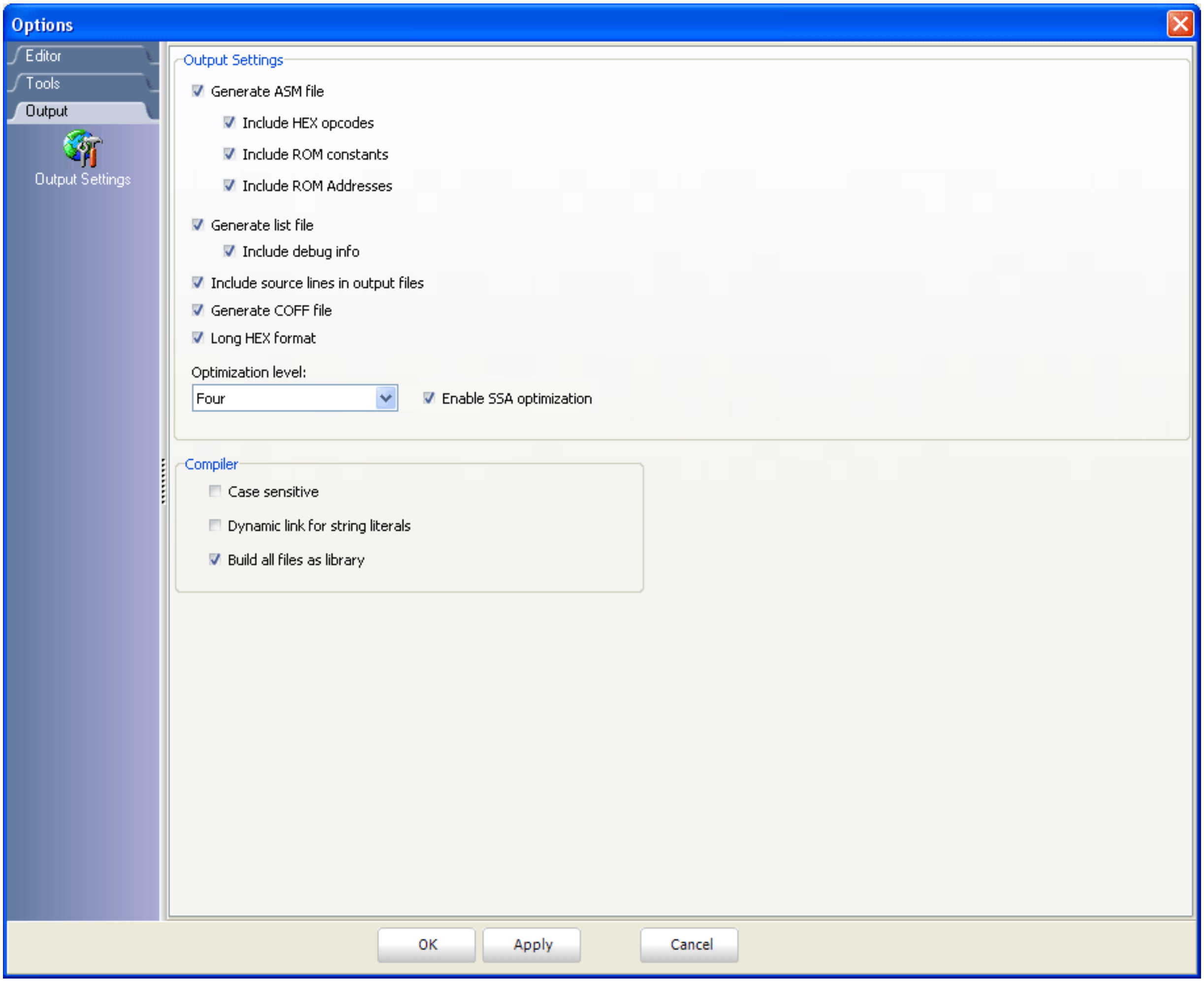
Output settings

By modifying Output Settings, user can configure the content of the output files.
You can enable or disable, for example, generation of ASM and List file.

Also, user can choose optimization level, and compiler specific settings, which include case sensitivity, dynamic link for string literals setting (described in mikroBasic PRO for dsPIC30/33 and PIC24 specifics).

Build all files as library enables user to use compiled library (`*.mcl`) on any MCU (when this box is checked), or for a selected MCU (when this box is left unchecked).

For more information on creating new libraries, see Creating New Library.



Output Settings

☑ Generate ASM file

☑ Include HEX opcodes

☑ Include ROM constants

☑ Include ROM Addresses

☑ Generate list file

☑ Include debug info

☑ Include source lines in output files

☑ Generate COFF file

☑ Long HEX format

Optimization level: Four

☑ Enable SSA optimization

Compiler

☐ Case sensitive

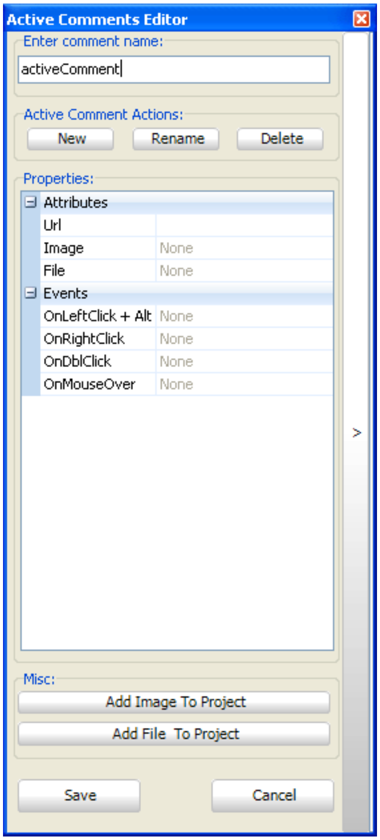
☐ Dynamic link for string literals

☑ Build all files as library

Integrated Tools

Active Comments Editor

Active Comments Editor is a tool, particularly useful when working with Lcd display. You can launch it from the drop-down menu **Tools › Active Comments Editor** or by clicking the Active Comment Editor Icon from Tools toolbar.

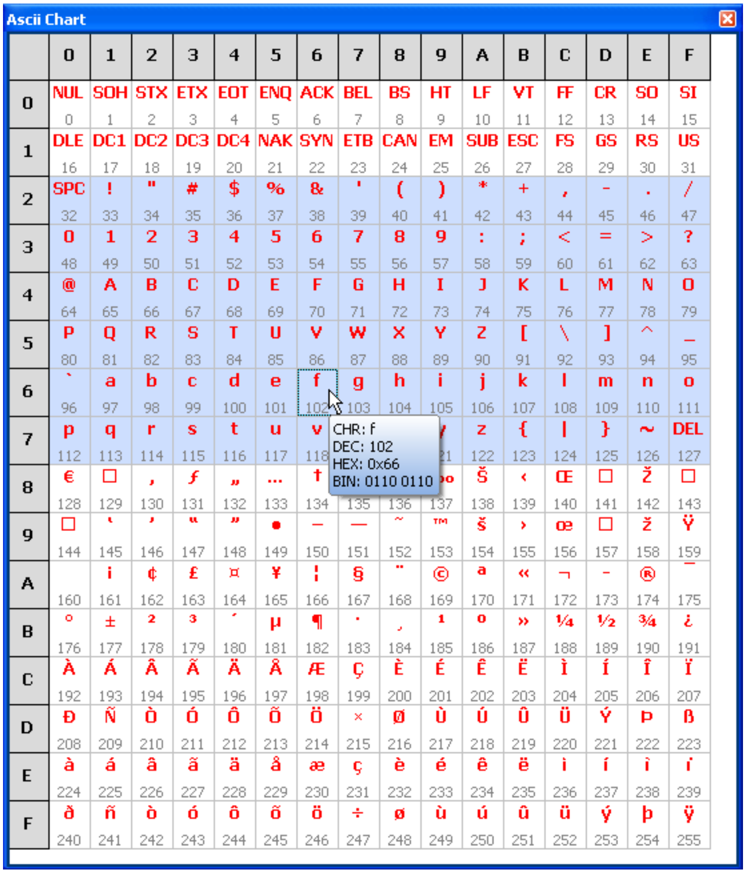


Enter comment name: activeComment

Attributes	
Url	
Image	None
File	None
Events	
OnLeftClick + Alt	None
OnRightClick	None
OnDblClick	None
OnMouseOver	None

ASCII Chart

The ASCII Chart is a handy tool, particularly useful when working with Lcd display. You can launch it from the drop-down menu **Tools › ASCII chart** or by clicking the View ASCII Chart Icon from Tools toolbar.



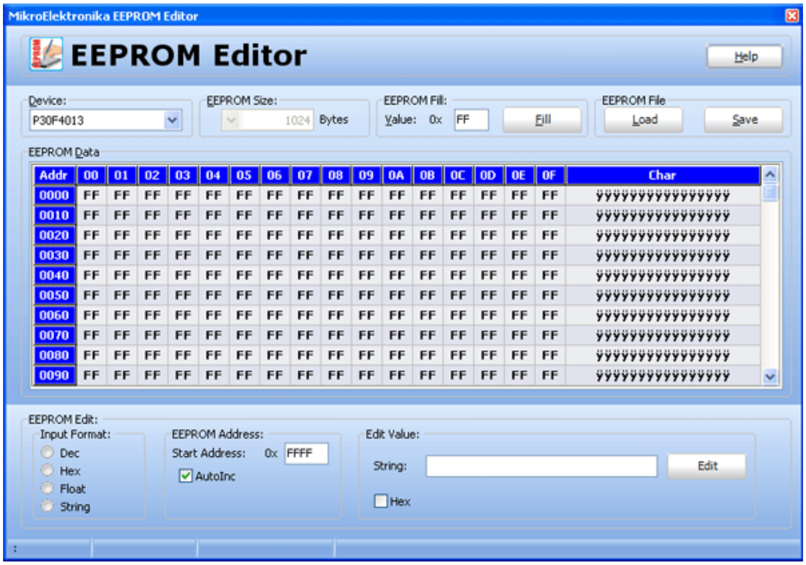
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0	SOH 1	STX 2	ETX 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
1	DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
2	SPC 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
6	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118			21	z 122	{ 123	| 124	} 125	~ 126	DEL 127
8	€ 128	□ 129	‚ 130	ƒ 131	„ 132	… 133	† 134	135	136	137	Š 138	‹ 139	Œ 140	□ 141	Ž 142	□ 143
9	□ 144	‘ 145	’ 146	“ 147	” 148	• 149	– 150	— 151	˜ 152	™ 153	š 154	› 155	œ 156	□ 157	ž 158	Ÿ 159
A	160	¡ 161	¢ 162	£ 163	¤ 164	¥ 165	¦ 166	§ 167	¨ 168	© 169	ª 170	« 171	¬ 172	- 173	® 174	¯ 175
B	° 176	± 177	² 178	³ 179	´ 180	µ 181	¶ 182	· 183	¸ 184	¹ 185	º 186	» 187	¼ 188	½ 189	¾ 190	¿ 191
C	À 192	Á 193	Â 194	Ã 195	Ä 196	Å 197	Æ 198	Ç 199	È 200	É 201	Ê 202	Ë 203	Ì 204	Í 205	Î 206	Ï 207
D	Ð 208	Ñ 209	Ò 210	Ó 211	Ô 212	Õ 213	Ö 214	× 215	Ø 216	Ù 217	Ú 218	Û 219	Ü 220	Ý 221	Þ 222	ß 223
E	à 224	á 225	â 226	ã 227	ä 228	å 229	æ 230	ç 231	è 232	é 233	ê 234	ë 235	ì 236	í 237	î 238	ï 239
F	ð 240	ñ 241	ò 242	ó 243	ô 244	õ 245	ö 246	÷ 247	ø 248	ù 249	ú 250	û 251	ü 252	ý 253	þ 254	ÿ 255

CHR: f
DEC: 102
HEX: 0x66
BIN: 0110 0110

EEPROM Editor

The EEPROM Editor is used for manipulating MCU's EEPROM memory. You can launch it from the drop-down menu **Tools › EEPROM Editor**.

When you run mikroElektronika programmer software from mikroBasic PRO for dsPIC30/33 and PIC24 IDE - `project_name.hex` file will be loaded automatically while `ihex` file must be loaded manually.



Device: P30F4013

EEPROM Size: 1024 Bytes

Value: 0x FF

Addr	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Char
0000	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0010	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0040	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0050	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0060	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0080	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0090	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Start Address: 0x FFFF

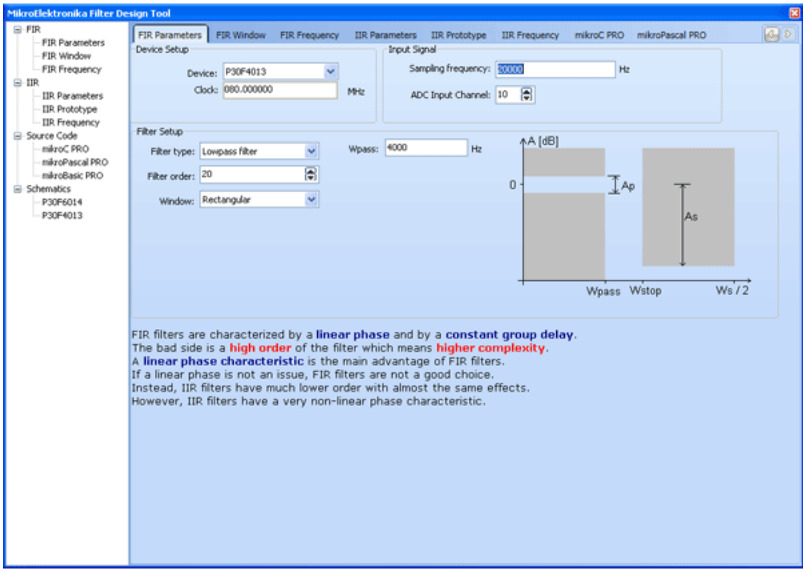
☑ AutoInc

String:

☐ Hex

Filter Designer

The Filter designer is a tool for designing FIR and IIR filters. It has an user-friendly visual interface for setting the filter parameters. Filter designer output is the mikroBasic PRO for dsPIC30/33 and PIC24 compatible code. You can launch it from the drop-down menu **Tools › Filter Designer**.

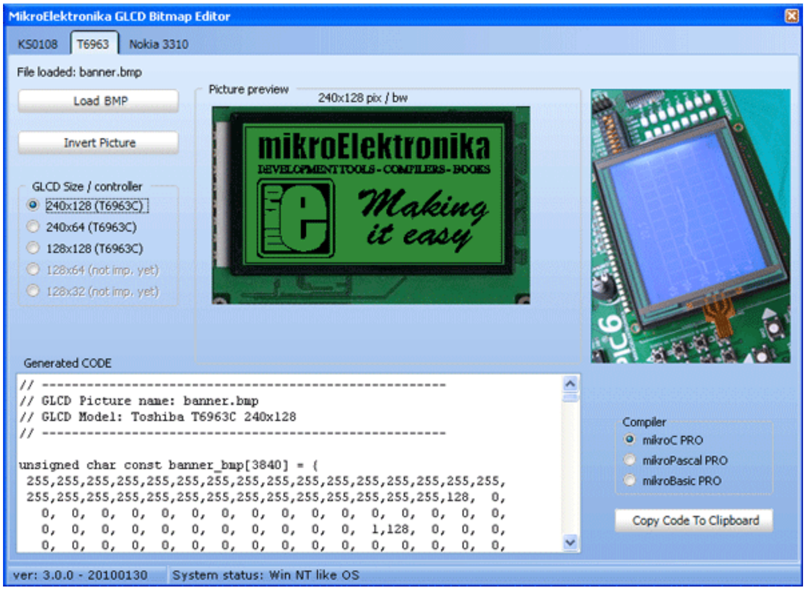


Device: P30F4013
Clock: 080.000000 MHz
Sampling frequency: 20000 Hz
ADC Input Channel: 10
Filter type: Lowpass filter
Filter order: 20
Window: Rectangular
Wpass: 4000 Hz

FIR filters are characterized by a **linear phase** and by a **constant group delay**.
The bad side is a **high order** of the filter which means **higher complexity**.
A **linear phase characteristic** is the main advantage of FIR filters.
If a linear phase is not an issue, FIR filters are not a good choice.
Instead, IIR filters have much lower order with almost the same effects.
However, IIR filters have a very non-linear phase characteristic.

Graphic Lcd Bitmap Editor

The mikroBasic PRO for dsPIC30/33 and PIC24 includes the Graphic Lcd Bitmap Editor. Output is the mikroBasic PRO for dsPIC30/33 and PIC24 compatible code. You can launch it from the drop-down menu **Tools › Glcd Bitmap Editor**.



File loaded: banner.bmp

GLCD Size / controller:
● 240x128 (T6963C)
○ 240x64 (T6963C)
○ 128x128 (T6963C)
○ 128x64 (not imp. yet)
○ 128x32 (not imp. yet)

Compiler:
● mikroC PRO
○ mikroPascal PRO
○ mikroBasic PRO

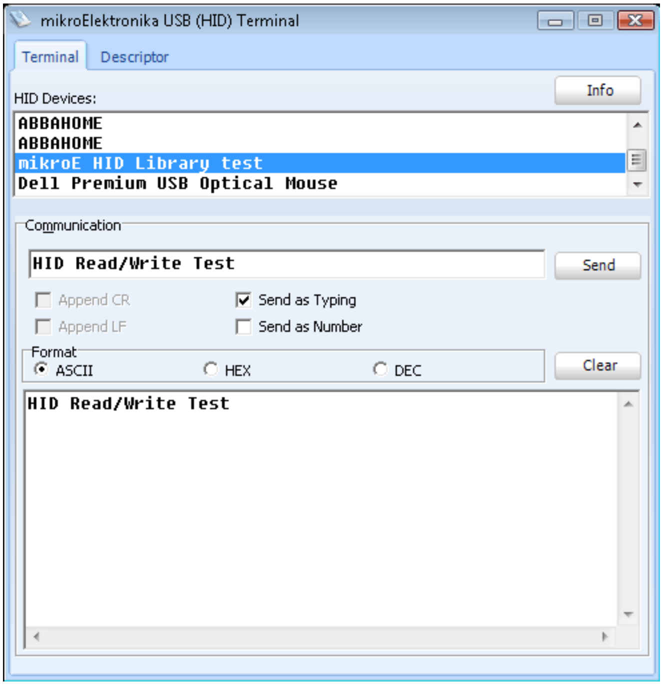
Generated CODE

```
// --------------------------------------------------------
// GLCD Picture name: banner.bmp
// GLCD Model: Toshiba T6963C 240x128
// --------------------------------------------------------

unsigned char const banner_bmp[3840] = {
 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
 255,255,255,255,255,255,255,255,255,255,255,255,255,255,128,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,128,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
```

ver: 3.0.0 - 20100130 System status: Win NT like OS

HID Terminal

The mikroBasic PRO for dsPIC30/33 and PIC24 includes the HID communication terminal for USB communication. You can launch it from the drop-down menu **Tools › HID Terminal**.



HID Devices:

```
ABBAHOME
ABBAHOME
mikroE HID Library test
Dell Premium USB Optical Mouse
```

Communication

```
HID Read/Write Test
```

☐ Append CR ☑ Send as Typing

☐ Append LF ☐ Send as Number

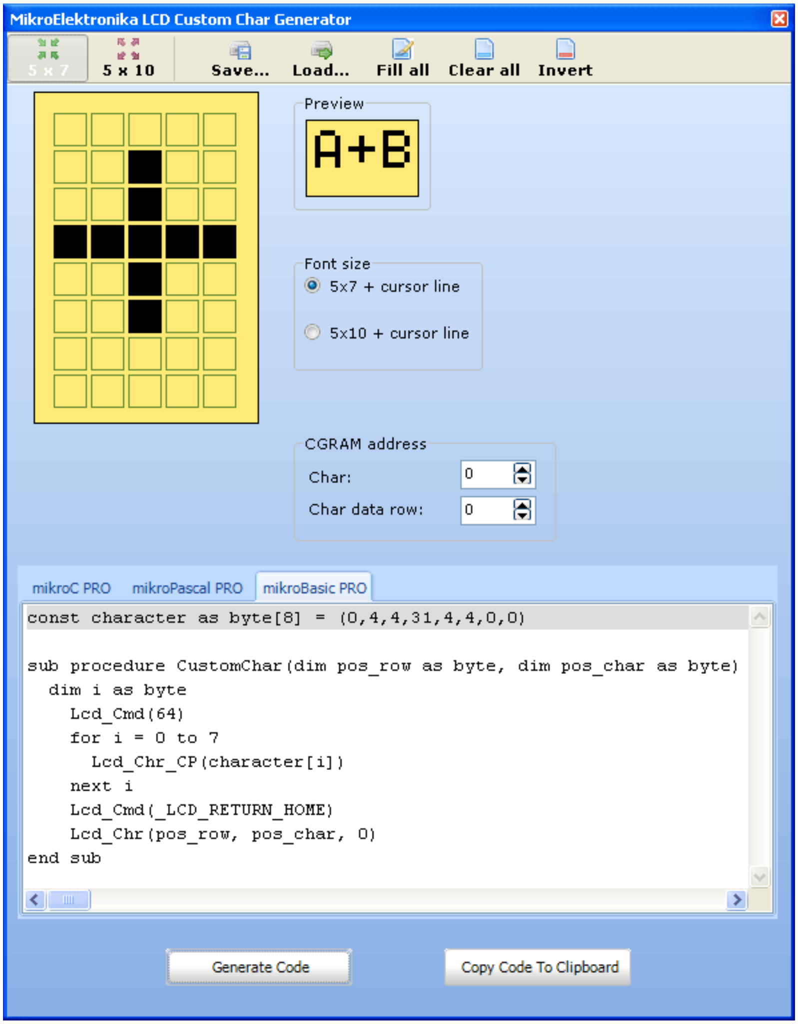
Format

● ASCII ○ HEX ○ DEC

```
HID Read/Write Test
```

Lcd Custom Character

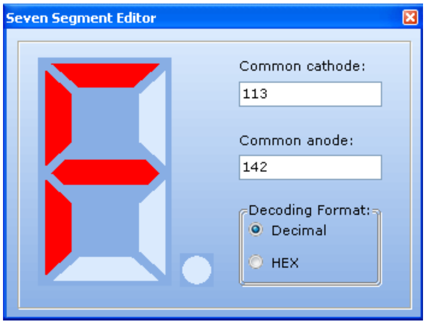
mikroBasic PRO for dsPIC30/33 and PIC24 includes the Lcd Custom Character. Output is mikroBasic PRO for dsPIC30/33 and PIC24 compatible code. You can launch it from the drop-down menu **Tools › Lcd Custom Character**.



```
const character as byte[8] = (0,4,4,31,4,4,0,0)

sub procedure CustomChar(dim pos_row as byte, dim pos_char as byte)
  dim i as byte
    Lcd_Cmd(64)
    for i = 0 to 7
      Lcd_Chr_CP(character[i])
    next i
    Lcd_Cmd(_LCD_RETURN_HOME)
    Lcd_Chr(pos_row, pos_char, 0)
end sub
```

Seven Segment Editor

The Seven Segment Editor is a convenient visual panel which returns decimal/hex value for any viable combination you would like to display on seven segment display. Click on the parts of seven segment image to get the requested value in the edit boxes. You can launch it from the drop-down menu **Tools › Seven Segment Editor** or by clicking the Seven Segment Editor Icon from Tools toolbar.



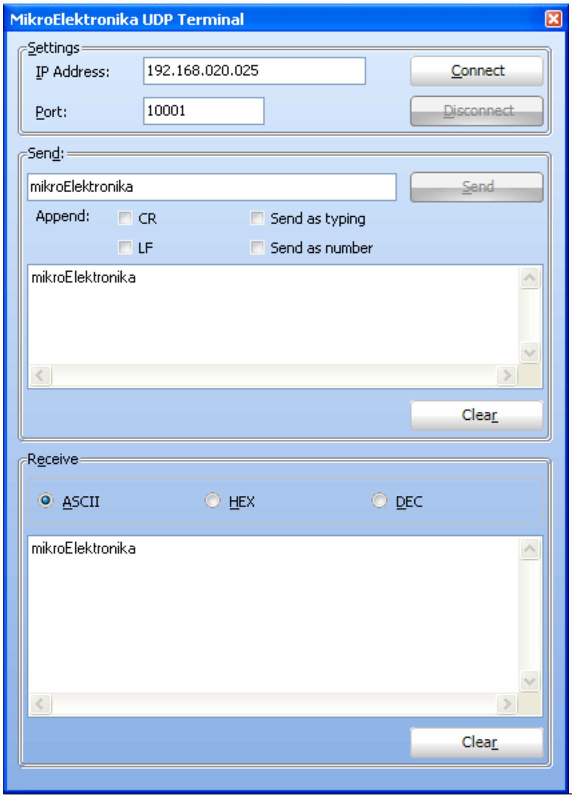
Common cathode: 113

Common anode: 142

Decoding Format:
- ● Decimal
- ○ HEX

UDP Terminal

The mikroBasic PRO for dsPIC30/33 and PIC24 includes the UDP Terminal. You can launch it from the drop-down menu **Tools › UDP Terminal**.



IP Address: 192.168.020.025

Port: 10001

Send: mikroElektronika

Append: ☐ CR ☐ LF ☐ Send as typing ☐ Send as number

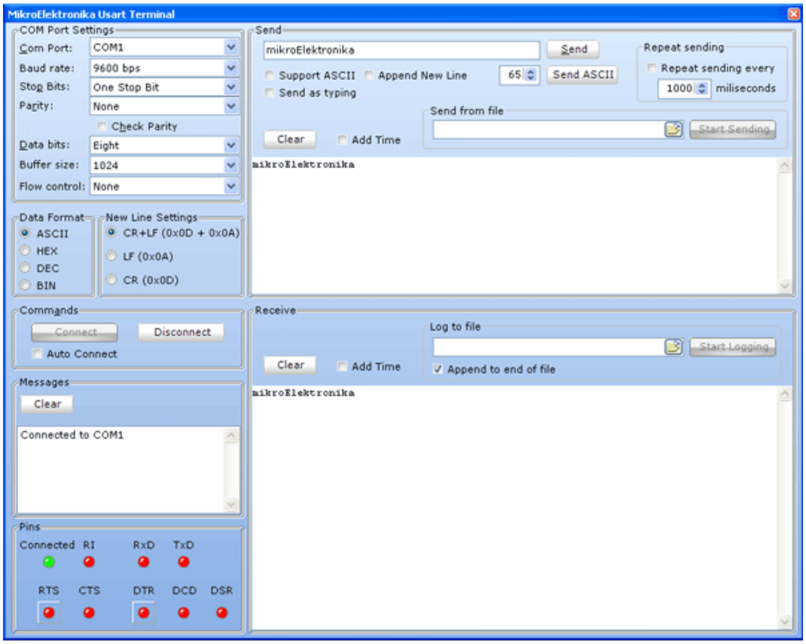
mikroElektronika

Receive: ● ASCII ○ HEX ○ DEC

mikroElektronika

USART Terminal

The mikroBasic PRO for dsPIC30/33 and PIC24 includes the USART communication terminal for RS232 communication. You can launch it from the drop-down menu **Tools › USART Terminal** or by clicking the USART Terminal Icon from Tools toolbar.



MikroElektronika Usart Terminal

COM Port Settings
Com Port: COM1
Baud rate: 9600 bps
Stop Bits: One Stop Bit
Parity: None
☐ Check Parity
Data bits: Eight
Buffer size: 1024
Flow control: None

Data Format
● ASCII
○ HEX
○ DEC
○ BIN

New Line Settings
● CR+LF (0x0D + 0x0A)
○ LF (0x0A)
○ CR (0x0D)

Commands
☐ Auto Connect

Messages
Connected to COM1

Send
mikroElektronika
☐ Support ASCII ☐ Append New Line 65
☐ Send as typing
Repeat sending
☐ Repeat sending every 1000 miliseconds
☐ Add Time

mikroElektronika

Receive
Log to file
☐ Add Time
☑ Append to end of file

mikroElektronika

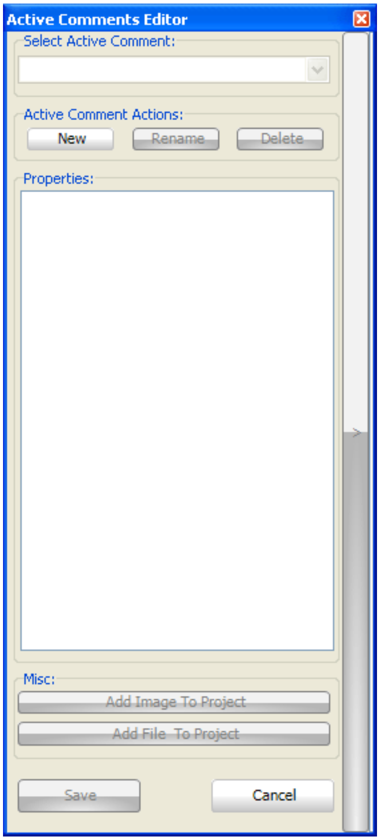
Active Comments

The idea of Active Comments is to make comments *alive* and give old fashioned comments new meaning and look. From now on, you can assign mouse event on your comments and 'tell' your comments what to do on each one. For example, on left mouse click, open some web address in your browser, on mouse over show some picture and on mouse double click open some file.

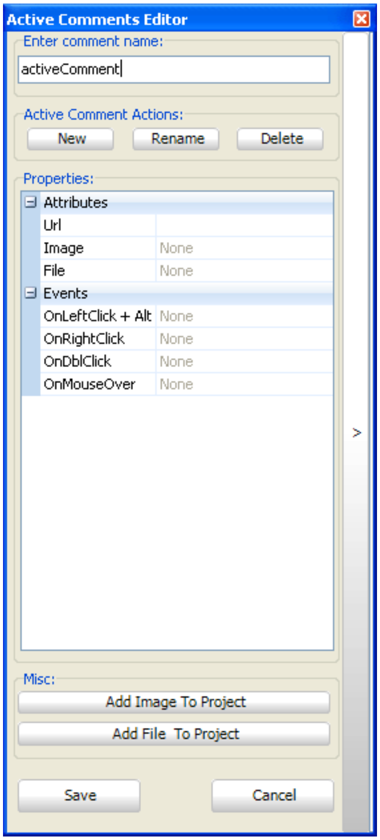
Suppose we are writing a example for a GSM/GPSR module which is connected to the EasyPIC6 and we would like to provide a photo of our hardware (jumpers, cables, etc.). within the example.
It would also be nice to put some documentation about chip we are using and a GSM module extra board. Now we can have all those things defined in one single comment using **Active Comment Editor**.

New Active Comment

When you start Active Comment Editor for the first time (from the View menu, from editor's pop-up menu, or by pressing Ctrl + Alt + P) you will get an empty editor:



By clicking the New button you are prompted to enter a name for the comment:



activeComment

Attributes

Url	
Image	None
File	None

Events

OnLeftClick + Alt	None
OnRightClick	None
OnDblClick	None
OnMouseOver	None

You can notice that when you start typing a name, properties pane is automatically displayed so you can edit properties if you wish. A Comment will be is created when you click Save button.

Properties are consisted of two major categories - Attributes and Events.

Attributes can be:

- URL - Valid web address.
- Image - Image has to be previously added to Project (Project Manager > Images).
- File - File has to be previously added to Project (Project Manager > Other Files).

There are four predefined event types you can apply to an Active Comment:

1. OnLeftClick + Alt
2. OnRightClick
3. OnDoubleClick
4. OnMouseOver

First three event types can have one of the following three actions:

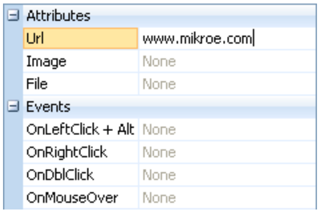
1. OpenUrl - Opens entered URL in default Web browser.
2. OpenFile - Opens a file within a default program associated with the file extension (defined by Windows).
3. None - Does nothing.

The fourth event, OnMouseOver, has only 2 actions:

1. PreviewImage - Shows image when cursor is moved over a comment.
2. None - Does nothing.

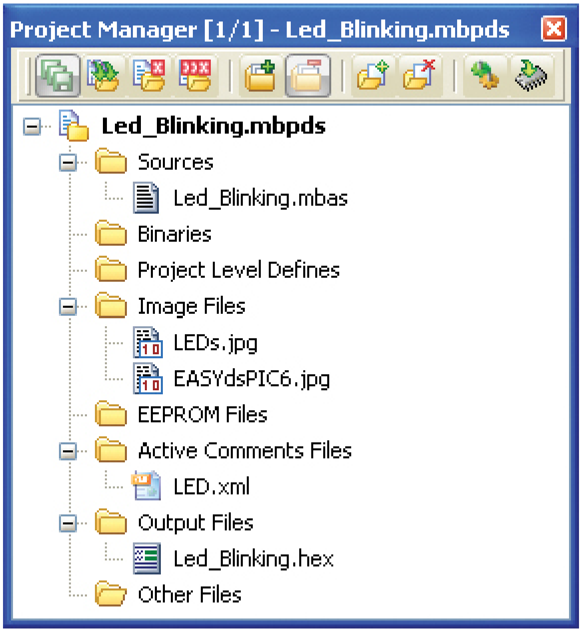
Attributes are tightly bounded with events. For example, you can not have OnLeftClick + Alt -> OpenFile if there is no file attribute set, or if there is no file added to project. The same behavior applies to image attribute.

Let's start editing our Active Comment by entering some valid web address in the URL field:

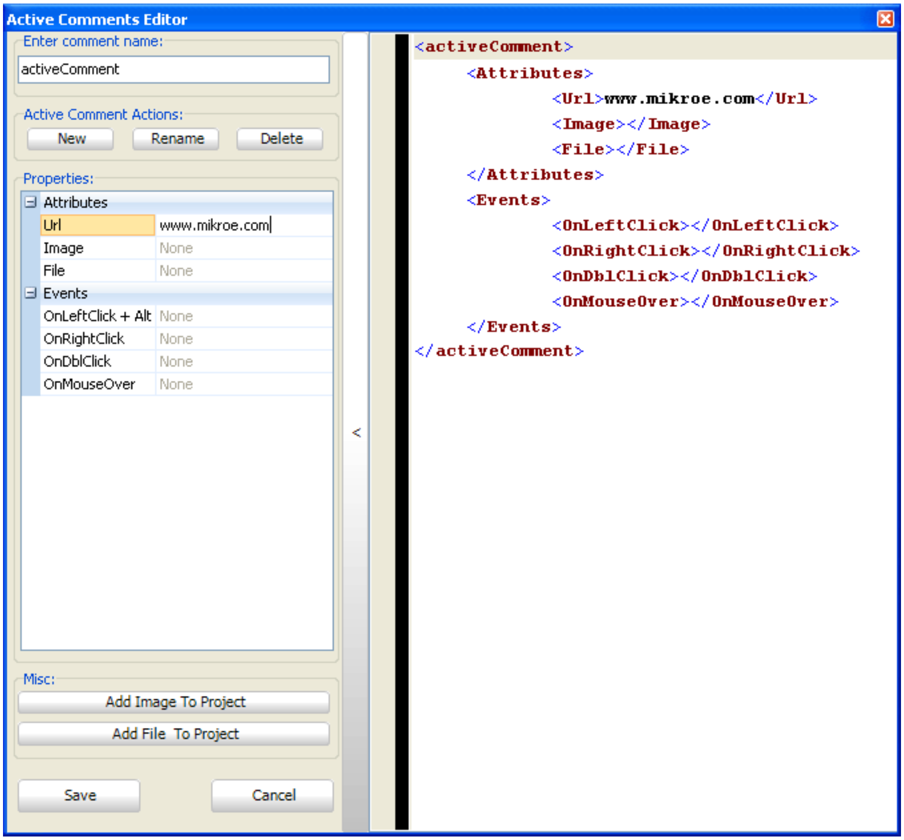


Attributes	
Url	www.mikroe.com
Image	None
File	None
Events	
OnLeftClick + Alt	None
OnRightClick	None
OnDblClick	None
OnMouseOver	None

For every Active Comment a XML file will be created, containing all valid information regarding the Active Comment - attributes, events, etc. and it is automatically added to Project manager after saving it:



You can see the contents of the created XML file by expanding Active Comment Editor:



Enter comment name: activeComment

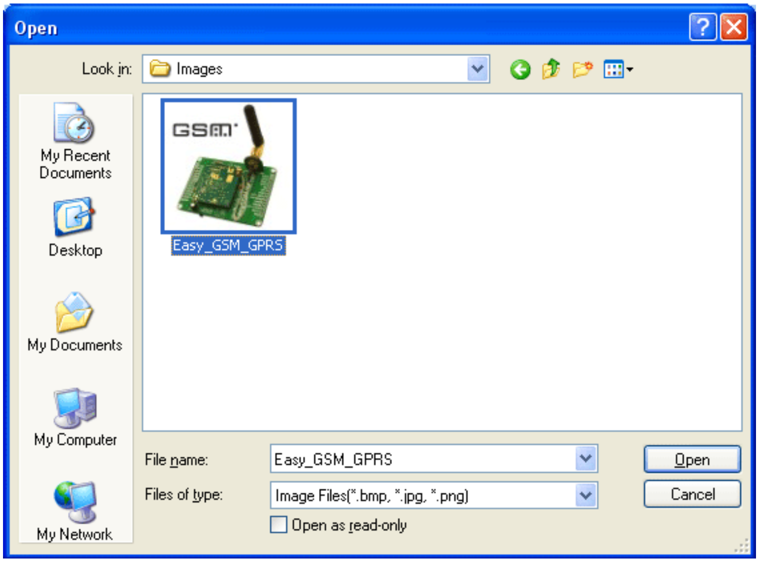
Properties:

Attributes	
Url	www.mikroe.com
Image	None
File	None
Events	
OnLeftClick + Alt	None
OnRightClick	None
OnDblClick	None
OnMouseOver	None

```
<activeComment>
     <Attributes>
          <Url>www.mikroe.com</Url>
          <Image></Image>
          <File></File>
     </Attributes>
     <Events>
          <OnLeftClick></OnLeftClick>
          <OnRightClick></OnRightClick>
          <OnDblClick></OnDblClick>
          <OnMouseOver></OnMouseOver>
     </Events>
</activeComment>
```

As we mentioned above you can add image or file which are already included in project. If the the desired image or file aren't added, you can do it directly from here by clicking the Add Image To Project or Add File To Project button.

Next file dialog will be opened:

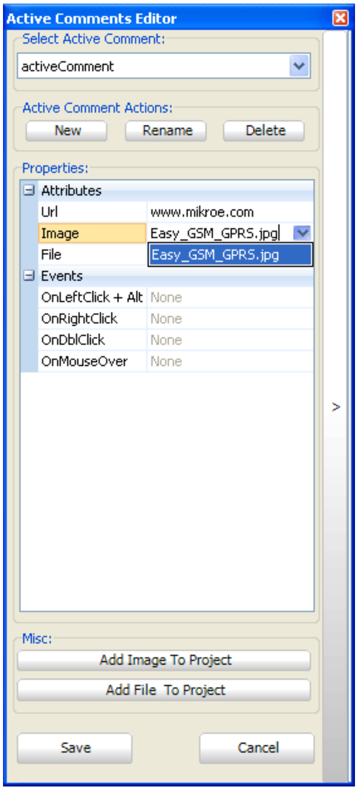


File name: Easy_GSM_GPRS

Files of type: Image Files(*.bmp, *.jpg, *.png)

There, you should select the desired image to be added. In our example, `Easy_GSM_GPRS.jpg` image will be added.

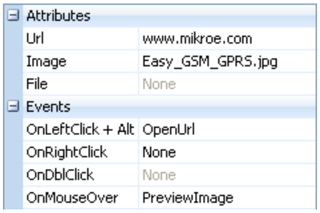
Selected picture is automatically added to the drop down list of the Image field in Active Comment Editor:



Select Active Comment: activeComment

Url	www.mikroe.com
Image	Easy_GSM_GPRS.jpg
File	Easy_GSM_GPRS.jpg
OnLeftClick + Alt	None
OnRightClick	None
OnDblClick	None
OnMouseOver	None

Now, when image has been selected, we can assign an event to it. For example, OnMouseOver will be used for PreviewImage action, and OnLeftClick + Alt will be assigned to OpenUrl action:

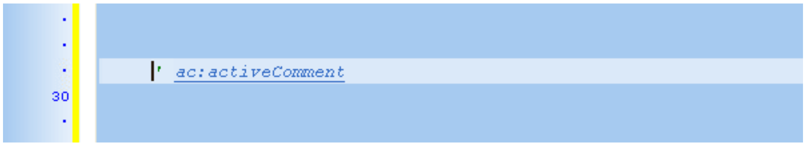


Attributes	
Url	www.mikroe.com
Image	Easy_GSM_GPRS.jpg
File	None
Events	
OnLeftClick + Alt	OpenUrl
OnRightClick	None
OnDblClick	None
OnMouseOver	PreviewImage

Now we can save our changes to Active Comment by clicking the Save button.

Note: Setting file attributes is same as for image, so it won't be explained separately.

Once we have finished creating our active comment, we can notice that it has been added to source file on current caret position with `ac`: prefix 'telling' IDE that it is active comment:

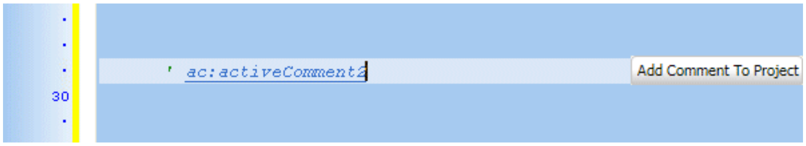


```
' ac:activeComment
```

Now let's try it. If you LeftClick+Alt on it, URL in default Web browser will be opened. If you hover the mouse over it, you will see an Image preview:

```
' ac:activeComment
```

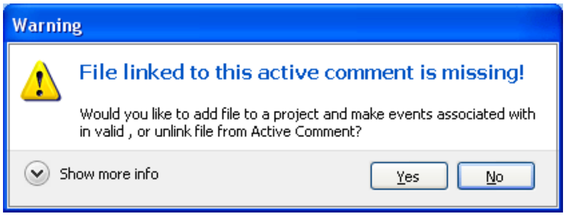
There is another way to add an active comment to an active project. You can do it simply by typing a comment in old fashion way, except with `ac`: prefix. So it would look like this:



```
' ac:activeComment2
```

Notice that when you stop typing, Add Comment To Project button will show. By clicking on it, you will open Active Comment Editor and comment name will be already set, so you need only to adjust attributes and settings.
After saving you can always edit your active comment by Active Comment Editor, and switch between comments directly from editor.

If you remove a file from the Project Manager or add an Active Comment File which contains information about the file which is no longer in project, and hover the mouse over the comment, you will be prompted to either add file to project or remove event definition from Active Comment for this file:

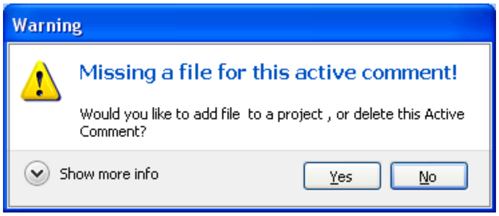


Warning

File linked to this active comment is missing!

Would you like to add file to a project and make events associated with in valid , or unlink file from Active Comment?

If you remove active comment file from the Project Manager, you'll receive this message:

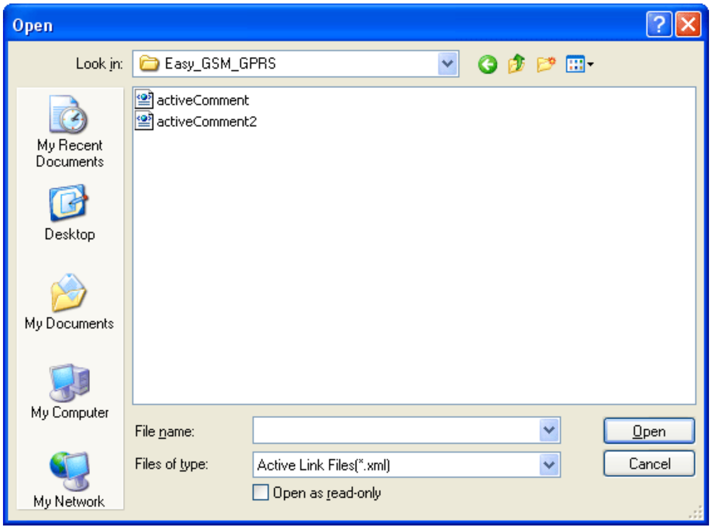


Warning

Missing a file for this active comment!

Would you like to add file to a project , or delete this Active Comment?

Click on Yes button you'll prompted for an active comment file:



Look in: Easy_GSM_GPRS

activeComment
activeComment2

File name:

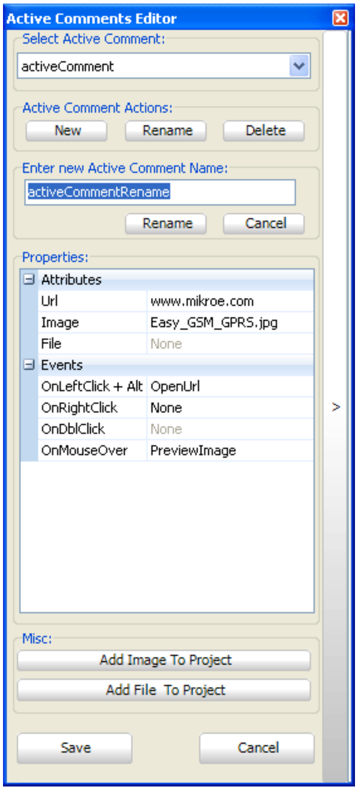
Files of type: Active Link Files(*.xml)

☐ Open as read-only

If you click No, comment will be removed from the source code.

Renaming Active Comment

When you click on rename button, you will be prompted to enter new name:



Select Active Comment: activeComment

Enter new Active Comment Name: activeCommentRename

Properties:

⊟ Attributes	
Url	www.mikroe.com
Image	Easy_GSM_GPRS.jpg
File	None
⊟ Events	
OnLeftClick + Alt	OpenUrl
OnRightClick	None
OnDblClick	None
OnMouseOver	PreviewImage

Now click again Rename button. Now you have renamed your Active Comment in such a way that its filename, source code name are changed:

```
' ac:activeCommentRename
```

Deleting Active Comment

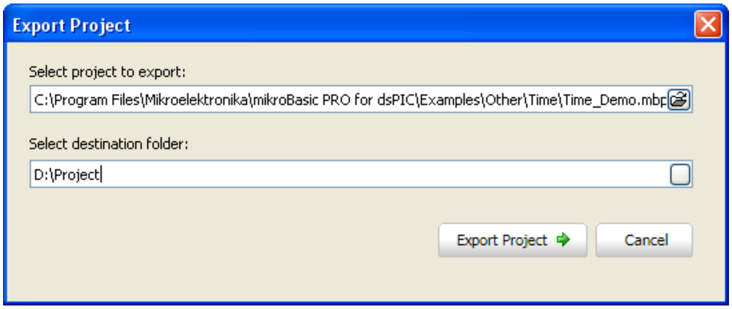
Deleting active comment works similar like renaming it. By clicking on delete button, you will remove an active comment from both code and Project Manager.

Export Project

This option is very convenient and finds its use in relocating your projects from one place to another (e.g. from your work computer to your home computer).

Often, project contains complicated search paths (files involved within your project could be in a different folders, even on different hard disks), so it is very likely that some files will be forgotten during manual relocation.
In order to simplify this, Export Project gives you opportunity to do this task automatically.

To open Export Project, from Project menu select Export Project or hit Ctrl + Alt + E. Following window will appear:



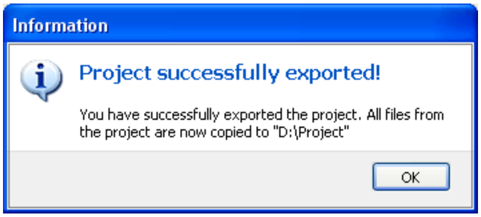
Select project to export: C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Examples\Other\Time\Time_Demo.mbp

Select destination folder: D:\Project

In the empty input boxes, current location and the destination folder of the desired project should be entered.

By default, currently active project will be set for export. You can change it any time by clicking the Open Button .

Once you have entered the appropriate data, click Export Project button. After exporting is done, and if everything was OK, you'll receive a message:



Project successfully exported!

You have successfully exported the project. All files from the project are now copied to "D:\Project"

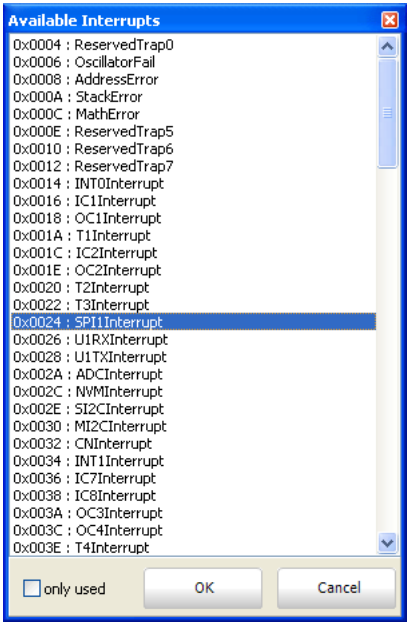
Now, Export Project has copied all project files into desired folder and changed project search paths, so you can easily move the entire folder to another location and run the project.

Jump To Interrupt

Lets you choose which interrupt you want to jump to.

Requirement: Interrupt routine is included in project.

You can call Jump To Interrupt by selecting **Run › Jump To Interrupt** from the drop-down menu, or by clicking the Jump To Interrupt Icon , from the Watch Values Window.



```
0x0004 : ReservedTrap0
0x0006 : OscillatorFail
0x0008 : AddressError
0x000A : StackError
0x000C : MathError
0x000E : ReservedTrap5
0x0010 : ReservedTrap6
0x0012 : ReservedTrap7
0x0014 : INT0Interrupt
0x0016 : IC1Interrupt
0x0018 : OC1Interrupt
0x001A : T1Interrupt
0x001C : IC2Interrupt
0x001E : OC2Interrupt
0x0020 : T2Interrupt
0x0022 : T3Interrupt
0x0024 : SPI1Interrupt
0x0026 : U1RXInterrupt
0x0028 : U1TXInterrupt
0x002A : ADCInterrupt
0x002C : NVMInterrupt
0x002E : SI2CInterrupt
0x0030 : MI2CInterrupt
0x0032 : CNInterrupt
0x0034 : INT1Interrupt
0x0036 : IC7Interrupt
0x0038 : IC8Interrupt
0x003A : OC3Interrupt
0x003C : OC4Interrupt
0x003E : T4Interrupt
```

By checking the Only Used box, you can display only the used breakpoints.

Regular Expressions

Introduction

Regular Expressions are a widely-used method of specifying patterns of text to search for. Special metacharacters allow you to specify, for instance, that a particular string you are looking for, occurs at the beginning, or end of a line, or contains `n` recurrences of a certain character.

Simple matches

Any single character matches itself, unless it is a metacharacter with a special meaning described below. A series of characters matches that series of characters in the target string, so the pattern `“short”` would match `“short”` in the target string. You can cause characters that normally function as metacharacters or escape sequences to be interpreted by preceding them with a backslash `“\”`.
For instance, metacharacter `“^”` matches beginning of string, but `“\^”` matches character `“^”`, and `“\\”` matches `“\”`, etc.

Examples:

`unsigned` matches string `'unsigned'`
`\^unsigned` matches string `'^unsigned'`

Escape sequences

Characters may be specified using a escape sequences: `“\n”` matches a newline, `“\t”` a tab, etc. More generally, `\xnn`, where `nn` is a string of hexadecimal digits, matches the character whose ASCII value is `nn`.
If you need wide (Unicode) character code, you can use `'\x{nnnn}'`, where `‘nnnn’` - one or more hexadecimal digits.

`\xnn` - char with hex code `nn`
`\x{nnnn)` - char with hex code `nnnn` (one byte for plain text and two bytes for Unicode)
`\t` - tab (HT/TAB), same as `\x09`
`\n` - newline (NL), same as `\x0a`
`\r` - car.return (CR), same as `\x0d`
`\f` - form feed (FF), same as `\x0c`
`\a` - alarm (bell) (BEL), same as `\x07`
`\e` - escape (ESC) , same as `\x1b`

Examples:

`unsigned\x20int` matches `'unsigned int'` (note space in the middle)
`\tunsigned` matches `'unsigned'` (predecessed by tab)

Character classes

You can specify a character class, by enclosing a list of characters in `[]`, which will match any of the characters from the list. If the first character after the `“[“` is `“^”`, the class matches any character not in the list.

Examples:

`count[aeiou]r` finds strings `'countar'`, `'counter'`, etc. but not `'countbr'`, `'countcr'`, etc.
`count[^aeiou]r` finds strings `'countbr'`, `'countcr'`, etc. but not `'countar'`, `'counter'`, etc.

Within a list, the `"-"` character is used to specify a range, so that `a-z` represents all characters between `"a"` and `"z"`, inclusive.

If you want `"-"` itself to be a member of a class, put it at the start or end of the list, or precede it with a backslash.
If you want `']'`, you may place it at the start of list or precede it with a backslash.

Examples:

`[-az]` matches `'a'`, `'z'` and `'-'`
`[az-]` matches `'a'`, `'z'` and `'-'`
`[a\-z]` matches `'a'`, `'z'` and `'-'`
`[a-z]` matches all twenty six small characters from `'a'` to `'z'`
`[\n-\x0D]` matches any of `#10,#11,#12,#13`.
`[\d-t]` matches any digit, `'-'` or `'t'`.
`[]-a]` matches any char from `']'`..`'a'`.

Metacharacters

Metacharacters are special characters which are the essence of regular expressions. There are different types of metacharacters, described below.

Metacharacters - Line separators

`^` - start of line
`$` - end of line
`\A` - start of text
`\Z` - end of text
`.` - any character in line

Examples:

`^PORTA` - matches string ‘ `PORTA` ‘ only if it’s at the beginning of line
`PORTA$` - matches string ‘ `PORTA` ‘ only if it’s at the end of line
`^PORTA$` - matches string ‘ `PORTA` ‘ only if it’s the only string in line
`PORT.r` - matches strings like ‘`PORTA`’, ‘`PORTB`’, ‘`PORT1`’ and so on

The `“^”` metacharacter by default is only guaranteed to match beginning of the input string/text, and the `“$”` metacharacter only at the end. Embedded line separators will not be matched by `^”` or `“$”`.
You may, however, wish to treat a string as a multi-line buffer, such that the `“^”` will match after any line separator within the string, and `“$”` will match before any line separator.
Regular expressions works with line separators as recommended at http://www.unicode.org/unicode/reports/tr18/

Metacharacters - Predefined classes

`\w` - an alphanumeric character (including `"_"`)
`\W` - a nonalphanumeric character
`\d` - a numeric character
`\D` - a non-numeric character
`\s` - any space (same as [`\t\n\r\f`])
`\S` - a non space

You may use `\w`, `\d` and `\s` within custom character classes.

Example:

`routi\de` - matches strings like `'routi1e'`, `'routi6e'` and so on, but not `'routine'`, `'routime'` and so on.

Metacharacters - Word boundaries

A word boundary (`"\b"`) is a spot between two characters that has an alphanumeric character (`"\w"`) on one side, and a nonalphanumeric character (`"\W"`) on the other side (in either order), counting the imaginary characters off the beginning and end of the string as matching a `"\W"`.

`\b` - match a word boundary)
`\B` - match a non-(word boundary)

Metacharacters - Iterators

Any item of a regular expression may be followed by another type of metacharacters - iterators. Using this metacharacters,you can specify number of occurences of previous character, metacharacter or subexpression.

`*` - zero or more (“greedy”), similar to {0,}
`+` - one or more (“greedy”), similar to {1,}
`?` - zero or one (“greedy”), similar to {0,1}
`{n}` - exactly n times (“greedy”)
`{n,}` - at least n times (“greedy”)
`{n,m}` - at least n but not more than m times (“greedy”)
`*?` - zero or more (“non-greedy”), similar to {0,}?
`+?` - one or more (“non-greedy”), similar to {1,}?
`??` - zero or one (“non-greedy”), similar to {0,1}?
`{n}?` - exactly n times (“non-greedy”)
`{n,}?` - at least n times (“non-greedy”)
`{n,m}?` - at least n but not more than m times (“non-greedy”)

So, digits in curly brackets of the form, `{n,m}`, specify the minimum number of times to match the item `n` and the maximum `m`. The form `{n}` is equivalent to `{n,n}` and matches exactly `n` times. The form `{n,}` matches `n` or more times. There is no limit to the size of `n` or `m`, but large numbers will chew up more memory and slow down execution.

If a curly bracket occurs in any other context, it is treated as a regular character.

Examples:

`count.*r ß`- matches strings like `'counter'`, `'countelkjdflkj9r'` and `'countr'`
`count.+r` - matches strings like `'counter'`, `'countelkjdflkj9r'` but not `'countr'`
`count.?r` - matches strings like `'counter'`, `'countar'` and `'countr'` but not `'countelkj9r'`
`counte{2}r` - matches string `'counteer'`
`counte{2,}r` - matches strings like `'counteer'`, `'counteeer'`, `'counteeer'` etc.
`counte{2,3}r` - matches strings like `'counteer'`, or `'counteeer'` but not `'counteeeer'`

A little explanation about "greediness". "Greedy" takes as many as possible, "non-greedy" takes as few as possible. For example, `'b+'` and `'b*'` applied to string `'abbbbc'` return `'bbbb'`, `'b+?'` returns `'b'`, `'b*?'` returns empty string, `'b{2,3}?'` returns `'bb'`, `'b{2,3}'` returns `'bbb'`.

Metacharacters - Alternatives

You can specify a series of alternatives for a pattern using `"|"` to separate them, so that `bit|bat|bot` will match any of `"bit"`, `"bat"`, or `"bot"` in the target string as would `"b(i|a|o)t)"`. The first alternative includes everything from the last pattern delimiter `("("`, `"["`, or the beginning of the pattern) up to the first `"|"`, and the last alternative contains everything from the last `"|"` to the next pattern delimiter. For this reason, it's common practice to include alternatives in parentheses, to minimize confusion about where they start and end.

Alternatives are tried from left to right, so the first alternative found for which the entire expression matches, is the one that is chosen. This means that alternatives are not necessarily greedy. For example: when matching `rou|rout` against `"routine"`, only the `"rou"` part will match, as that is the first alternative tried, and it successfully matches the target string (this might not seem important, but it is important when you are capturing matched text using parentheses.) Also remember that "|" is interpreted as a literal within square brackets, so if you write `[bit|bat|bot]`, you're really only matching `[biao|]`.

Examples:

`rou(tine|te)` - matches strings `'routine'` or `'route'`.

Metacharacters - Subexpressions

The bracketing construct `( ... )` may also be used for define regular subexpressions. Subexpressions are numbered based on the left to right order of their opening parenthesis. The first subexpression has number `‘1’`

Examples:

`(int){8,10}` matches strings which contain 8, 9 or 10 instances of the `‘int’`
`routi([0-9]|a+)e` matches `‘routi0e’`, `‘routi1e’` , `‘routine’`, `‘routinne’`, `‘routinnne’` etc.

Metacharacters - Backreferences

Metacharacters `\1` through `\9` are interpreted as backreferences. `\` matches previously matched subexpression `#`.

Examples:

`(.)\1+` matches `‘aaaa’` and `‘cc’`.
`(.+)\1+` matches `‘abab’` and `‘123123’`
`([‘”]?)(\d+)\1` matches `“13”` (in double quotes), or `‘4’` (in single quotes) or `77` (without quotes) etc.

Keyboard Shortcuts

Below is a complete list of keyboard shortcuts available in mikroBasic PRO for dsPIC30/33 and PIC24 IDE.

IDE Shortcuts	
F1	Help
Ctrl+N	New Unit
Ctrl+O	Open
Ctrl+Shift+O	Open Project
Ctrl+Shift+N	New Project
Ctrl+K	Close Project
Ctrl+F4	Close unit
Ctrl+Shift+E	Edit Project
Ctrl+F9	Build
Shift+F9	Build All
Ctrl+F11	Build And Program
Shift+F4	View Breakpoints
Ctrl+Shift+F5	Clear Breakpoints
F11	Start mE Programmer
Ctrl+Shift+F11	Project Manager
F12	Options
Alt + X	Close mikroBasic PRO for dsPIC30/33 and PIC24
Basic Editor Shortcuts	
F3	Find, Find Next
Shift+F3	Find Previous
Alt+F3	Grep Search, Find In Files
Ctrl+A	Select All
Ctrl+C	Copy
Ctrl+F	Find
Ctrl+R	Replace
Ctrl+P	Print
Ctrl+S	Save Unit
Ctrl+Shift+S	Save All
Ctrl+V	Paste
Ctrl+X	Cut
Ctrl+Y	Delete Entire Line
Ctrl+Z	Undo
Ctrl+Shift+Z	Redo

Advanced Editor Shortcuts	
Ctrl+Space	Code Assistant
Ctrl+Shift+Space	Parameters Assistant
Ctrl+D	Find Declaration
Ctrl+E	Incremental Search
Ctrl+L	Routine List
Ctrl+G	Goto Line
Ctrl+J	Insert Code Template
Ctrl+Shift+.	Comment Code
Ctrl+Shift+,	Uncomment Code
Ctrl+*number*	Goto Bookmark
Ctrl+Shift+number	Set Bookmark
Ctrl+Shift+I	Indent Selection
Ctrl+Shift+U	Unindent Selection
TAB	Indent Selection
Shift+TAB	Unindent Selection
Alt+Select	Select Columns
Ctrl+Alt+Select	Select Columns
Alt + Left Arrow	Fold Region (if available)
Alt + Right Arrow	Unfold Region (if available)
Ctrl+Alt+L	Convert Selection to Lowercase
Ctrl+Alt+U	Convert Selection to Uppercase
Ctrl+Alt+T	Convert to Titlecase
Ctrl+T	USART Terminal
Ctrl+Q	Quick Converter
mikroICD Debugger and Software Simulator Shortcuts	
F2	Jump To Interrupt
F4	Run to Cursor
F5	Toggle Breakpoint
F6	Run/Pause Debugger
F7	Step Into
F8	Step Over
F9	Start Debugger
Ctrl+F2	Stop Debugger

Ctrl+F5	Add to Watch List
Ctrl+F8	Step Out
Alt+D	Disassembly View
Shift+F5	Open Watch Window
Ctrl+Shift+A	Show Advanced Breakpoints

CHAPTER 3

mikroBasic PRO for dsPIC30/33 and PIC24 Command Line Options

Usage: `mBdsPIC.exe [-<opts> [-<opts>]] [<infile> [-<opts>]] [-<opts>]]`
Infile can be of `*.mbas, *.mcl` and `*.pld` type.

The following parameters and some more (see manual) are valid:

`-P <devicename>` : MCU for which compilation will be done.
`-FO <oscillator>` : Set oscillator [in MHz].
`-SP <directory>` : Add directory to the search path list.
`-N <filename>` : Output files generated to file path specified by filename.
`-B <directory>` : Save compiled binary files (`*.mcl`) to ‘directory’.
`-O` : Miscellaneous output options.
`-DBG` : Generate debug info.
`-L` : Check and rebuild new libraries.
`-DL` : Build all files as libraries.
`-UICD` : ICD build type.
`-EH <filename>` : Full EEPROM HEX file name with path.
`-Y` : Dynamic link for string literals.
`-LHF` : Generate Long hex format.
`-GC` : Generate COFF file.
`-PF` : Project file name.
`-RA` : Rebuild all sources in project.

Example:

```
mBdsPIC.exe -MSF -DBG -p30F4013 -Y -DL -O11111114 -fo80 -N”C:\Lcd\Lcd.mbpds” -SP”C:\
Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Defs”
            -SP”C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Uses” -SP”C:\
Lcd\” “__Lib_Math.mcl” “__Lib_MathDouble.mcl”
                “__Lib_System.mcl” “__Lib_Delays.mcl” “__Lib_LcdConsts.mcl” “__Lib_Lcd.
mcl” “Lcd.mbas”
```

Parameters used in the example:

`-MSF`: Short Message Format; used for internal purposes by IDE.
`-DBG`: Generate debug info.
`-p30F4013`: MCU 30F4013 selected.
`-Y`: Dynamic link for string literals enabled.
`-DL`: All files built as libraries.
`-O11111114`: Miscellaneous output options.
`-fo80`: Set oscillator frequency [in MHz].
`-N”C:\Lcd\Lcd.mbpds” -SP”C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Defs”`: Output files generated to file path specified by filename.
`-SP”C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Defs”`: Add directory to the search path list.
`-SP”C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Uses”`: Add directory to the search path list.
`-SP”C:\Lcd\”`: Add directory to the search path list.
`- “Lcd.mbas” “__Lib_Math.mcl” “__Lib_MathDouble.mcl” “__Lib_System.mcl” “__Lib_Delays.mcl” “__Lib_LcdConsts.mcl” “__Lib_Lcd.mcl”`: Specify input files.

CHAPTER 4

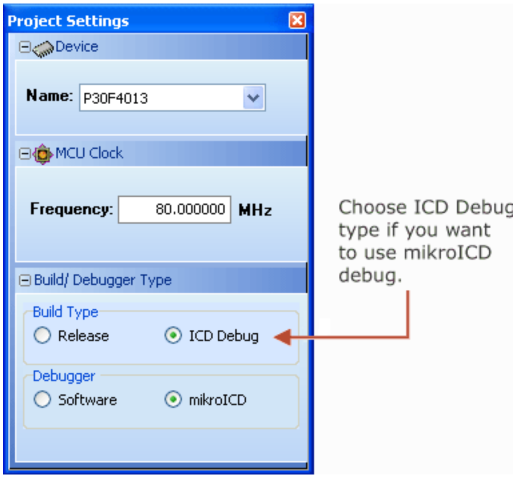
mikroICD (In-Circuit Debugger)

Introduction

The mikroICD is a highly effective tool for a **Real-Time debugging** on hardware level. The mikroICD debugger enables you to execute the mikroBasic PRO for dsPIC30/33 and PIC24 program on a host dsPIC30/33 or PIC24 microcontroller and view variable values, Special Function Registers (SFR), RAM, CODE and EEPROM memory along with the mikroICD code execution on hardware.

Step No. 1

If you have appropriate hardware and software for using the mikroICD select **mikroICD Debug** Build Type before compiling the project.



Project Settings

Device

Name: P30F4013

MCU Clock

Frequency: 80.000000 MHz

Build/ Debugger Type

Build Type: ○ Release ● ICD Debug

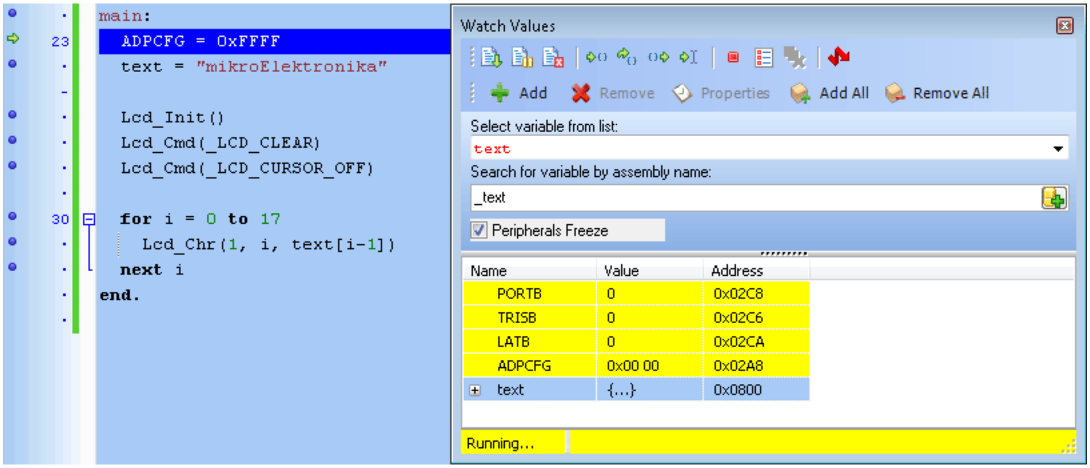
Debugger: ○ Software ● mikroICD

Choose ICD Debug type if you want to use mikroICD debug.

Now, compile the project by pressing Ctrl + F9, or by pressing Build Icon on Build Toolbar.

Step No. 2

Run the mikroICD by selecting **Run › Start Debugger** from the drop-down menu or by clicking the Start Debugger Icon . Starting the Debugger makes more options available: Step Into, Step Over, Run to Cursor, etc. Line that is to be executed is color highlighted (blue by default). There is also notification about the program execution and it can be found in the Watch Window (yellow status bar). Note that some functions take more time to execute; execution is indicated with "Running..." message in the Watch Window Status Bar.



```
main:
  ADPCFG = 0xFFFF
  text = "mikroElektronika"

  Lcd_Init()
  Lcd_Cmd(_LCD_CLEAR)
  Lcd_Cmd(_LCD_CURSOR_OFF)

  for i = 0 to 17
    Lcd_Chr(1, i, text[i-1])
  next i
end.
```

Select variable from list: text

Search for variable by assembly name: _text

☑ Peripherals Freeze

Name	Value	Address
PORTB	0	0x02C8
TRISB	0	0x02C6
LATB	0	0x02CA
ADPCFG	0x00 00	0x02A8
⊞ text	{...}	0x0800

Running...

Related topics: mikroICD Debugger Example, Debug Windows, Debugger Options

mikroICD Debugger Options

Debugger Options

Name	Description	Function Key	Toolbar Icon
`Start Debugger`	Starts Debugger.	F9	📄⬇
`Run/Pause Debugger`	Run/Pause Debugger.	F6	📄⏸
`Stop Debugger`	Stop Debugger.	Ctrl + F2	📄✖
`Step Into`	Executes the current program line, then halts. If the executed program line calls another routine, the debugger steps into the routine and halts after executing the first instruction within it.	F7	➔()
`Step Over`	Executes the current program line, then halts. If the executed program line calls another routine, the debugger will not step into it. The whole routine will be executed and the debugger halts at the first instruction following the call.	F8	↷()
`Step Out`	Executes all remaining program lines within the subroutine. The debugger halts immediately upon exiting the subroutine. this option is provided with the PIC18 microcontroller family, but not with the PIC16 family.	F8	()➔
`Run To Cursor`	Executes the program until reaching the cursor position.	Ctrl + F8	➔I
`Toggle Breakpoint`	Toggle breakpoints option sets new breakpoints or removes those already set at the current cursor position.	F5	■

Related topics: Run Menu, Debug Toolbar

mikroICD Debugger Example

Here is a step-by-step mikroICD Debugger Example.

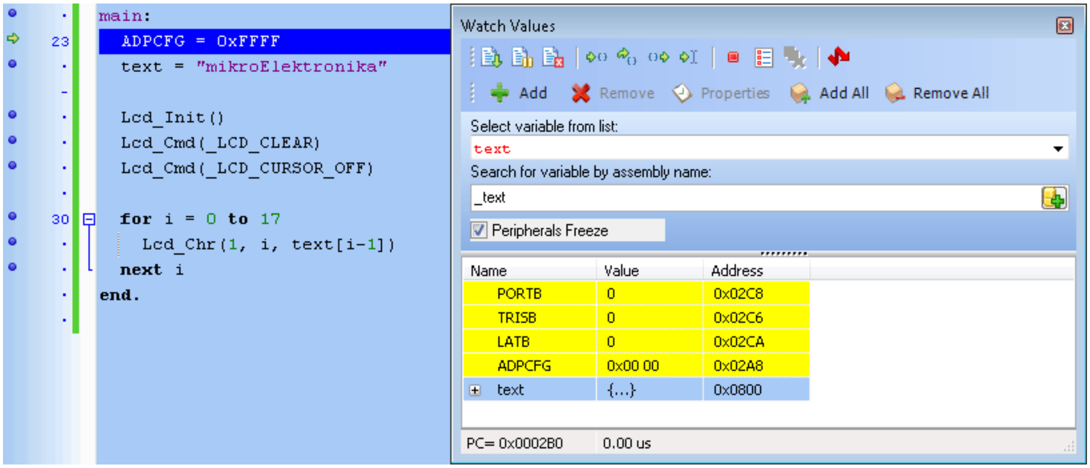
Step No. 1

First you have to write a program. We will show how the mikroICD works using this example:

```
program Lcd_Test

' LCD module connections
dim LCD_RS as sbit at LATD0_bit
dim LCD_EN as sbit at LATD1_bit
dim LCD_D4 as sbit at LATB0_bit
dim LCD_D5 as sbit at LATB1_bit
dim LCD_D6 as sbit at LATB2_bit
dim LCD_D7 as sbit at LATB3_bit

dim LCD_RS_Direction as sbit at TRISD0_bit
dim LCD_EN_Direction as sbit at TRISD1_bit
dim LCD_D4_Direction as sbit at TRISB0_bit
dim LCD_D5_Direction as sbit at TRISB1_bit
dim LCD_D6_Direction as sbit at TRISB2_bit
dim LCD_D7_Direction as sbit at TRISB3_bit
' End LCD module connections

dim text as char[16]
       i as byte

main:
  ADPCFG = 0xFFFF
  text = "mikroElektronika"

  Lcd_Init()
  Lcd_Cmd(_LCD_CLEAR)
  Lcd_Cmd(_LCD_CURSOR_OFF)

  for i = 0 to 17
    Lcd_Chr(1, i, text[i-1])
  next i
end.
```

Step No. 2

After successful compilation and MCU programming press **F9** to start the mikroICD. After the mikroICD initialization a blue active line should appear.



```
main:
  ADPCFG = 0xFFFF
  text = "mikroElektronika"

  Lcd_Init()
  Lcd_Cmd(_LCD_CLEAR)
  Lcd_Cmd(_LCD_CURSOR_OFF)

  for i = 0 to 17
    Lcd_Chr(1, i, text[i-1])
  next i
end.
```

Select variable from list: text

Search for variable by assembly name: _text

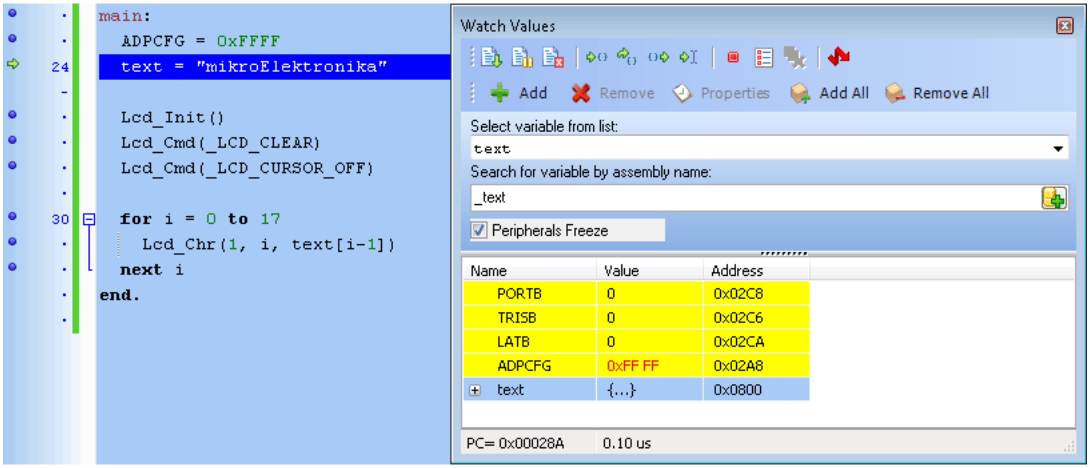
☑ Peripherals Freeze

Name	Value	Address
PORTB	0	0x02C8
TRISB	0	0x02C6
LATB	0	0x02CA
ADPCFG	0x00 00	0x02A8
⊞ text	{...}	0x0800

PC= 0x0002B0 0.00 us

Step No. 3

We will debug the program line by line. Pressing **[F8]** we are executing code line by line. However, it is not recommended that user does not use Step Into **[F7]** and Step Over **[F8]** over Delays routines and routines containing delays. Instead use Run to cursor **[F4]** and Breakpoints functions.
All changes are read from MCU and loaded into Watch Window. Note that **TRISB** changed its value from 255 to 0.



```
main:
  ADPCFG = 0xFFFF
  text = "mikroElektronika"

  Lcd_Init()
  Lcd_Cmd(_LCD_CLEAR)
  Lcd_Cmd(_LCD_CURSOR_OFF)

  for i = 0 to 17
    Lcd_Chr(1, i, text[i-1])
  next i
end.
```

Select variable from list: text

Search for variable by assembly name: _text

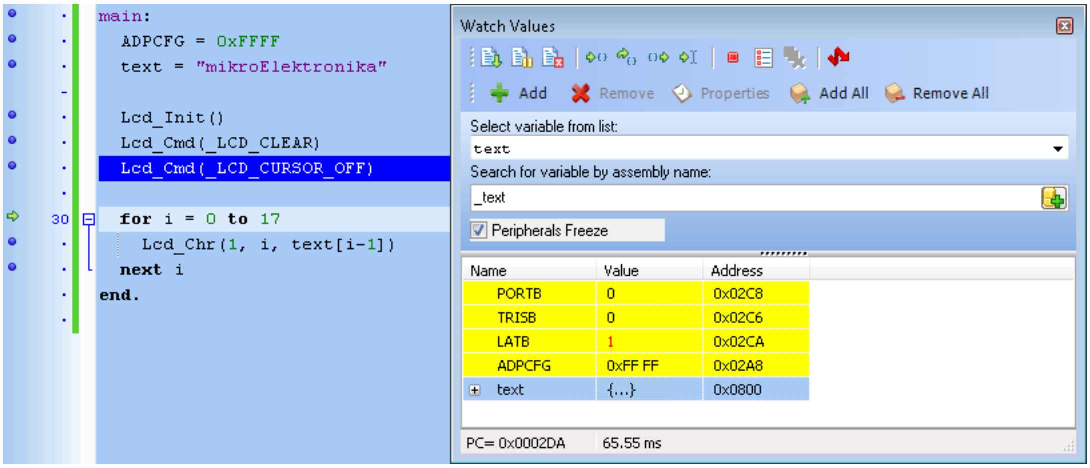
☑ Peripherals Freeze

Name	Value	Address
PORTB	0	0x02C8
TRISB	0	0x02C6
LATB	0	0x02CA
ADPCFG	0xFF FF	0x02A8
⊞ text	{...}	0x0800

PC= 0x00028A 0.10 us

Step No. 4

Step Into **[F7]**, Step Over **[F8]** and Step Out **[Ctrl+F8]** are mikroICD debugger functions that are used in stepping mode. There is also a Real-Time mode supported by the mikroICD. Functions that are used in the Real-Time mode are Run/Pause Debugger **[F6]** and Run to cursor **[F4]**. Pressing **F4** executes the code until the program reaches the cursor position line.



```
main:
  ADPCFG = 0xFFFF
  text = "mikroElektronika"

  Lcd_Init()
  Lcd_Cmd(_LCD_CLEAR)
  Lcd_Cmd(_LCD_CURSOR_OFF)

  for i = 0 to 17
    Lcd_Chr(1, i, text[i-1])
  next i
end.
```

Select variable from list: text

Search for variable by assembly name: _text

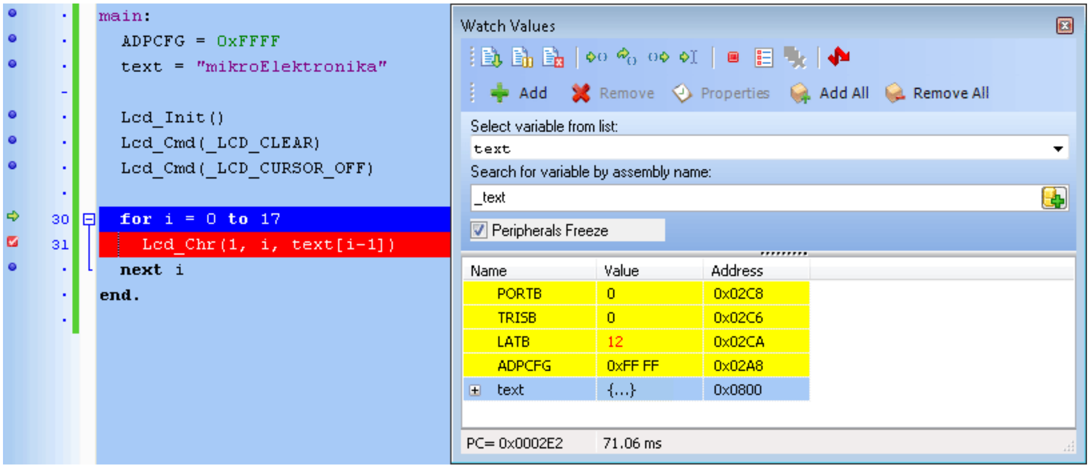
☑ Peripherals Freeze

Name	Value	Address
PORTB	0	0x02C8
TRISB	0	0x02C6
LATB	1	0x02CA
ADPCFG	0xFF FF	0x02A8
⊞ text	{...}	0x0800

PC= 0x0002DA 65.55 ms

Step No. 5

Run(Pause) Debugger **[F6]** and Toggle Breakpoints **[F5]** are mikroICD debugger functions that are used in the Real-Time mode. Pressing **F5** marks the line selected by the user for breakpoint. **F6** executes code until the breakpoint is reached. After reaching the breakpoint Debugger halts. Here in our example we will use breakpoints for writing “mikroElektronika” on Lcd char by char. Breakpoint is set on Lcd_Chr and the program will stop every time this function is reached. After reaching breakpoint we must press **F6** again to continue the program execution.



```
main:
  ADPCFG = 0xFFFF
  text = "mikroElektronika"

  Lcd_Init()
  Lcd_Cmd(_LCD_CLEAR)
  Lcd_Cmd(_LCD_CURSOR_OFF)

  for i = 0 to 17
    Lcd_Chr(1, i, text[i-1])
  next i
end.
```

Select variable from list: text

Search for variable by assembly name: _text

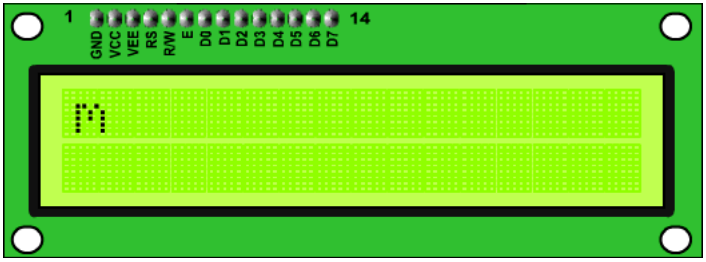
☑ Peripherals Freeze

Name	Value	Address
PORTB	0	0x02C8
TRISB	0	0x02C6
LATB	12	0x02CA
ADPCFG	0xFF FF	0x02A8
⊞ text	{...}	0x0800

PC= 0x0002E2 71.06 ms

Breakpoints are divided into two groups: hardware and software breakpoints. The hardware breakpoints are placed in the MCU and provide fastest debugging. Number of hardware breakpoints is limited (4 for PIC24 and dsPIC33 family, for dsPIC30 family this number depends on the MCU used). If all hardware brekpoints are used, then the next breakpoint will be software breakpoint. These breakpoints are placed inside the mikroICD and simulate hardware breakpoints. Software breakpoints are much slower than hardware breakpoints. These differences between hardware and software breakpoints are not visible in the mikroICD software but their different timings are quite notable.That's why it is important to know that there are two types of breakpoints.

The picture below demonstrates step-by-step execution of the code used in above mentioned examples.



Common Errors:

- Trying to program the MCU while the mikroICD is active.
- Trying to debug **Release** build version of the program with the mikroICD debugger.
- Trying to debug program code which has been changed, but has not been compiled and programmed into the MCU.
- Trying to select line that is empty for Run to cursor [**F4**] and Toggle Breakpoints [**F5**] functions.
- Trying to debug MCU with mikroICD while Watch Dog Timer is enabled.
- Trying to debug MCU with mikroICD while Power Up Timer is enabled.
- Trying to **Step Into** [**F7**] the mikroBasic PRO for dsPIC30/33 and PIC24 Library routines. Use **Step Over** [**F8**] command for these routines.
- It is not possible to force Code Protect while trying to debug MCU with mikroICD.
- Trying to debug MCU with mikroICD with pull-up resistors set to ON on RB6 and RB7.

Related topics: mikroICD Debugger, Debug Windows, Debugger Options

mikroICD Debugger Windows

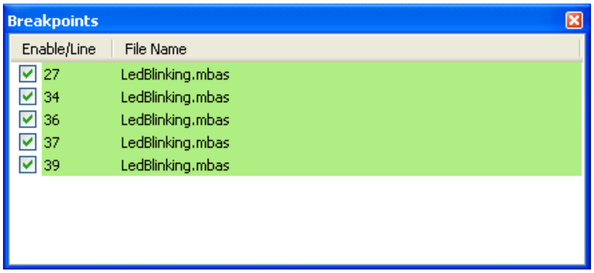
Debug Windows

This section provides an overview of available Debug Windows in mikroBaisc PRO for dsPIC30/33 and PIC24:

- Breakpoints Window
- Watch Values Window
- RAM Window
- Stopwatch Window
- EEPROM Watch Window
- Code Watch Window

Breakpoints Window

The Breakpoints window manages the list of currently set breakpoints in the project. Doubleclicking the desired breakpoint will cause cursor to navigate to the corresponding location in source code.
In situations when multiple breakpoints are used within the code, it is sometimes handy to enable/disable certain breakpoints. To do this, just check/uncheck the desired breakpoint using the checkbox in front of the breakpoint's name.



Breakpoints

Enable/Line	File Name
☑ 27	LedBlinking.mbas
☑ 34	LedBlinking.mbas
☑ 36	LedBlinking.mbas
☑ 37	LedBlinking.mbas
☑ 39	LedBlinking.mbas

Watch Values Window

Watch Values Window is the main Debugger window which allows you to monitor program execution. To show the Watch Values Window, select **Debug Windows › Watch** from the **View** drop-down menu.

The Watch Values Window displays variables and registers of the MCU, with their addresses and values. Values are updated along with the code execution. Recently changed items are coloured red.

There are two ways to add variable/register into the watch list:

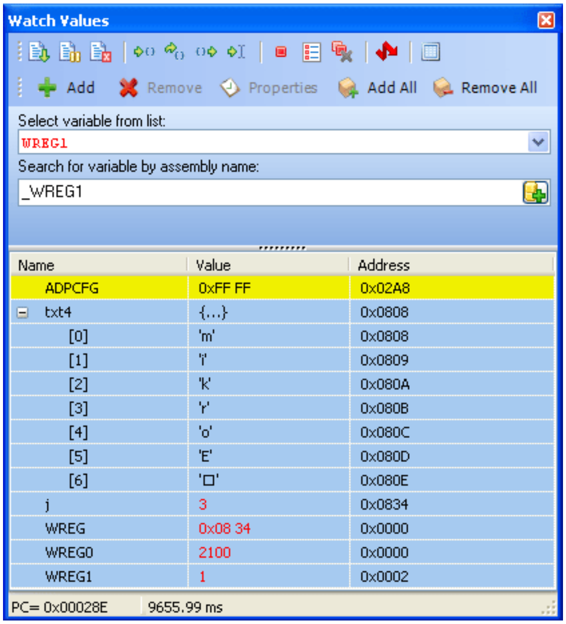
- by its real name (variable's name in program code). Just select wanted variable/register from **Select variable from list** drop-down menu and click the Add button.
- by its name ID (assembly variable name). Simply type name ID of the variable/register you want to display into **Search for variable by assemby name** box and click the Add button.

Also, it is possible to add all variables in the Watch Values Window by clicking Add All button.

To remove a variable from the Watch Values Window, just select the variable that you want to remove and then click the Remove button, or press the Delete key.

It is possible to remove all variables from the Watch Values Window by clicking Remove All button.

You can also expand/collapse complex variables i.e. struct type variables, strings, etc, by clicking the appropriate button (⊞ or ⊟) beside variable name.



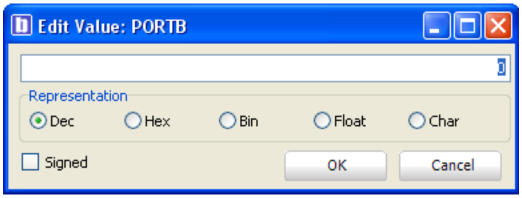
Select variable from list: WREG1

Search for variable by assembly name: _WREG1

Name	Value	Address
ADPCFG	0xFF FF	0x02A8
⊟ txt4	{...}	0x0808
[0]	'm'	0x0808
[1]	'i'	0x0809
[2]	'k'	0x080A
[3]	'r'	0x080B
[4]	'o'	0x080C
[5]	'E'	0x080D
[6]	'□'	0x080E
j	3	0x0834
WREG	0x08 34	0x0000
WREG0	2100	0x0000
WREG1	1	0x0002

PC= 0x00028E 9655.99 ms

Double clicking a variable or clicking the Properties button opens the Edit Value window in which you can assign a new value to the selected variable/register. Also, you can choose the format of variable/register representation between decimal, hexadecimal, binary, float or character. All representations except float are unsigned by default. For signed representation click the check box next to the **Signed** label.



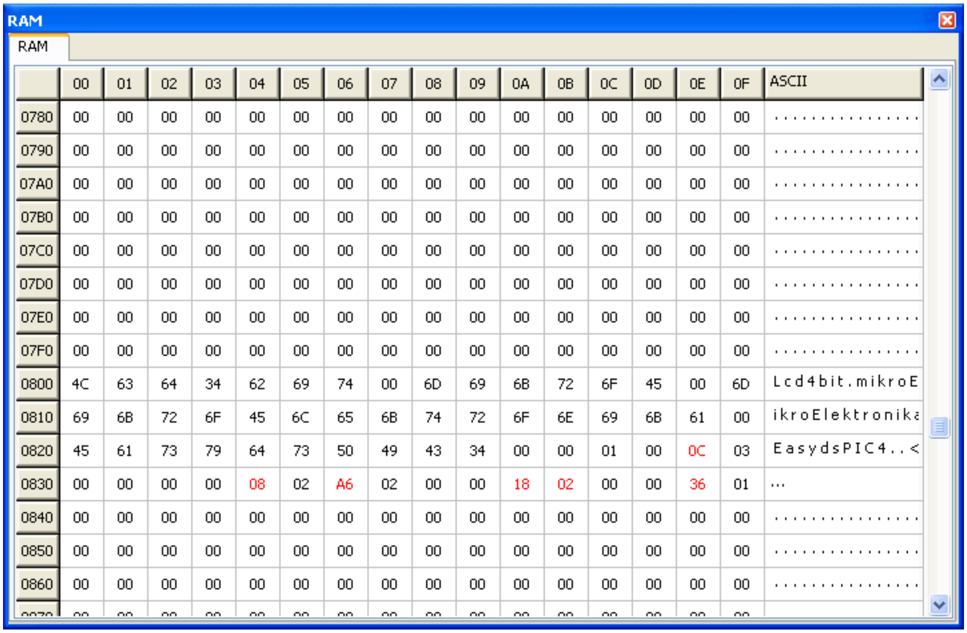
An item's value can also be changed by double clicking item's value field and typing the new value directly.

RAM Window

The RAM Window is available from the drop-down menu, **View › Debug Windows › RAM**.

The RAM Window displays the map of MCU's RAM, with recently changed items colored red. The user can edit and change the values in the RAM window.

mikroICD Specific: RAM window content will be written to the MCU before the next instruction execution.

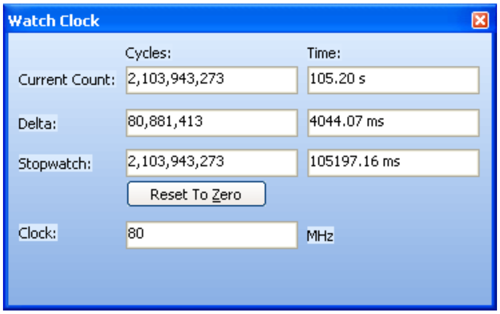


	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0780	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0790	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0800	4C	63	64	34	62	69	74	00	6D	69	6B	72	6F	45	00	6D	Lcd4bit.mikroE
0810	69	6B	72	6F	45	6C	65	6B	74	72	6F	6E	69	6B	61	00	ikroElektronika
0820	45	61	73	79	64	73	50	49	43	34	00	00	01	00	0C	03	EasydsPIC4..<
0830	00	00	00	00	08	02	A6	02	00	00	18	02	00	00	36	01	...
0840	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0850	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0860	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Stopwatch Window

The Software Simulator Stopwatch Window is available from the drop-down menu, **View › Debug Windows › Stopwatch**.

The Stopwatch Window displays a `Current Count` of cycles/time since the last Software Simulator action. `Stopwatch` measures the execution time (number of cycles) from the moment Software Simulator has started and can be reset at any time.
`Delta` represents the number of cycles between the lines where Software Simulator action has started and ended.



	Cycles:	Time:
Current Count:	2,103,943,273	105.20 s
Delta:	80,881,413	4044.07 ms
Stopwatch:	2,103,943,273	105197.16 ms
Clock:	80	MHz

Notes:

- The user can change the clock in the Stopwatch Window, which will recalculate values for the latest specified frequency.
- Changing the clock in the Stopwatch Window does not affect actual project settings – it only provides a simulation.
- Stopwatch is available only when Software Simulator is selected as a debugger.

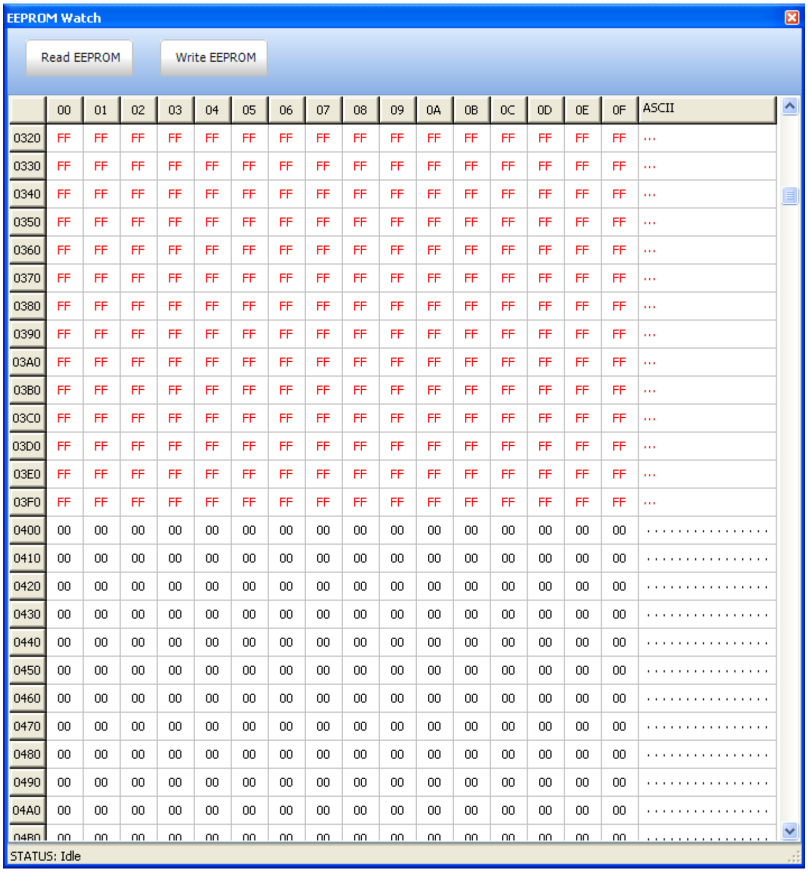
EEPROM Watch Window

Note : EEPROM Watch Window is available only when mikroICD is selected as a debugger.

To show the EEPROM Watch Window, select **Debug Windows › EEPROM** from the **View** drop-down menu.
The EEPROM Watch Window shows current content of the MCU's internal EEPROM memory.

There are two action buttons concerning the EEPROM Watch Window:

Read EEPROM	- Reads data from MCU's internal EEPROM memory and loads it up into the EEPROM window.
Write EEPROM	- Writes data from the EEPROM window into MCU's internal EEPROM memory.



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0320	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0330	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0340	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0350	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0360	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0370	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0380	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0390	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
03A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
03B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
03C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
03D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
03E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
03F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0400	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0410	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0420	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0430	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0440	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0450	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0460	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0470	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0480	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0490	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
04A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
04B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

STATUS: Idle

Code Watch Window

Note: Code Watch Window is available only when mikroICD is selected as a debugger.

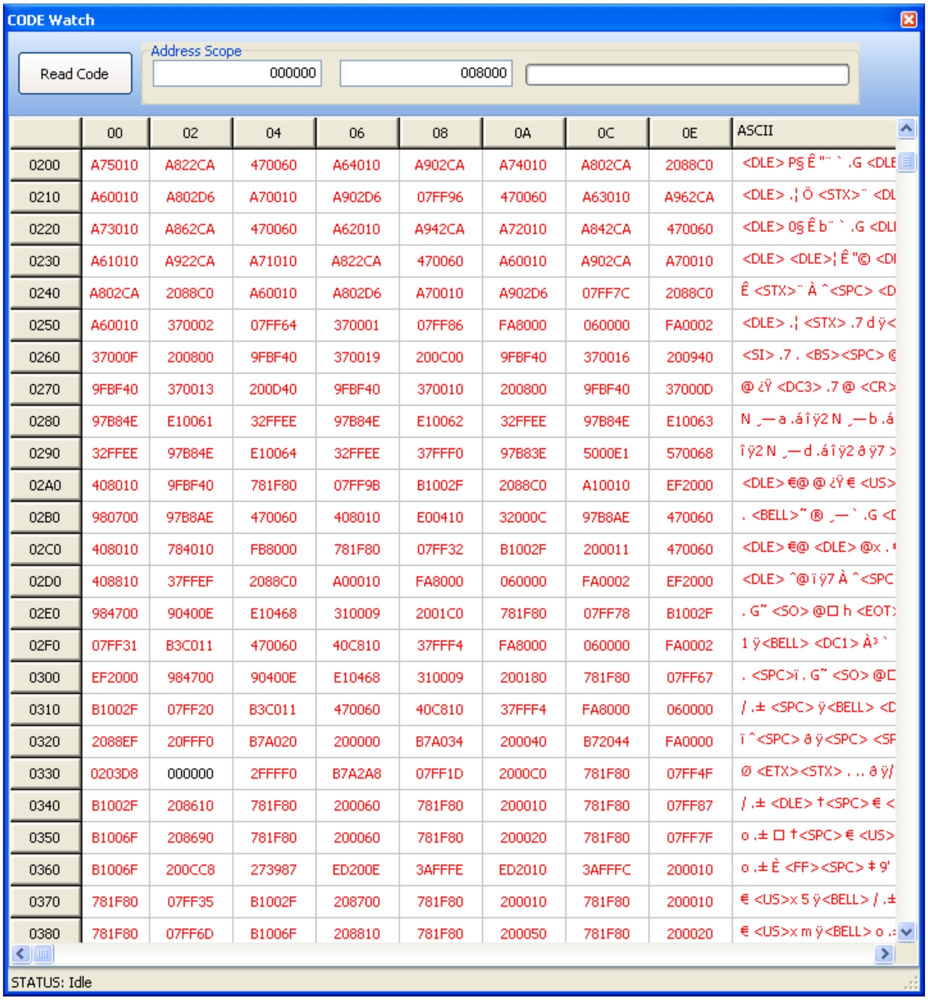
To show the Code Watch Window, select **Debug Windows › Code** from the **View** drop-down menu.

The Code Watch Window shows code (hex format) written into the MCU.

There is one action button concerning the Code Watch Window:

Read Code - Reads code from the MCU and loads it up into the Code Window. Code reading is resources consuming operation so the user should wait until the reading is over.

Also, you can set an address scope in which hex code will be read.



	00	02	04	06	08	0A	0C	0E	ASCII
0200	A75010	A822CA	470060	A64010	A902CA	A74010	A802CA	2088C0	<DLE> P§ Ê "¨ ` .G <DLE
0210	A60010	A802D6	A70010	A902D6	07FF96	470060	A63010	A962CA	<DLE> .¦ Ö <STX>¨ <DL
0220	A73010	A862CA	470060	A62010	A942CA	A72010	A842CA	470060	<DLE> 0§ Ê b¨ ` .G <DL
0230	A61010	A922CA	A71010	A822CA	470060	A60010	A902CA	A70010	<DLE> <DLE>¦ Ê "© <D
0240	A802CA	2088C0	A60010	A802D6	A70010	A902D6	07FF7C	2088C0	Ê <STX>¨ À ˆ<SPC> <D
0250	A60010	370002	07FF64	370001	07FF86	FA8000	060000	FA0002	<DLE> .¦ <STX> .7 d ÿ<
0260	37000F	200800	9FBF40	370019	200C00	9FBF40	370016	200940	<SI> .7 . <BS><SPC> @
0270	9FBF40	370013	200D40	9FBF40	370010	200800	9FBF40	37000D	@ ¿Ÿ <DC3> .7 @ <CR>
0280	97B84E	E10061	32FFEE	97B84E	E10062	32FFEE	97B84E	E10063	N ¸—a .áîÿ2 N ¸—b .á
0290	32FFEE	97B84E	E10064	32FFEE	37FFF0	97B83E	5000E1	570068	îÿ2 N ¸—d .áîÿ2 ð ÿ7 >
02A0	408010	9FBF40	781F80	07FF9B	B1002F	2088C0	A10010	EF2000	<DLE> €@ @ ¿Ÿ € <US>
02B0	980700	97B8AE	470060	408010	E00410	32000C	97B8AE	470060	. <BELL>˜ ® ¸—` .G <D
02C0	408010	784010	FB8000	781F80	07FF32	B1002F	200011	470060	<DLE> €@ <DLE> @x .
02D0	408810	37FFEF	2088C0	A00010	FA8000	060000	FA0002	EF2000	<DLE> ˆ@ ïÿ7 À ˆ<SPC
02E0	984700	90400E	E10468	310009	2001C0	781F80	07FF78	B1002F	. G˜ <SO> @□ h <EOT
02F0	07FF31	B3C011	470060	40C810	37FFF4	FA8000	060000	FA0002	1 ÿ<BELL> <DC1> À³ `
0300	EF2000	984700	90400E	E10468	310009	200180	781F80	07FF67	. <SPC>ï . G˜ <SO> @□
0310	B1002F	07FF20	B3C011	470060	40C810	37FFF4	FA8000	060000	/ .± <SPC> ÿ<BELL> <D
0320	2088EF	20FFF0	B7A020	200000	B7A034	200040	B72044	FA0000	ï ˆ<SPC> ð ÿ<SPC> <SP
0330	0203D8	000000	2FFFF0	B7A2A8	07FF1D	2000C0	781F80	07FF4F	Ø <ETX><STX> . .. ð ÿ/
0340	B1002F	208610	781F80	200060	781F80	200010	781F80	07FF87	/ .± <DLE> †<SPC> € <
0350	B1006F	208690	781F80	200060	781F80	200020	781F80	07FF7F	o .± □ †<SPC> € <US>
0360	B1006F	200CC8	273987	ED200E	3AFFFE	ED2010	3AFFFC	200010	o .± È <FF><SPC> ‡ 9'
0370	781F80	07FF35	B1002F	208700	781F80	200010	781F80	200010	€ <US>x 5 ÿ<BELL> / .±
0380	781F80	07FF6D	B1006F	208810	781F80	200050	781F80	200020	€ <US>x m ÿ<BELL> o .:

STATUS: Idle

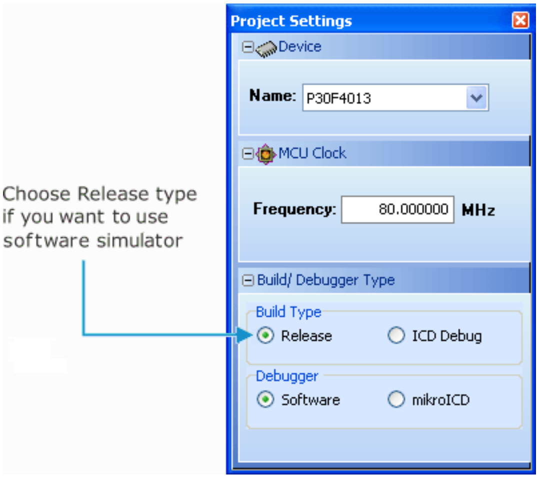
CHAPTER 5

Software Simulator Overview

Software Simulator

The Source-level Software Simulator is an integral component of the mikroBasic PRO for dsPIC30/33 and PIC24 environment. It is designed to simulate operations of the Microchip dsPIC30/33 and PIC24 MCUs and assist the users in debugging code written for these devices.

Upon completion of writing your program, choose **Release** build Type in the Project Settings window:



Choose Release type if you want to use software simulator

Project Settings

Device

Name: P30F4013

MCU Clock

Frequency: 80.000000 MHz

Build/ Debugger Type

Build Type
● Release ○ ICD Debug

Debugger
● Software ○ mikroICD

After you have successfuly compiled your project, you can run the Software Simulator by selecting **Run › Start Debugger** from the drop-down menu, or by clicking the Start Debugger Icon from the Debugger Toolbar.

Starting the Software Simulator makes more options available: Step Into, Step Over, Step Out, Run to Cursor, etc. Line that is to be executed is color highlighted (blue by default).

Note: The Software Simulator simulates the program flow and execution of instruction lines, but it cannot fully emulate dsPIC device behavior, i.e. it doesn't update timers, interrupt flags, etc.

Related topics: Software Simulator Debug Windows, Software Simulator Debugger Options

Software Simulator Debug Windows

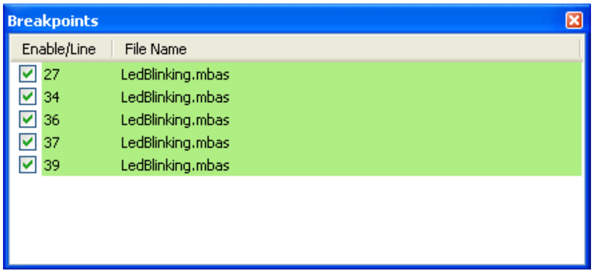
Debug Windows

This section provides an overview of available Debug Windows in mikroBasic PRO for dsPIC30/33 and PIC24:

- Breakpoints Window
- Watch Values Window
- RAM Window
- Stopwatch Window
- EEPROM Watch Window
- Code Watch Window

Breakpoints Window

The Breakpoints window manages the list of currently set breakpoints in the project. Doubleclicking the desired breakpoint will cause cursor to navigate to the corresponding location in source code.
In situations when multiple breakpoints are used within the code, it is sometimes handy to enable/disable certain breakpoints. To do this, just check/uncheck the desired breakpoint using the checkbox in front of the breakpoint's name.



Breakpoints

Enable/Line	File Name
☑ 27	LedBlinking.mbas
☑ 34	LedBlinking.mbas
☑ 36	LedBlinking.mbas
☑ 37	LedBlinking.mbas
☑ 39	LedBlinking.mbas

Watch Values Window

Watch Values Window is the main Debugger window which allows you to monitor program execution. To show the Watch Values Window, select **Debug Windows › Watch** from the **View** drop-down menu.

The Watch Values Window displays variables and registers of the MCU, with their addresses and values. Values are updated along with the code execution. Recently changed items are coloured red.

There are two ways to add variable/register into the watch list:

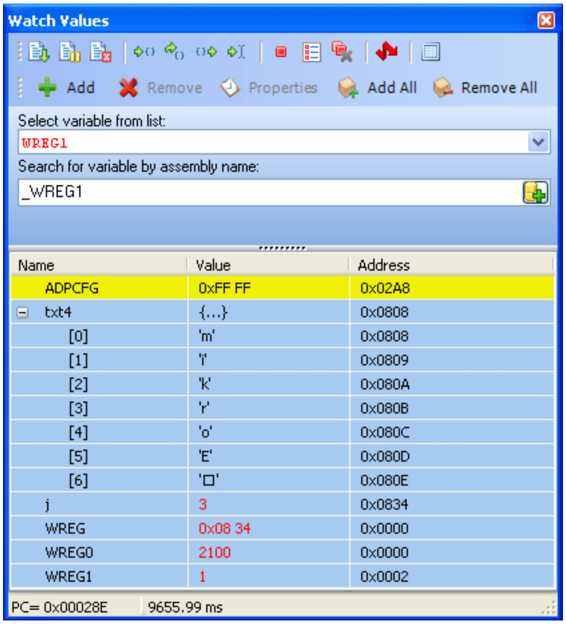
- by its real name (variable's name in program code). Just select wanted variable/register from **Select variable from list** drop-down menu and click the Add button.
- by its name ID (assembly variable name). Simply type name ID of the variable/register you want to display into **Search for variable by assemby name** box and click the Add button.

Also, it is possible to add all variables in the Watch Values Window by clicking Add All button.

To remove a variable from the Watch Values Window, just select the variable that you want to remove and then click the Remove button, or press the Delete key.

It is possible to remove all variables from the Watch Values Window by clicking Remove All button.

You can also expand/collapse complex variables i.e. struct type variables, strings, etc, by clicking the appropriate button (⊞ or ⊟) beside variable name.



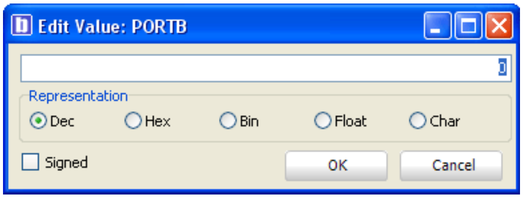
Select variable from list: WREG1

Search for variable by assembly name: _WREG1

Name	Value	Address
ADPCFG	0xFF FF	0x02A8
⊟ txt4	{...}	0x0808
[0]	'm'	0x0808
[1]	'i'	0x0809
[2]	'k'	0x080A
[3]	'r'	0x080B
[4]	'o'	0x080C
[5]	'E'	0x080D
[6]	'□'	0x080E
j	3	0x0834
WREG	0x08 34	0x0000
WREG0	2100	0x0000
WREG1	1	0x0002

PC= 0x00028E 9655.99 ms

Double clicking a variable or clicking the Properties button opens the Edit Value window in which you can assign a new value to the selected variable/register. Also, you can choose the format of variable/register representation between decimal, hexadecimal, binary, float or character. All representations except float are unsigned by default. For signed representation click the check box next to the **Signed** label.



0

Representation: ● Dec ○ Hex ○ Bin ○ Float ○ Char

☐ Signed

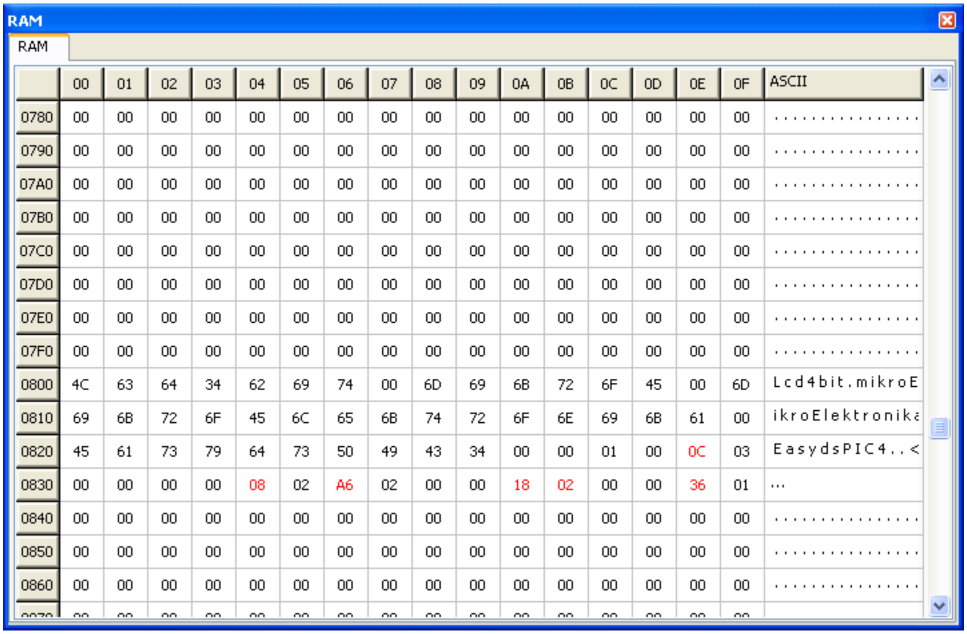
An item's value can also be changed by double clicking item's value field and typing the new value directly.

RAM Window

The RAM Window is available from the drop-down menu, **View › Debug Windows › RAM**.

The RAM Window displays the map of MCU’s RAM, with recently changed items colored red. The user can edit and change the values in the RAM window.

mikroICD Specific: RAM window content will be written to the MCU before the next instruction execution.

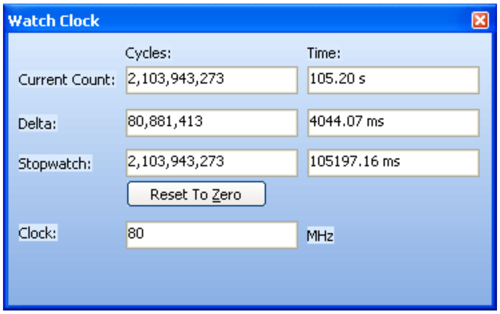


	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0780	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0790	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
07F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0800	4C	63	64	34	62	69	74	00	6D	69	6B	72	6F	45	00	6D	Lcd4bit.mikroE
0810	69	6B	72	6F	45	6C	65	6B	74	72	6F	6E	69	6B	61	00	ikroElektronika
0820	45	61	73	79	64	73	50	49	43	34	00	00	01	00	0C	03	EasydsPIC4..<
0830	00	00	00	00	08	02	A6	02	00	00	18	02	00	00	36	01	...
0840	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0850	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0860	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Stopwatch Window

The Software Simulator Stopwatch Window is available from the drop-down menu, **View › Debug Windows › Stopwatch**.

The Stopwatch Window displays a `Current Count` of cycles/time since the last Software Simulator action. `Stopwatch` measures the execution time (number of cycles) from the moment Software Simulator has started and can be reset at any time.
`Delta` represents the number of cycles between the lines where Software Simulator action has started and ended.



	Cycles:	Time:
Current Count:	2,103,943,273	105.20 s
Delta:	80,881,413	4044.07 ms
Stopwatch:	2,103,943,273	105197.16 ms
Clock:	80	MHz

Notes:

- The user can change the clock in the Stopwatch Window, which will recalculate values for the latest specified frequency.
- Changing the clock in the Stopwatch Window does not affect actual project settings – it only provides a simulation.
- Stopwatch is available only when Software Simulator is selected as a debugger.

EEPROM Watch Window

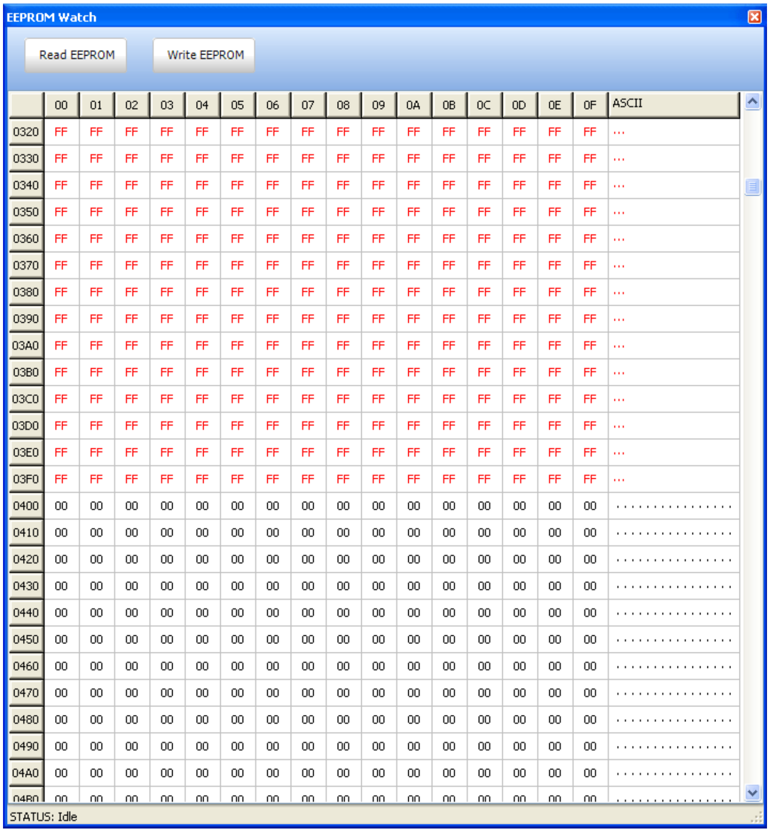
Note: EEPROM Watch Window is available only when mikroICD is selected as a debugger.

To show the EEPROM Watch Window, select **Debug Windows › EEPROM** from the **View** drop-down menu.
The EEPROM Watch Window shows current content of the MCU's internal EEPROM memory.

There are two action buttons concerning the EEPROM Watch Window:

Read EEPROM - Reads data from MCU's internal EEPROM memory and loads it up into the EEPROM window.

Write EEPROM - Writes data from the EEPROM window into MCU's internal EEPROM memory.



```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
0320  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
0330  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
0340  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
0350  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
0360  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
0370  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
0380  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
0390  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
03A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
03B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
03C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
03D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
03E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
03F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
0400  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0410  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0420  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0430  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0440  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0450  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0460  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0470  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0480  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0490  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
04A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
04B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
STATUS: Idle
```

Code Watch Window

Note: Code Watch Window is available only when mikroICD is selected as a debugger.

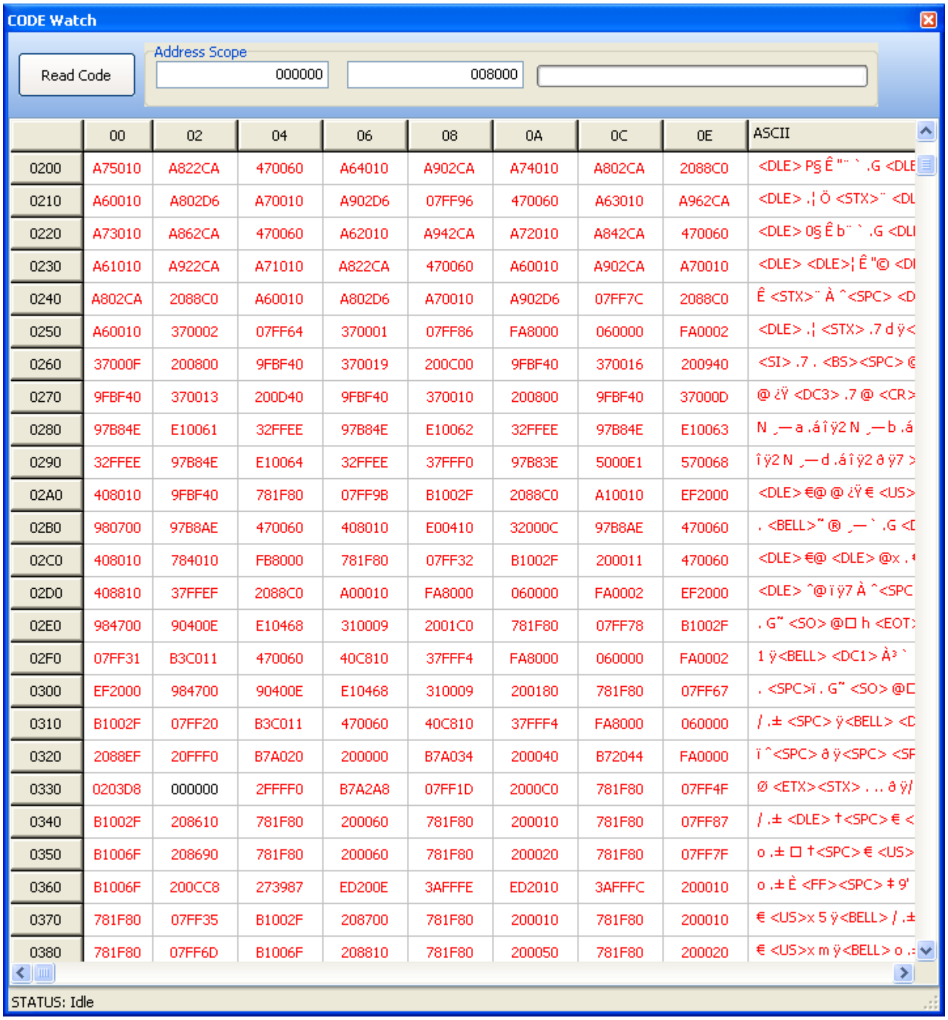
To show the Code Watch Window, select **Debug Windows › Code** from the **View** drop-down menu.

The Code Watch Window shows code (hex format) written into the MCU.

There is one action button concerning the Code Watch Window:

- Read Code - Reads code from the MCU and loads it up into the Code Window. Code reading is resources consuming operation so the user should wait until the reading is over.

Also, you can set an address scope in which hex code will be read.



000000
008000

	00	02	04	06	08	0A	0C	0E	ASCII
0200	A75010	A822CA	470060	A64010	A902CA	A74010	A802CA	2088C0	<DLE> P§ Ê "¨ ` .G <DLE
0210	A60010	A802D6	A70010	A902D6	07FF96	470060	A63010	A962CA	<DLE> .¦ Ö <STX>¨ <DL
0220	A73010	A862CA	470060	A62010	A942CA	A72010	A842CA	470060	<DLE> 0§ Ê b¨ ` .G <DL
0230	A61010	A922CA	A71010	A822CA	470060	A60010	A902CA	A70010	<DLE> <DLE>¦ Ê "© <D
0240	A802CA	2088C0	A60010	A802D6	A70010	A902D6	07FF7C	2088C0	Ê <STX>¨ À ˆ<SPC> <D
0250	A60010	370002	07FF64	370001	07FF86	FA8000	060000	FA0002	<DLE> .¦ <STX> .7 d ÿ<
0260	37000F	200800	9FBF40	370019	200C00	9FBF40	370016	200940	<SI> .7 . <BS><SPC> @
0270	9FBF40	370013	200D40	9FBF40	370010	200800	9FBF40	37000D	@ ¿Ÿ <DC3> .7 @ <CR>
0280	97B84E	E10061	32FFEE	97B84E	E10062	32FFEE	97B84E	E10063	N ¸— a .áîÿ2 N ¸— b .á
0290	32FFEE	97B84E	E10064	32FFEE	37FFF0	97B83E	5000E1	570068	îÿ2 N ¸— d .áîÿ2 ð ÿ7 >
02A0	408010	9FBF40	781F80	07FF9B	B1002F	2088C0	A10010	EF2000	<DLE> €@ @ ¿Ÿ € <US>
02B0	980700	97B8AE	470060	408010	E00410	32000C	97B8AE	470060	. <BELL>˜ ® ¸— ` .G <D
02C0	408010	784010	FB8000	781F80	07FF32	B1002F	200011	470060	<DLE> €@ <DLE> @x .
02D0	408810	37FFEF	2088C0	A00010	FA8000	060000	FA0002	EF2000	<DLE> ˆ@ ïÿ7 À ˆ<SPC
02E0	984700	90400E	E10468	310009	2001C0	781F80	07FF78	B1002F	. G˜ <SO> @□ h <EOT
02F0	07FF31	B3C011	470060	40C810	37FFF4	FA8000	060000	FA0002	1 ÿ<BELL> <DC1> À³ `
0300	EF2000	984700	90400E	E10468	310009	200180	781F80	07FF67	. <SPC>ï . G˜ <SO> @□
0310	B1002F	07FF20	B3C011	470060	40C810	37FFF4	FA8000	060000	/ .± <SPC> ÿ<BELL> <D
0320	2088EF	20FFF0	B7A020	200000	B7A034	200040	B72044	FA0000	ï ˆ<SPC> ð ÿ<SPC> <SP
0330	0203D8	000000	2FFFF0	B7A2A8	07FF1D	2000C0	781F80	07FF4F	Ø <ETX><STX> . .. ð ÿ/
0340	B1002F	208610	781F80	200060	781F80	200010	781F80	07FF87	/ .± <DLE> †<SPC> € <
0350	B1006F	208690	781F80	200060	781F80	200020	781F80	07FF7F	o .± □ †<SPC> € <US>
0360	B1006F	200CC8	273987	ED200E	3AFFFE	ED2010	3AFFFC	200010	o .± È <FF><SPC> ‡ 9'
0370	781F80	07FF35	B1002F	208700	781F80	200010	781F80	200010	€ <US>x 5 ÿ<BELL> / .±
0380	781F80	07FF6D	B1006F	208810	781F80	200050	781F80	200020	€ <US>x m ÿ<BELL> o .:

STATUS: Idle

Software Simulator Debugger Options

Debugger Options

Name	Description	Function Key	Toolbar Icon
Start Debugger	Starts Debugger.	F9	
Run/Pause Debugger	Run/Pause Debugger.	F6	
Stop Debugger	Stop Debugger.	Ctrl + F2	
Step Into	Executes the current program line, then halts. If the executed program line calls another routine, the debugger steps into the routine and halts after executing the first instruction within it.	F7	
Step Over	Executes the current program line, then halts. If the executed program line calls another routine, the debugger will not step into it. The whole routine will be executed and the debugger halts at the first instruction following the call.	F8	
Step Out	Executes all remaining program lines within the subroutine. The debugger halts immediately upon exiting the subroutine. this option is provided with the PIC18 microcontroller family, but not with the PIC16 family.	F8	
Run To Cursor	Executes the program until reaching the cursor position.	Ctrl + F8	
Toggle Breakpoint	Toggle breakpoints option sets new breakpoints or removes those already set at the current cursor position.	F5	🟥

Related topics: Run Menu, Debug Toolbar

CHAPTER 6

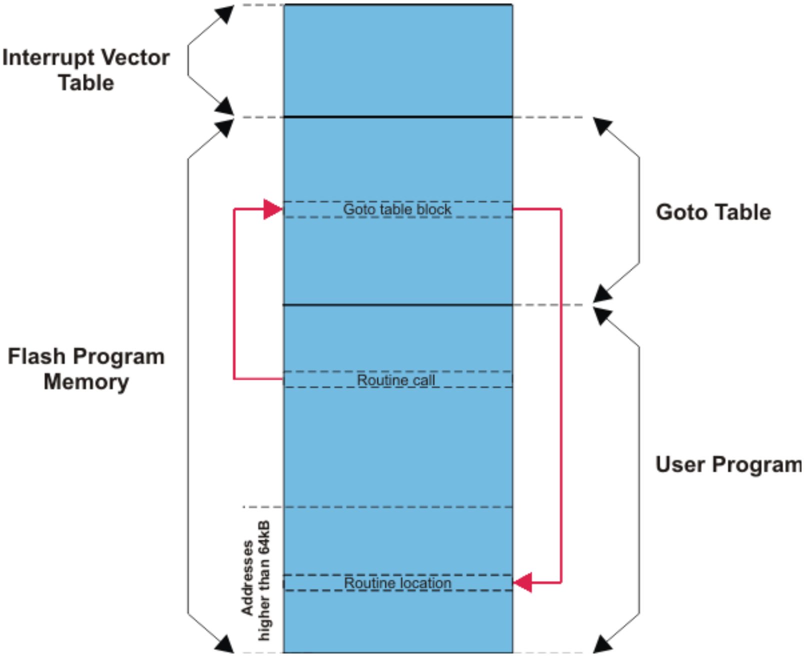
mikroBasic PRO for dsPIC30/33 and PIC24 Specifics

The following topics cover the specifics of mikroBasic PRO for dsPIC30/33 and PIC24 compiler:

- Basic Standard Issues
- Predefined Globals and Constants
- Accessing Individual Bits
- Interrupts
- Linker Directives
- Built-in Routines
- Code Optimization

GOTO Table

If a certain routine is allocated on the address higher than 64kB and can not be accessed directly, a GOTO table is created just after the Interrupt Vector Table to enable this routine call.
GOTO table comprises of addresses of those routines that are allocated on the addresses higher than 64kB.
So, whenever a call is made to a routine which is not directly accessible, it jumps to an assigned GOTO table block which contains address of a desired routine. From there, a GOTO call is generated to that address, and the routine is executed.



See also Linker Directives.

Basic Standard Issues

Divergence from the Basic Standard

- Function recursion is not supported because of no easily-usable stack and limited memory dsPIC Specific

C Language Extensions

mikroBasic PRO for dsPIC30/33 and PIC24 has additional set of keywords that do not belong to the standard Basic language keywords:

```
- code
- data
- rx
- sfr
- at
- sbit
- bit
- iv
```

Related topics: Keywords, dsPIC Specifics

Predefined Globals and Constants

To facilitate dsPIC30/33 and PIC24 programming, the mikroBasic PRO for dsPIC30/33 and PIC24 implements a number of predefined globals and constants.

All dsPIC30/33 and PIC24 SFRs are implicitly declared as global variables of volatile word. These identifiers have an external linkage, and are visible in the entire project. When creating a project, the mikroBasic PRO for dsPIC30/33 and PIC24 will include an appropriate (*.mbas) file from defs folder, containing declarations of available SFRs and constants (such as PORTB, ADPCFG, etc). All identifiers are in upper case, identical to nomenclature in the Microchip datasheets.

For a complete set of predefined globals and constants, look for “Defs” in the mikroBasic PRO for dsPIC30/33 and PIC24 installation folder, or probe the Code Assistant for specific letters (Ctrl+Space in the Code Editor).

Predefined project level defines

mikroBasic PRO for dsPIC30/33 and PIC24 provides several predefined project level defines that you can use in your project :

First one is equal to the name of selected device for the project. For example:

```
#IFDEF 30F4013
...
#ENDIF
```

Other predefined project level defines are:

```
#IFDEF P30}...#ENDIF
#IFDEF P33}...#ENDIF
#IFDEF P24}...#ENDIF
#IFDEF MIKRO_ICD}...#ENDIF
```

Related topics: Project Level Defines

Accessing Individual Bits

The mikroBasic PRO for dsPIC30/33 and PIC24 allows you to access individual bits of 16-bit variables. It also supports sbit and bit data types.

Lets use the Zero bit as an example. This bit is defined in the definition file of the particular MCU as:

```
const Z = 1
dim   Z_bit as sbit at SR.B1
```

To access this bit in your code by its name, you can write something like this:

```
' Clear Zero Bit
SR.Z = 0
```

In this way, if Zero bit changes its position in the register, you are sure that the appropriate bit will be affected.
But, if Zero bit is not located in the designated register, you may get errors.

Another way of accesing bits is by using the direct member selector (`.`) with a variable, followed by a primary expression. Primary expression can be variable, constant, function call or an expression enclosed by parentheses. For individual bit access there are predefined global constants `B0, B1, … , B15`, or `0, 1, … 15`, with `15` being the most significant bit:

```
' predefined globals as bit designators
' Clear bit 0 in STATUS register
SR.B0 = 0

' literal constant as bit designator
' Set bit 5 in STATUS register
SR.F5 = 1

' expression as bit designator
' Set bit 6 in STATUS register
i = 5
SR.(i+1) = 1
```

In this way, if the target bit changes its position in the register, you cannot be sure that you are invoking the appropriate bit.

This kind of selective access is an intrinsic feature of mikroBasic PRO for dsPIC30/33 and PIC24 and can be used anywhere in the code. Identifiers `B0-B15` are not case sensitive and have a specific namespace.

You may override them with your own members `B0-B15` within any given structure.

When using literal constants as bit designators instead of predefined ones, make sure not to exceed the appropriate type size.

Also, you can access the desired bit by using its alias name, in this case `Z_bit`:

```
' Set Zero Bit
Z_bit = 1
```

In this way, if the Zero bit changes its register or position in the register, you are sure that the appropriate bit will be affected.

See Predefined Globals and Constants for more information on register/bit names.

sbit type

The mikroBasic PRO for dsPIC30/33 and PIC24 compiler has `sbit` data type which provides access to registers, SFRs, variables, etc.
You can declare a `sbit` varible in a unit in such way that it points to a specific bit in SFR register:

```
module MyModule

dim Abit as sbit sfr external ' Abit is precisely defined in some external file, for example in the main program unit
...
implements
....
end.
```

In the main program you have to specify to which register this sbit points to, for example:

```
program MyProgram
...
dim Abit as sbit at PORTB.0 ' this is where Abit is fully defined
...
main:
...
end.
```

In this way the variable `Abit` will actually point to PORTB.0. Please note that we used the keyword `sfr` for declaration of `Abit`, because we are pointing it to PORTB which is defined as a `sfr` variable.
In case we want to declare a bit over a variable which is not defined as `sfr`, then the keyword `sfr` is not necessary, for example:

```
module Mymodule

dim AnotherBit as sbit external ' Abit is precisely defined in some external file, for example in the main program unit
...
implements
...
end.


program MyProgram
...
dim MyVar as byte
dim Abit as sbit at MyVar.0 ' this is where Abit is fully defined
...
main:
...
end.
```

at keyword

You can use the keyword “at” to make an alias to a variable, for example, you can write a library without using register names, and later in the main program to define those registers, for example:

```
module MyModule

dim PORTAlias as byte external  ' here in the library we can use its symbolic name
...
implements
...
end.


program MyProgram
...
dim PORTAlias byte as at PORTB  ' this is where PORTAlias is fully defined
...
main:
...
end.
```

Note: Bear in mind that when using `at` operator in your code over a variable defined through a `external` modifier, appropriate memory specifer must be appended also.

bit type

The mikroBasic PRO for dsPIC30/33 and PIC24 compiler provides a `bit` data type that may be used for variable declarations. It can not be used for argument lists, and function-return values.

```
dim bf as bit   ' bit variable
```

There are no pointers to bit variables:

~~`dim ptr as ^bit`~~ `' invalid`

An array of type bit is not valid:

~~`dim arr as array[5] of bit`~~ `' invalid`

Note:

- Bit variables can not be initialized.
- Bit variables can not be members of structures and unions.
- Bit variables do not have addresses, therefore unary operator `@` (address of) is not applicable to these variables.

Related topics: Predefined globals and constants, External modifier

Interrupts

The dsPIC30/33 and PIC24 interrupt controller module reduces numerous peripheral interrupt request signals to a single interrupt request signal to the dsPIC30/33 and PIC24 CPU and has the following features:

- Up to 8 processor exceptions and software traps
- 7 user-selectable priority levels
- Interrupt Vector Table (IVT) with up to 62 vectors (dsPIC30) or up to 118 vectors (dsPIC33 and PIC24)
- A unique vector for each interrupt or exception source
- Fixed priority within a specified user priority level
- Alternate Interrupt Vector Table (AIVT) for debug support

ISRs are organized in IVT. ISR is defined as a standard function but with the iv directive afterwards which connects the function with specific interrupt vector. For example `iv IVT_ADDR_T1INTERRUPT` is IVT address of Timer1 interrupt source of the dsPIC 30F3014 MCU. For more information on IVT refer to the dsPIC30/33 and PIC24 Family Reference Manual.

Function Calls from Interrupt

Calling functions from within the interrupt routine is possible. The compiler takes care about the registers being used, both in “interrupt” and in “main” thread, and performs “smart” context-switching between two of them, saving only the registers that have been used in both threads. It is not recommended to use a function call from interrupt. In case of doing that take care of stack depth.

Use the DisableContextSaving to instruct the compiler not to automatically perform context-switching. This means that no register will be saved/restored by the compiler on entrance/exit from interrupt service routine.
This enables the user to manually write code for saving registers upon entrance and to restore them before exit from interrupt.

Interrupt Handling

For the sake of interrupt handling convenience, new keyword, `iv`, is introduced. It is used to declare Interrupt Vector Table (IVT) address for a defined interrupt routine:

```
sub procedure int1() iv IVT_ADDR_U1RXINTERRUPT
asm
  nop
end asm
end sub
```

Now it is possible to explicitly declare interrupt routine address:

```
sub procedure int1() org 0x600 iv IVT_ADDR_U1RXINTERRUPT
asm
  nop
end asm
end sub
```

For the sake of backward compatibility, user may write also:

```
sub procedure int1() org IVT_ADDR_U1RXINTERRUPT
asm
  nop
end asm
end sub
```

which is equivalent to:

```
sub procedure int1() iv IVT_ADDR_U1RXINTERRUPT
asm
  nop
end asm
end sub
```

Is is recommended that interrupts are handled in this way for the sake of better readability of the user projects.

Interrupt Example

Here is a simple example of handling the interrupts from `Timer1` (if no other interrupts are allowed):

```
//-------------- Interrupt routine
sub procedure Timer1Int iv IVT_ADDR_T1INTERRUPT
  '** it is necessary to clear manually the interrupt flag:
  IFS0 = IFS0 and $FFF7    ' Clear TMR1IF

  '** user code starts here
  LATB = not PORTB         ' Invert PORTB
  '** user code ends here
end sub
```

Linker Directives

mikroBasic PRO for dsPIC30/33 and PIC24 uses internal algorithm to distribute objects within memory. If you need to have a variable or routine at the specific predefined address, use the linker directives `absolute` and `org`.

Directive absolute

Directive `absolute` specifies the starting address in RAM for a variable. If the variable is multi-byte, higher bytes will be stored at the consecutive locations.

Directive `absolute` is appended to declaration of a variable:

```
' Variable x will occupy 1 word (16 bits) at address 0x32
dim x as word absolute 0x32

' Variable y will occupy 2 words at addresses 0x34 and 0x36
dim y as longint absolute 0x34
```

Be careful when using `absolute` directive, as you may overlap two variables by accident. For example:

```
dim  i as word absolute 0x42
' Variable i will occupy 1 word at address 0x42;

dim jj as longint absolute 0x40
' Variable will occupy 2 words at 0x40 and 0x42; thus,
' changing i changes jj at the same time and vice versa
```

Directive orgall

Directive `org` specifies the starting address of a constant or a routine in ROM. It is appended to the constant or a routine declaration.

To place a constant array in Flash memory, write the following:

```
' Constant array MONTHS will be placed starting from the address 0x800
const MONTHS as byte[12] = (31,28,31,30,31,30,31,31,30,31,30,31) org 0x800
```

If you want to place simple type constant into Flash memory, instead of following declaration:

```
const SimpleConstant as byte = 0xAA org 0x2000
```

use an array consisting of single element:

```
const SimpleConstant as byte[1] = (0xAA) org 0x800
```

In first case, compiler will recognize your attempt, but in order to save Flash space, and boost performance, it will automatically replace all instances of this constant in code with it’s literal value.
In the second case your constant will be placed in Flash in the exact location specified.

To place a routine on a specific address in Flash memory you should write the following:

```
sub procedure proc(dim par as word) org 0x200
' Procedure will start at the address 0x200;
...
end sub
```

`org` directive can be used with `main` routine too. For example:

```
program Led_Blinking

main: org 0x800                ' main procedure starts at 0x800
...
end
```

Directive orgall

Use the `orgall` directive to specify the address above which all routines and constants will be placed. Example:

```
main:
  orgall(0x200)  ' All the routines, constants in main program will be above the address 
0x200
  ...
end.
```

Built-in Routines

The mikroBasic PRO for dsPIC30/33 and PIC24 compiler provides a set of useful built-in utility functions. Built-in functions do not have any special requirements. You can use them in any part of your project.

The `Delay_us` and `Delay_ms` routines are implemented as “inline”; i.e. code is generated in the place of a call, so the call doesn’t count against the nested call limit.

The `Vdelay_ms, Vdelay_advanced_ms, Delay_Cyc, Delay_Cyc_Long, Get_Fosc_kHz and Get_Fosc_Per_Cyc` are actual Basic routines. Their sources can be found in the `__Lib_Delays.mbas` file located in the `Uses` folder of the compiler.

- Lo
- Hi
- Higher
- Highest
- LoWord
- HiWord

- Inc
- Dec

- Chr
- Ord

- SetBit
- ClearBit
- TestBit

- Delay_us
- Delay_ms
- Vdelay_ms
- Vdelay_Advanced_ms
- Delay_Cyc
- Delay_Cyc_Long

- Clock_kHz
- Clock_MHz
- Get_Fosc_kHz
- Get_Fosc_Per_Cyc

- Reset
- ClrWdt

- DisableContextSaving

- SetFuncCall
- SetOrg

- DoGetDateTime
- DoGetVersion

Lo

Prototype	`sub function Lo(dim number as longint) as byte`
Description	Function returns the lowest byte of `number`. Function does not interpret bit patterns of `number` – it merely returns 8 bits as found in register. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Parameters	`number`: input value
Returns	Lowest 8 bits (byte) of `number`, bits `7..0`.
Requires	Arguments must be variable of scalar type (i.e. Arithmetic Types and Pointers).

Example	`d = 0x12345678`
	`tmp = Lo(d)   ' Equals 0x78`
	
	`Lo(d) = 0xAA ' d equals 0x123456AA`

Notes	None.

Hi

Prototype	`sub function Hi(dim number as longint) as byte`
Description	Function returns next to the lowest byte of `number`. Function does not interpret bit patterns of `number` – it merely returns 8 bits as found in register. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Parameters	`number`: input value
Returns	Returns next to the lowest byte of `number`, bits 8..15.
Requires	Arguments must be variable of scalar type (i.e. Arithmetic Types and Pointers).

Example	`d = 0x12345678`
	`tmp = Hi(d)   ' Equals 0x56`
	
	`Hi(d) = 0xAA ' d equals 0x1234AA78`

Notes	None.

Higher

Prototype	`sub function Higher(dim number as longint) as byte`
Description	Function returns next to the highest byte of `number`. Function does not interpret bit patterns of `number` – it merely returns 8 bits as found in register. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Parameters	`number`: input value
Returns	Returns next to the highest byte of `number`, bits 16..23.
Requires	Arguments must be variable of scalar type (i.e. Arithmetic Types and Pointers).

Example	`d = 0x12345678`
	`tmp = Higher(d)   ' Equals 0x34`
	
	`Higher(d) = 0xAA ' d equals 0x12AA5678`

Notes	None.

Highest

Prototype	`sub function Highest(dim number as longint) as byte`
Description	Function returns the highest byte of `number`. Function does not interpret bit patterns of `number` – it merely returns 8 bits as found in register. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Parameters	`number`: input value
Returns	Returns the highest byte of `number`, bits 24..31.
Requires	Arguments must be variable of scalar type (i.e. Arithmetic Types and Pointers).

Example	`d = 0x12345678`
	`tmp = Highest(d)   ' Equals 0x12`
	
	`Highest(d) = 0xAA ' d equals 0xAA345678`

Notes	None.

LoWord

Prototype	`sub function LoWord(dim val as longint) as word`
Description	The function returns low word of `val`. The function does not interpret bit patterns of `val` – it merely returns 16 bits as found in register. Parameters: - `val`: input value
Parameters	`number`
Returns	Low word of `val`, bits `15..0`.
Requires	Nothing.
Example	see below
Notes	None.

```
d = 0x12345678
tmp = LoWord(d)   ' Equals 0x5678

LoWord(d) = 0xAAAA ' d equals 0x1234AAAA
```

HiWord

Prototype	`sub function HiWord(dim val as longint) as word`
Description	The function returns high word of `val`. The function does not interpret bit patterns of `val` – it merely returns 16 bits as found in register. Parameters: - `val`: input value
Parameters	`number`
Returns	High word of `val`, bits `31..16`.
Requires	Nothing.
Example	see below
Notes	None.

```
d = 0x12345678
tmp = HiWord(d)   ' Equals 0x1234

HiWord(d) = 0xAAAA ' d equals 0xAAAA5678
```

Inc

Prototype	`sub procedure Inc(dim byref par as longint)`
Description	Increases parameter `par` by 1.
Parameters	- `par`: value which will be incremented by 1
Returns	Nothing.
Requires	Nothing.
Example	`p = 4` `Inc(p)   ' p is now 5`
Notes	None.

Dec

Prototype	`sub procedure Dec(dim byref par as longint)`
Description	Decreases parameter `par` by 1.
Parameters	- `par`: value which will be decremented by 1
Returns	Nothing.
Requires	Nothing.
Example	`p = 4` `Dec(p)   ' p is now 3`
Notes	None.

Chr

Prototype	`sub function Chr(dim code_ as byte) as char`
Description	Function returns a character associated with the specified character `code_`. Numbers from 0 to 31 are the standard non-printable ASCII codes. This is an “inline” routine; the code is generated in the place of the call.
Parameters	- `code`: input character
Returns	Returns a character associated with the specified character `code_`.
Requires	Nothing.
Example	`c = Chr(10)   ' returns the linefeed character`
Notes	None.

Ord

Prototype	`sub function Ord(dim character as char) as byte`
Description	Function returns ASCII code of the `character`. This is an “inline” routine; the code is generated in the place of the call.
Parameters	- `character`: input character
Returns	ASCII code of the `character`.
Requires	Nothing.
Example	`c = Ord(“A”)   ' returns 65`
Notes	None.

SetBit

Prototype	`sub procedure SetBit(dim byref register_ as word, dim rbit as byte)`
Description	Function sets the bit `rbit` of `register_`. Parameter `rbit` needs to be a variable or literal with value 0..15. For more information on register identifiers see Predefined Globals and Constants . This is an “inline” routine; the code is generated in the place of the call.
Parameters	- `register_`: desired register - `rbit`: desired bit
Returns	Nothing.
Requires	Nothing.
Example	`SetBit(PORTB, 2)   ' Set RB2`
Notes	None.

ClearBit

Prototype	`sub procedure ClearBit(dim byref register_ as word, dim rbit as byte)`
Description	Function clears the bit `rbit` of `register`. Parameter `rbit` needs to be a variable or literal with value 0..7. See Predefined globals and constants for more information on register identifiers. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Parameters	- `register_`: desired register - `rbit`: desired bit
Returns	Nothing.
Requires	Nothing.
Example	`ClearBit(PORTC, 7)   ' Clear RC7`
Notes	None.

TestBit

Prototype	`sub function TestBit(dim register_, rbit as byte) as byte`
Description	Function tests if the bit `rbit` of `register` is set. If set, function returns 1, otherwise returns 0. Parameter `rbit` needs to be a variable or literal with value 0..7. See Predefined globals and constants for more information on register identifiers. This is an “inline” routine; code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Parameters	- `register_`: desired register - `rbit`: desired bit
Returns	If the bit is set, returns 1, otherwise returns 0.
Requires	Nothing.
Example	`flag = TestBit(PORTE, 2)   ' 1 if RE2 is set, otherwise 0`
Notes	None.

Delay_us

Prototype	`sub procedure Delay_us(const time_in_us as longword)`
Description	Creates a software delay in duration of `Time_In_us` microseconds. This is an “inline” routine; the code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Parameters	`time_in_us`: delay time in microseconds. Valid values: constant values, range of applicable constants depends on the oscillator frequency
Returns	Nothing.
Requires	Nothing.
Example	`Delay_us(1000)   ' One millisecond pause`
Notes	None.

Delay_ms

Prototype	`sub procedure Delay_ms(const time_in_ms as longword)`
Description	Creates a software delay in duration of `Time_In_ms` milliseconds. This is an “inline” routine; the code is generated in the place of the call, so the call doesn’t count against the nested call limit.
Parameters	`Time_in_ms`: delay time in milliseconds. Valid values: constant values, range of applicable constants depends on the oscillator frequency
Returns	Nothing.
Requires	Nothing.
Example	`Delay_ms(1000)   ' One second pause`
Notes	For generating delays with variable as input parameter use the Vdelay_ms routine.

Vdelay_ms

Prototype	`sub procedure Vdelay_ms(dim time_in_ms as word)`
Description	Creates a software delay in duration of `Time_ms` milliseconds. Generated delay is not as precise as the delay created by Delay_ms.
Parameters	`Time_ms`: delay time in milliseconds
Returns	Nothing.
Requires	Nothing.

Example	`pause = 1000`
	`` ` ... ``
	`Vdelay_ms(pause)   ' ~ one second pause`

Notes	None.

VDelay_advanced_ms

Prototype	`sub procedure VDelay_advanced_ms(dim time_ms, Current_Fosc_kHz as word)`
Description	Creates a software delay in duration of `time_in_ms` milliseconds (a variable), for a given oscillator frequency. Generated delay is not as precise as the delay created by Delay_ms. Note that `Vdelay_ms` is library function rather than a built-in routine; it is presented in this topic for the sake of convenience.
Parameters	- `time_ms`: delay time in milliseconds - `Current_Fosc_kHz`: frequency in kHz
Returns	Nothing.
Requires	Nothing.

Example	`pause = 1000`
	`fosc = 10000`
	
	`VDelay_advanced_ms(pause, fosc)   ' Generates approximately one second pause, for a oscillator frequency of 10 MHz`

Notes	None.

Delay_Cyc

Prototype	`sub procedure Delay_Cyc(dim x, y as word)`
Description	Creates a delay based on MCU clock. Delay lasts for `x*16384 + y` MCU clock cycles.
Parameters	`x`: NumberOfCycles divided by 16384 `y`: remainder of the NumberOfCycles/16384 division
Returns	Nothing.
Requires	Nothing.
Example	`Delay_Cyc(1, 10)   ' 1x16384 + 10 = 16394 cycles pause`
Notes	`Delay_Cyc` is a library function rather than a built-in routine; it is presented in this topic for the sake of convenience.

Delay_Cyc_Long

Prototype	`sub procedure Delay_Cyc_Long(dim CycNo as word)`
Description	Creates a delay based on MCU clock. Delay lasts for `CycNo` MCU clock cycles.
Parameters	`- CycNo`: number of MCU cycles
Returns	Nothing.
Requires	Nothing.
Example	`Delay_Cyc_Long(16384)   ' 16384 cycles pause`
Notes	`Delay_Cyc_Long` is a library function rather than a built-in routine; it is presented in this topic for the sake of convenience.

Clock_kHz

Prototype	`sub function Clock_kHz() as longint`
Description	Returns device clock in kHz, rounded to the nearest integer. This is an “inline” routine; the code is generated in the place of the call.
Parameters	None.
Returns	Device clock in kHz, rounded to the nearest integer.
Requires	Nothing.
Example	`clk = Clock_kHz()`
Notes	None.

Clock_MHz

Prototype	`sub function Clock_MHz() as word`
Description	Returns device clock in MHz, rounded to the nearest integer. This is an “inline” routine; the code is generated in the place of the call.
Parameters	None.
Returns	Device clock in MHz, rounded to the nearest integer.
Requires	Nothing.
Example	`clk = Clock_MHz()`
Notes	None.

Get_Fosc_kHz

Prototype	`sub function Get_Fosc_kHz() as longint`
Description	Function returns device clock in kHz, rounded to the nearest integer.
Parameters	None.
Returns	Device clock in kHz.
Requires	Nothing.
Example	`clk = Get_Fosc_kHz()`
Notes	`Get_Fosc_kHz` is a library function rather than a built-in routine; it is presented in this topic for the sake of convenience.

Get_Fosc_Per_Cyc

Prototype	`sub function Get_Fosc_Per_Cyc() as word`
Description	Function returns device’s clock per cycle, rounded to the nearest integer. Note that `Get_Fosc_Per_Cyc` is library function rather than a built-in routine; it is presented in this topic for the sake of convenience.
Parameters	None.
Returns	Device’s clock per cycle, rounded to the nearest integer.
Requires	Nothing.

Example	`dim clk_per_cyc as word`
	`...`
	`clk_per_cyc = Get_Fosc_Per_Cyc()`

Notes	None.

Reset

Prototype	`sub procedure Reset()`
Description	This procedure is equal to assembler instruction **reset**.
Parameters	None.
Returns	Nothing.
Requires	Nothing.
Example	`Reset()  ' Resets the MCU`
Notes	None.

ClrWdt

Prototype	`sub procedure ClrWdt()`
Description	This procedure is equal to assembler instruction `clrwdt`.
Parameters	None.
Returns	Nothing.
Requires	Nothing.
Example	`ClrWdt()  ' Clears WDT`
Notes	None.

DisableContextSaving()

Prototype	`sub procedure DisableContextSaving()`
Description	Use the `DisableContextSaving()` to instruct the compiler not to automatically perform context-switching. This means that no register will be saved/restored by the compiler on entrance/exit from interrupt service routine. This enables the user to manually write code for saving registers upon entrance and to restore them before exit from interrupt.
Parameters	None.
Returns	Nothing.
Requires	This routine must be called from main.
Example	`DisableContextSaving()  ' instruct the compiler not to automatically perform context-switching`
Notes	None.

SetFuncCall

Prototype	`sub procedure SetFuncCall(dim FuncName as string)`
Description	If the linker encounters an indirect function call (by a pointer to function), it assumes that any routine whose address was taken anywhere in the program can be called at that point if it's prototype matches the pointer declaration. Use the SetFuncCall directive within routine body to instruct the linker which routines can be called indirectly from that routine: `SetFunCCall (called_func[, ,...])` Routines specified in the `SetFunCCall` argument list will be linked if the routine containing `SetFunCCall` directive is called in the code no matter whether any of them was explicitly called or not. Thus, placing `SetFuncCall` directive in main will make compiler link specified routines always.
Parameters	- `FuncName`: function name
Returns	Nothing.
Requires	Nothing.

Example

```
sub procedure first(p, q as byte)
...
   SetFuncCall(second)  ‘ let linker know that we will call the routine ‘second’
...
end sub
```

Notes	The `SetFuncCall` directive can help the linker to optimize function frame allocation in the compiled stack.

SetOrg

Prototype	`sub procedure SetOrg(dim RoutineName as string, dim address as longint)`
Description	Use the `SetOrg()` routine to specify the starting address of a routine in ROM.
Parameters	- `RoutineName`: routine name - `address`: starting address
Returns	Nothing.
Requires	This routine must be called from main.
Example	`SetOrg(UART1_Write, 0x1234)`
Notes	None.

DoGetDateTime

Prototype	`sub function DoGetDateTime() as string`
Description	Use the `DoGetDateTime()` to get date and time of compilation as string in your code.
Parameters	None.
Returns	String with date and time when this routine is compiled.
Requires	Nothing.
Example	`str = DoGetDateTime()`
Notes	None.

DoGetVersion

Prototype	`sub function DoGetVersion() as string`
Description	Use the `DoGetVersion()` to get the current version of compiler.
Parameters	None.
Returns	String with current compiler version.
Requires	Nothing.
Example	`str = DoGetVersion()  ' for example, str will take the value of '8.2.1.6'`
Notes	None.

Code Optimization

Optimizer has been added to extend the compiler usability, cut down the amount of code generated and speed-up its execution. The main features are:

Constant folding

All expressions that can be evaluated in the compile time (i.e. are constant) are being replaced by their results. (3 + 5 -> 8);

Constant propagation

When a constant value is being assigned to a certain variable, the compiler recognizes this and replaces the use of the variable by constant in the code that follows, as long as the value of a variable remains unchanged.

Copy propagation

The compiler recognizes that two variables have the same value and eliminates one of them further in the code.

Value numbering

The compiler “recognizes” if two expressions yield the same result and can therefore eliminate the entire computation for one of them.

"Dead code" ellimination

The code snippets that are not being used elsewhere in the programme do not affect the final result of the application. They are automatically removed.

Stack allocation

Temporary registers (“Stacks”) are being used more rationally, allowing VERY complex expressions to be evaluated with a minimum stack consumption.

Local vars optimization

No local variables are being used if their result does not affect some of the global or volatile variables.

Better code generation and local optimization

Code generation is more consistent and more attention is payed to implement specific solutions for the code “building bricks” that further reduce output code size.

Related topics: SSA Optimization, dsPIC specifics, mikroBasic PRO for dsPIC30/33 and PIC24 specifics, Memory type specifiers

Single Static Assignment Optimization

Introduction

In compiler design, static single assignment form (often abbreviated as SSA form or SSA) is an intermediate representation (IR) in which every variable is assigned exactly once.

An SSA-based compiler modifies the program representation so that every time a variable is assigned in the original program, a new version of the variable is created.
A new version of the variable is distinguished (renamed) by subscripting the variable name with its version number or an index, so that every definition of each variable in a program becomes unique.
At a joining point of the control flow graph where two or more different definitions of a variable meet, a hypothetical function called a phi-function is inserted so that these multiple definitions are merged.

In mikroBasic PRO for dsPIC, SSA's main goal is in allocating local variables into the RX space (instead onto the frame). To do that, SSA has to make an alias and data flow analysis of the Control Flow Graph.

Besides these savings, there are a number of compiler optimization algorithms enhanced by the use of SSA, like:

- Constant Propagation
- Dead Code Elimination
- Global Value Numbering
- Register Allocation

Changes that SSA brings is also in the way in which routine parameters are passed. When the SSA is enabled, parameters are passed through a part of the RX space which is reserved exclusively for this purpose (W10-W13 for dsPIC).
Allocating local variables and parameters in RX space has its true meaning for those architectures with hardware frame.

Enabling SSA optimization in compiler is done by checki ☑ Enable SSA optimization box from the Output Settings Menu.

Lets consider a trivial case:

```
program Example

sub procedure SSA_Test(dim y as integer, dim k as integer)
  if (y+k) then
    asm
      nop
    end asm
  end if
end sub

main:
  SSA_Test(5,5)
end.
```

With SSA enabled, sub procedure `SSA_Test` this example is consisted of 3 asm instructions:

```
;Example.mbas,29 ::                      if (y+k) then
0x0100          0x45000B          ADD         W10, W11, W0
```

```
0x0102         0x320001          BRA Z        L__SSA_Test2
L__SSA_Test6:
;Example.mbas,31 ::                   nop
0x0104         0x000000          NOP
```

Without SSA enabled, sub procedure `SSA_Test` this example is consisted of 5 asm instructions :

```
;Example.mbas,29 ::                   if (y+k) then
0x0102         0x97B8CE          MOV         [W14-8], W1
0x0104         0x57006A          SUB         W14, #10, W0
0x0106         0x408010          ADD         W1, [W0], W0
0x0108         0x320001          BRA Z         L__SSA_Test2
L__SSA_Test6:
;Example.mbas,31 ::                   nop
0x010A         0x000000          NOP
```

Proper Coding Recommendations

To get the maximum out of the SSA, user should regard the following rules during the coding process:

- Routines should not contain too many parameters (not more than 4 words).
- Don’t change the value of the parameter in the function body (it is better to use a new local variable).
- If the `function1` parameters are passed as `function2` parameters, then parameter order should remain the same:

```
sub procedure f2(dim a as integer, dim b as integer)

end sub

sub procedure f1(dim x as integer, dim y as integer)
  ' routine call
f2(x,y)  ' x->a and y->b (1 to 1 and 2 to 2) is far more efficient than:
f2(y,x)  ' y->a and x->b (1 to 2 and 2 to 1)
end sub
```

- Large amount of nested loops and complex structures as its members should be avoided.
- When writing a code in assembly, keep in mind that there are registers reserved exclusively for routine parameters.
- Using `goto` and `label` statements in nested loops should be avoided.
- Obtaining address of the local variable with the global pointer and using it to alter the variable’s address should be avoided.

Note:

- `mcl` files compiled with or without SSA enabled are fully compatible and can be used and mixed without any restrictions, except pointers to functions.
- Functions, functions declarations and pointers that may point to these functions must be compiled with the same option, either SSA enabled or disabled. If this is not the case, compiler will report an error.

Asm code and SSA optimization

If converting code from an earlier version of the compiler, which consists of mixed asm code with the Basic code, keep in mind that the generated code can substantially differ when SSA optimization option is enabled or disabled.
This is due to the fact that SSA optimization uses certain working registers to store routine parameters (W10-W13), rather than storing them onto the function frame.

Because of this, user must be very careful when writing asm code as existing values in the working registers used by SSA optimization can be overwritten.
To avoid this, it is recommended that user includes desired asm code in a separate routine.

Debugging Notes

SSA also influences the code debugging in such a way that the local variables will be available in the Watch Window only in those parts of the procedure where they have useful value (eg. on entering the procedure, variable isn't available until its definition).
Variables can be allocated in one part of the procedure in register W4, and in another part of the procedure in register W2, if the optimizer estimates that it is better that way. That means that the local variable has no static address.

Warning Messages Enhancement

Besides the smaller code, SSA also deals with the intensive code analysis, which in turn has the consequence in enhancing the warning messages.
For example, compiler will warn the user that the uninitialized variable is used:

```
sub procedure SSA_Test()
dim y as char

  if (y) then     ' Variable y might not have been initialized
    asm
      nop
    end asm
  end if

end sub

main:
  SSA_Test()
end.
```

Related topics: Code Optimization, dsPIC Specifics, mikroBasic PRO for dsPIC30/33 and PIC24 specifics, Memory type specifiers

Common Object File Format (COFF)

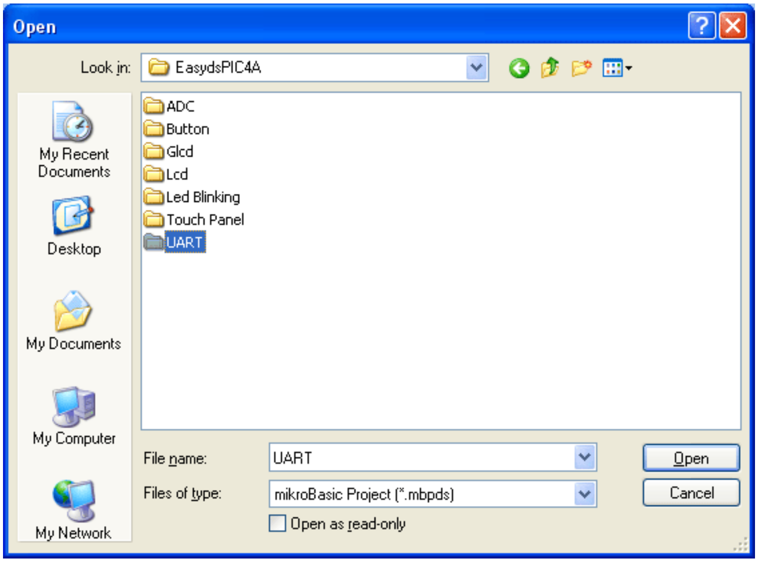
COFF File Format

The Common Object File Format (COFF) is a specific file format suitable for code debugging.
The COFF incorporates symbolic procedure, function, variable and constant names information; line number information, breakpoints settings, code highlighter and all the necessary information for effective and fast debugging.

By using COFF, it is possible to import and debug code generated by mikroElektronika compilers under Microchip's MPLAB®.

COFF File Generation

1. Start mikroBasic PRO for dsPIC30/33 and PIC24 Help and open the desired project. For example, UART project for EasydsPIC4A board and dsPIC30F4013 will be opened:

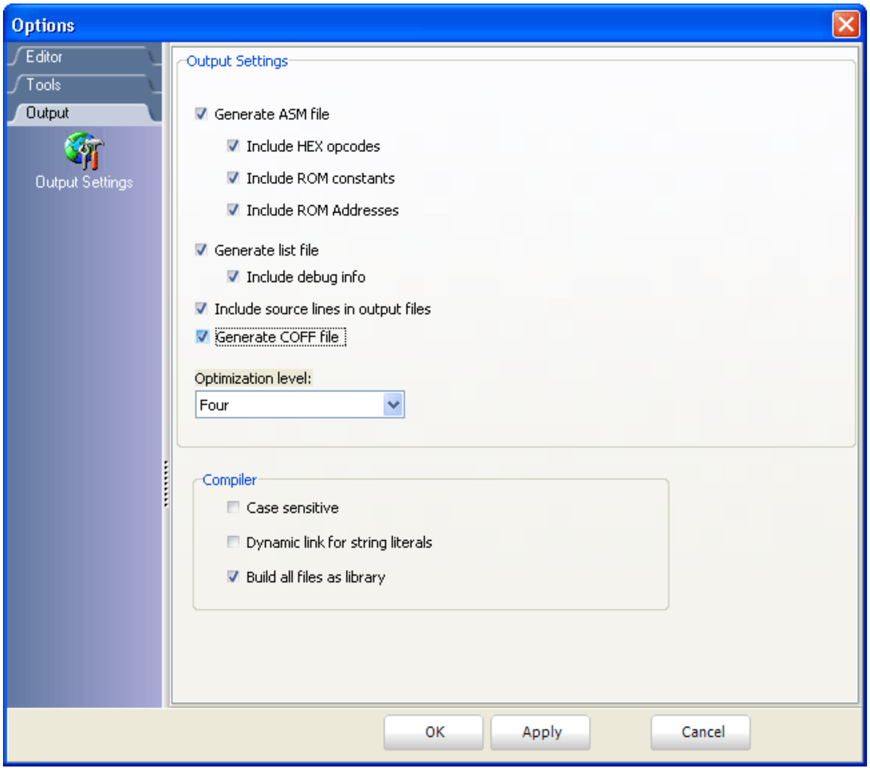


Look in: EasydsPIC4A

File name: UART

Files of type: mikroBasic Project (*.mbpds)

2. When the project is opened, go to **Tools › Options › Output settings**, and check the "**Generate COFF file**" option, and click the OK button:



Output Settings

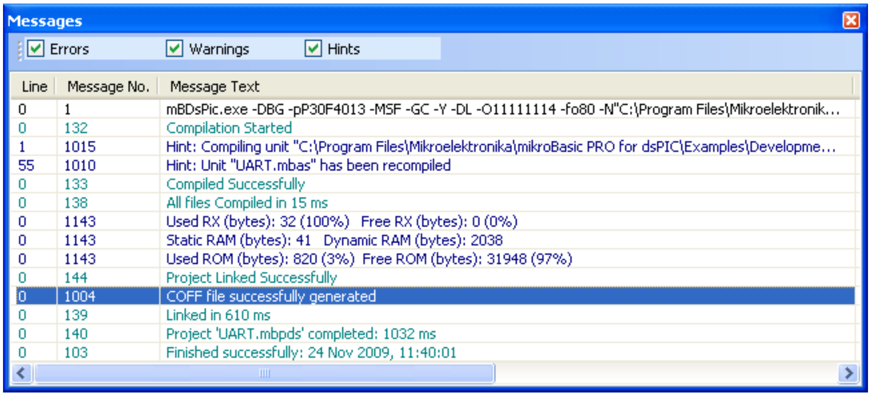
☑ Generate ASM file
  ☑ Include HEX opcodes
  ☑ Include ROM constants
  ☑ Include ROM Addresses
☑ Generate list file
  ☑ Include debug info
☑ Include source lines in output files
☑ Generate COFF file

Optimization level: Four

Compiler

☐ Case sensitive
☐ Dynamic link for string literals
☑ Build all files as library

3. Now, compile the project. In the messages window, appropriate message on COFF file generation should appear:



Line	Message No.	Message Text
0	1	mBDsPic.exe -DBG -pP30F4013 -MSF -GC -Y -DL -O11111114 -fo80 -N"C:\Program Files\Mikroelektronik...
0	132	Compilation Started
1	1015	Hint: Compiling unit "C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Examples\Developme...
55	1010	Hint: Unit "UART.mbas" has been recompiled
0	133	Compiled Successfully
0	138	All files Compiled in 15 ms
0	1143	Used RX (bytes): 32 (100%) Free RX (bytes): 0 (0%)
0	1143	Static RAM (bytes): 41 Dynamic RAM (bytes): 2038
0	1143	Used ROM (bytes): 820 (3%) Free ROM (bytes): 31948 (97%)
0	144	Project Linked Successfully
0	1004	COFF file successfully generated
0	139	Linked in 610 ms
0	140	Project 'UART.mbpds' completed: 1032 ms
0	103	Finished successfully: 24 Nov 2009, 11:40:01

4. Generated COFF file will be created in the project folder, with the `.cof` extension.

Related topics: Using MPLAB® ICD 2 Debugger, Using MPLAB® Simulator

CHAPTER 7

dsPIC30/33 and PIC24 Specifics

In order to get the most from the mikroBasic PRO for dsPIC30/33 and PIC24 compiler, the user should be familiar with certain aspects of dsPIC30/33 and PIC24 MCU. This knowledge is not essential, but it can provide a better understanding of the dsPIC30/33 and PIC24's capabilities and limitations, and their impact on the code writing as well.

Types Efficiency

First of all, the user should know that dsPIC30/33 and PIC24's ALU, which performs arithmetic operations, is optimized for working with 16-bit types. Although mikroBasic PRO for dsPIC30/33 and PIC24 is capable of handling types like `byte, char` or `short`, dsPIC30/33 and PIC24 will generate a better code for 16-bit types `word` and `integer` type so use `byte, char` and `short` only in places where you can significantlly save RAM (e.g. for arrays `dim a as byte[30]`).

Nested Calls Limitations

There are no Nested Calls Limitations, except by RAM size. A Nested call represents a function call within the function body, either to itself (recursive calls) or to another function.

Recursive calls, as a form of cross-calling, are supported by mikroBasic PRO for dsPIC30/33 and PIC24, but they should be used very carefully due to dsPIC30/33 and PIC24 stack and memory limitations. Also calling functions from interrupt is allowed. Calling function from both interrupt and main thread is allowed. Be carefull because this programming technique may cause unpredictable results if common resources are used in both main and interrupt.

Limits of Indirect Approach Through PSV

Constant aggregates are stored in Flash and are accessible through PSV. mikroBasic PRO for dsPIC30/33 and PIC24 can allocate more than 32KByte of constants. See `near` and `far` memory specifiers.

Limits of Pointer to Function

Currently pointer to functions are 16-bit variables. For functions which address exceeds 16 bit limit, the compiler uses handle (16-bit pointer on GOTO). A handle usage is automatic compiler process so there is no need for the user to intervene.

Variable, constant and routine alignment

Simple type variables whose size exceeds 1 byte (`word, integer, dword, longint, real`) are always set to alignment 2 (i.e. are always allocated on even address).
Derived types and constant aggregates whose at least one element exceeds size of 1 byte are set to alignment 2.
Routines are always set to aligment 2.

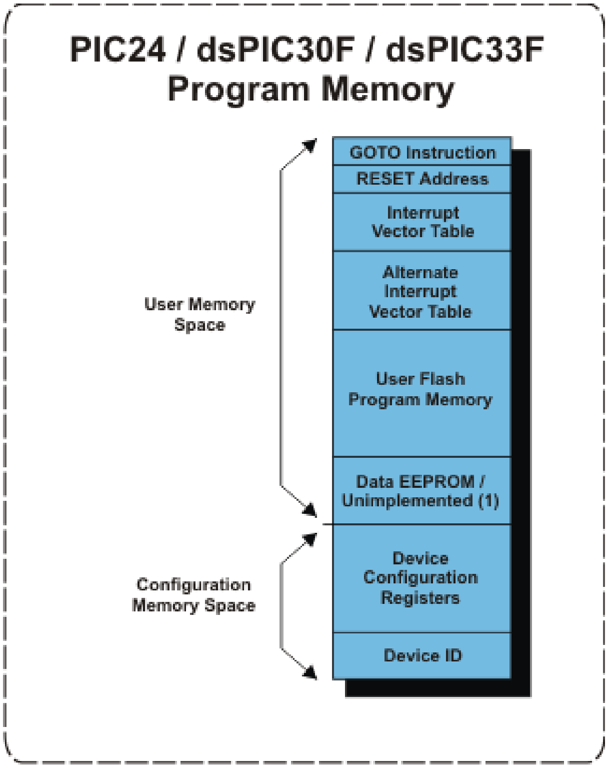
dsPIC Memory Organization

The dsPIC microcontroller’s memory is divided into Program Memory and Data Memory. Program Memory (ROM) is used for permanent saving program being executed, while Data Memory (RAM) is used for temporarily storing and keeping intermediate results and variables.

Program Memory (ROM)

Program Memory (ROM) is used for permanent saving program code being executed, and it is divided into several sections, as on the picture below. The size of these sections is device dependant.
The program memory map is divided into the User Memory Space and Configuration Memory Space. The User Memory Space contains the Reset vector, interrupt vector tables, program memory and data EEPROM memory (dsPIC30 family and some PIC24 family MCU’s).
The Configuration Memory Space contains non-volatile configuration bits for setting device options and the device ID locations.

Program Memory (ROM) Organization



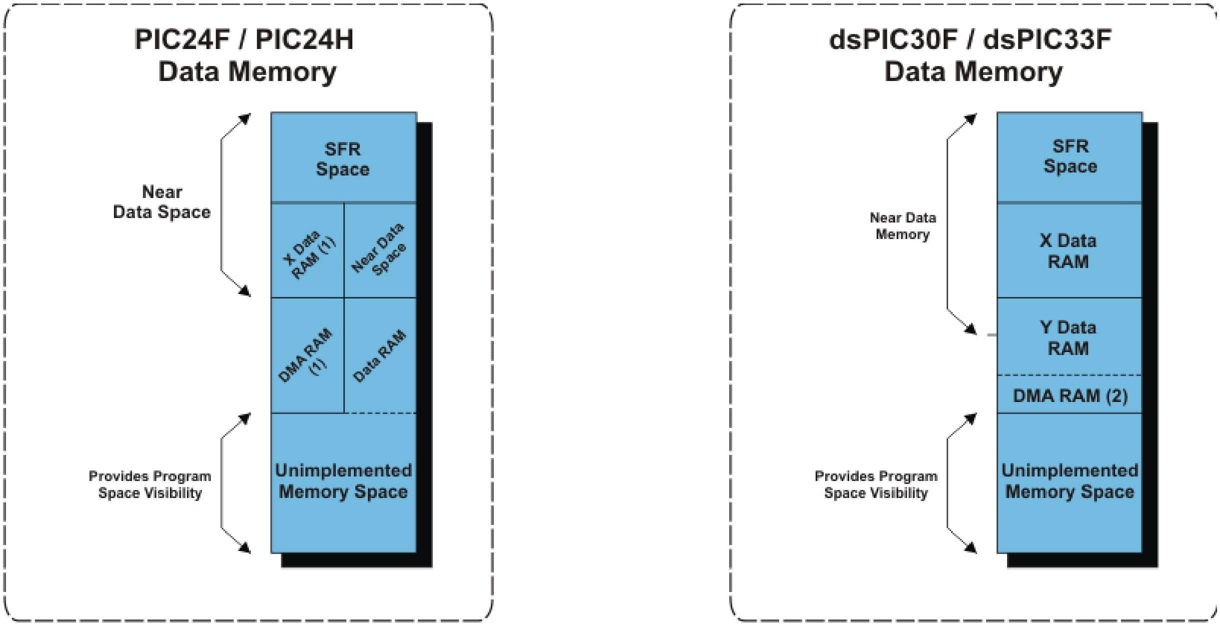
1. dsPIC33F Program Memory Organization

Data Memory (RAM)

Data memory consists of:

- SFR Memory Space
- X and Y Data RAM
- DMA RAM (only for dsPIC33F Family)
- Unimplemented Memory Space

Data Memory (RAM) Organization



1. PIC24F Data Memory Organization
2. dsPIC33F Data Memory Organization

SFR Memory Space

The first 2kB of data memory is allocated to the Special Function Registers (SFRs). The SFRs are control and status register for core and peripheral functions in the dsPIC.

X and Y Data RAM

Up to 8 kB of data RAM is implemented after the SFRs. This is general purpose RAM that can be used for data storage. This RAM is split into X and Y memory for dsPIC instructions.
This allows DSP instructions to support dual operand reads, so that data can be fetched from X and Y memory space at the same time for a single instruction.
The X and Y data space boundary is fixed for any given device. When not doing DSP instructions, the memory is all treated as a single block of X memory.

DMA RAM

Every dsPIC33F device contains a portion of dual ported DMA RAM located at the end of Y data space. Direct Memory Access (DMA) is a very efficient mechanism of copying data between peripheral SFRs and buffers or variables stored in RAM, with minimal CPU intervention.
The DMA controller can automatically copy entire blocks of data without requiring the user software to read or write the peripheral Special Function Registers (SFRs) every time a peripheral interrupt occurs.
The DMA controller uses a dedicated bus for data transfers and therefore, does not steal cycles from the code execution flow of the CPU. To exploit the DMA capability, the corresponding user buffers or variables must be located in DMA RAM.

Unimplemented Memory Space

The last segment of data RAM space is not implemented, but can be mapped into program space for Program Space Visibility. This allows program memory to be read as though it were in data RAM.

Notes:

- Boundaries between memory spaces are device specific. Please, refer to the appropriate datasheet for details.
- Memory spaces are not shown to scale. Please, refer to the appropriate datasheet for details.

There are seven memory type specifiers that can be used to refer to the data memory: `rx, data, code, sfr, xdata, ydata,` and `dma`

Related topics: Accessing individual bits, SFRs, Memory type specifiers, dsPIC Memory Type Qualifiers

Memory Type Specifiers

The mikroBasic PRO for dsPIC30/33 and PIC24 supports usage of all memory areas.

Each variable may be explicitly assigned to a specific memory space by including a memory type specifier in the declaration, or implicitly assigned.

The following memory type specifiers can be used:

- `code`
- `data`
- `rx`
- `sfr`
- `xdata`
- `ydata`
- `dma`

code

Description	The `code` memory type may be used for allocating constants in program memory.
Example	`' puts txt in program memory` `const txt = "Enter parameter" code`

data

Description	This memory specifier is used when storing variable to the Data RAM.
Example	`' puts data_buffer in data ram` `dim data_buffer as char data`

rx

Description	This memory specifier allows variable to be stored in the working registers space (WREG0-WREG15).
Example	`' puts y in the working registers space` `dim y as char rx`

sfr

Description	This memory specifier allows user to access special function registers. It also instructs compiler to maintain same identifier in source and assembly.
Example	`dim y as char sfr ' puts y in SFR space`

xdata

Description	This memory specifier allows user to access X Data memory space.
Example	`dim y as char xdata ' puts x in xdata memory space`

ydata

Description	This memory specifier allows user to access Y Data memory space.
Example	`dim y as char ydata ' puts y in ydata memory space`

dma

Description	This memory specifier allows user to access DMA memory space (dsPIC33F specific).
Example	`dim y as char dma ' puts y in DMA memory space`

Note: If none of the memory specifiers are used when declaring a variable, `data` specifier will be set as default by the compiler.

Related topics: dsPIC Memory Organization, dsPIC Memory Type Qualifiers, Accessing individual bits, SFRs, Constants, Functions

Memory Type Qualifiers

In addition to the standard storage qualifiers(`const, volatile`) the compiler introduces storage qualifiers of `near` and `far.`

Near Memory Qualifier

1. **Data Memory Objects**

   The qualifier `near` is used to denote that a variable is allocated in near data space (the first 8 kB of Data memory). Such variables can sometimes be accessed more efficiently than variables not allocated (or not known to be allocated) in near data space.
   If variables are allocated in the near data section, the compiler is often able to generate better (more compact) code than if the variables are not allocated in the near data section.

2. **Program Memory Objects**

   The qualifier `near` is used to denote that a constant is allocated in the default program memory page (32kB segment of program memory). Default program memory page is the one with most free space and is set by the compiler by analyzing program memory pages.
   This qualifier is set as default by the compiler, if no other qualifier is used.

Far Memory Qualifier

1 **Data Memory Objects**

The qualifier `far` is used to denote that a variable will not be in near data space (i.e. the variable can be located anywhere in data memory). This qualifier is set as default by the compiler, if no other qualifier is used.

2. **Program Memory Objects**

The qualifier `far` is used to denote that a constant can be allocated anywhere in the program memory, in the page pointed to by PSVPAG register.

Location of object based on memory qualifiers:

Qualifier/Memory	Data Memory	Program Memory
`near`	First 8 kB of RAM	In default page
`far`	Anywhere in RAM	In page pointed to PSVPAG register

Example:

```
dim i as char   ' far memory qualifier is set, variable i can allocated somewhere in data memory
dim j as char near ' near memory qualifier is set, variable j will be allocated in the first 8kB of data memory
const k as longint = 10000  ' near memory qualifier is set, constant k will be allocated in the default memory page
```

Related topics: dsPIC Memory Organization, dsPIC Memory Type Specifiers

Read Modify Write Problem

The Microchip microcontrollers use a sequence known as **Read-Modify-Write** (RMW) when changing an output state (1 or 0) on a pin. This can cause unexpected behavior under certain circumstances.

When your program changes the state on a specific pin, for example RB0 in PORTB, the microcontroller first **READs** all 8 bits of the PORTB register which represents the states of all 8 pins in PORTB (RB7-RB0).
The microcontroller then stores this data in the MCU. The bit associated with RB that you've commanded to **MODIFY** is changed, and then the microcontroller **WRITEs** all 8 bits (RB7-RB0) back to the PORTB register.

During the first reading of the PORT register, you will be reading the actual state of the physical pin.
The problem arises when an output pin is loaded in such a way that its logic state is affected by the load. Instances of such loads are LEDs without current-limiting resistors or loads with high capacitance or inductance.

For example, if a capacitor is attached between pin and ground, it will take a short while to charge when the pin is set to 1.
On the other hand, if the capacitor is discharged, it acts like a short circuit, forcing the pin to '0' state, and, therefore, a read of the PORT register will return 0, even though we wrote a 1 to it.

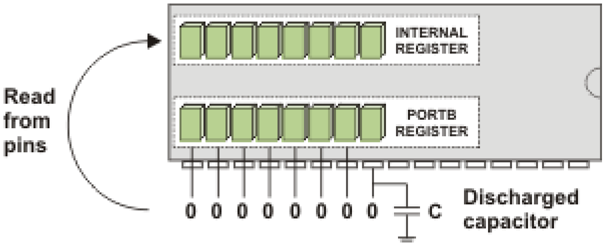
Lets analyze the following example:

```
PORTB.B0 = 1
PORTB.B1 = 1
```

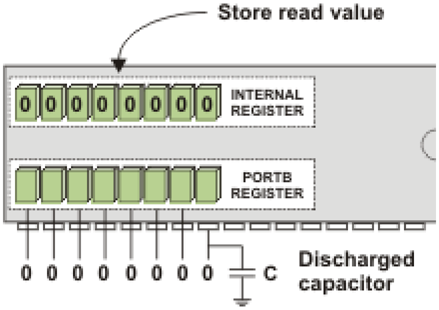
Assume that the PORTB is initially set to zero, and that all pins are set to output. Let's say we connect a discharged capacitor to RB0 pin.

The first line, `PORTB.B0 = 1` will be decoded like in this way:

READ PORTB is **read**:

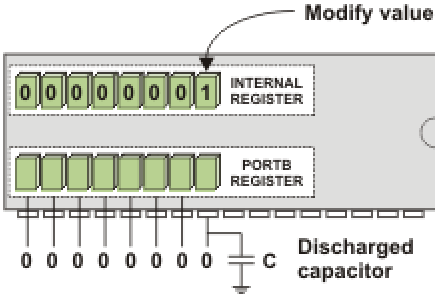


STORE Data is stored inside a temporary internal register in the MCU:

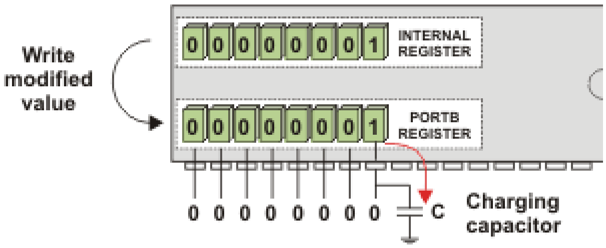


Actual voltage levels on MCU pins are relevant.

MODIFY Data is **modified** to set the RB0 bit:

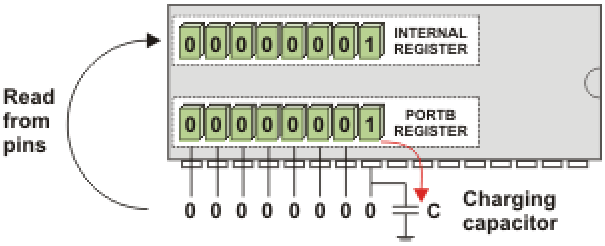


WRITE PORTB is **written** with the modified data. The output driver for RB0 turns on, and the capacitor starts to charge:

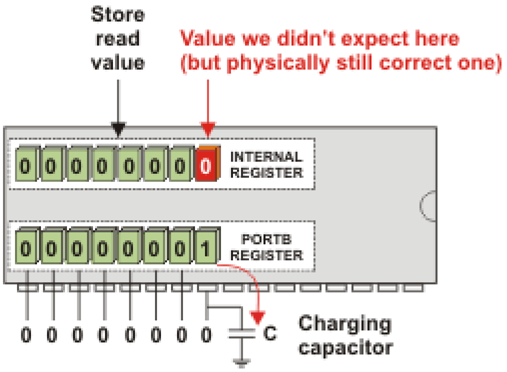


The second line, PORTB.B1 = 1 will be decoded in this way:

READ PORTB is **read**:

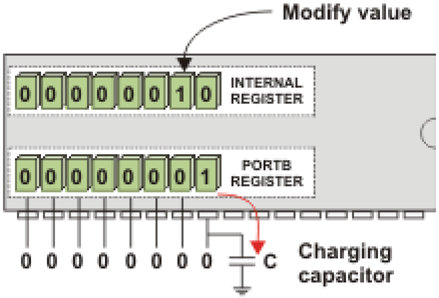


STORE Because the capacitor is still charging, the voltage at RB0 is still low and reads as a ‘0’ (since we are reading from the pins directly, not from the PORTB register):

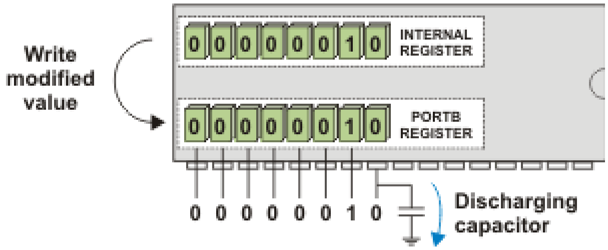


Actual voltage levels on MCU pins are relevant.

MODIFY Data is **modified** to set the bit:



WRITE PORTB is **written** with the new data. The output driver for RB1 turns on, **but the driver for RB0 turns back off:**



To correct the problem in the code, insert a delay after each PORTB.Bx = 1 line, or modify the entire PORTB register in a single line PORTB = 0b00000011.

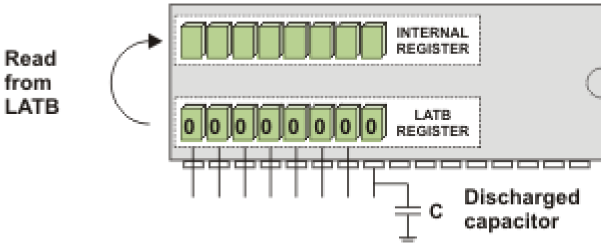
This problem can be avoided by using LATx register when writing to ports, rather than using PORTx registers. Writing to a LATx register is equivalent to writing to a PORTx register, **but readings from LATx registers return the data value held in the port latch, regardless of the state of the actual pin.**

For example, lets analyze the following example:

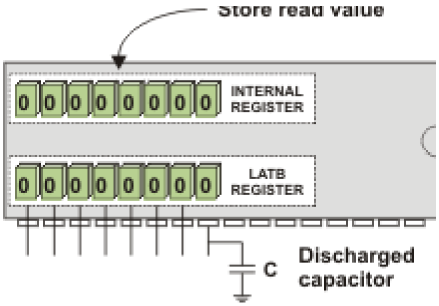
```
LATB.B0 = 1
LATB.B1 = 1
```

The first line, LATB.B0 = 1 will be decoded in this way:

READ LATB is **read**:

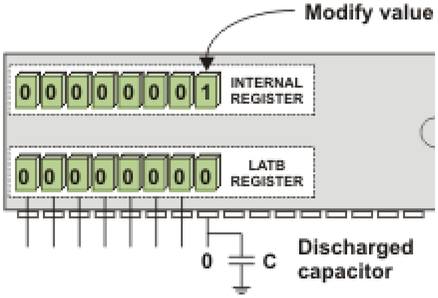


STORE Data is stored inside a temporary internal register in the MCU:

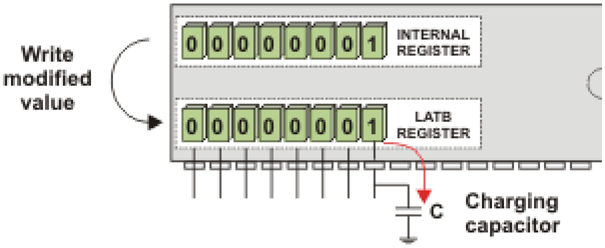


Actual voltage levels on MCU pins are no longer relevant when using LATx for output

MODIFY Data is **modified** to set the RB0 bit:

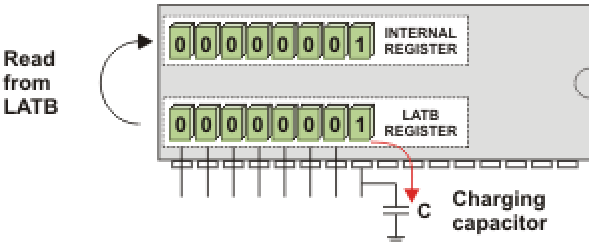


WRITE LATB is **written** with the modified data. The output driver for RB0 turns on, and the capacitor starts to charge:

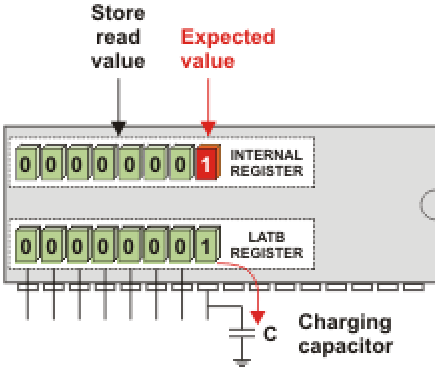


The second line, LATB.B1 = 1 will be decoded in this way:

READ LATB is **read**:

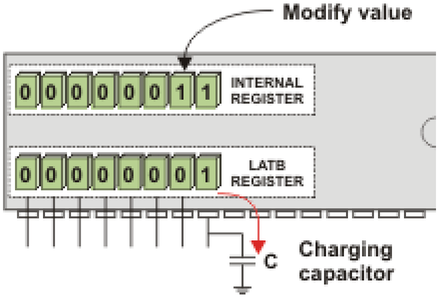


STORE Since the voltage levels on MCU pins are no longer relevant, we get the expected value:

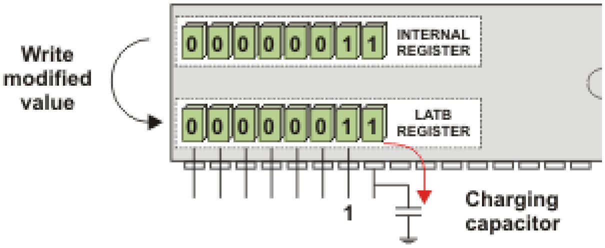


Actual voltage levels on MCU pins are no longer relevant when using LATx for output

MODIFY Data is **modified** to set the bit:



WRITE LATB is **written** with the new data. The output driver for RB1 turns on, and the output driver for RB0 remains turned on:



When to use LATx instead of PORTx

Depending on your hardware, one may experience unpredictable behavior when using PORTx bits for driving output. Displays (GLCD, LCD), chip select pins in SPI interfaces and other cases when you need fast and reliable output, **LATx should be used instead of PORTx**.

CHAPTER 8

mikroBasic PRO for dsPIC30/33 and PIC24 Language Reference

- **Lexical Elements**

    - Whitespace
    - Comments
    - Tokens

        - Literals
        - Keywords
        - Identifiers
        - Punctuators

- **Program Organization**

    - Program Organization
    - Scope and Visibility
    - Modules

- **Variables**
- **Constants**
- **Labels**
- **Symbols**
- **Functions and Procedures**

    - Functions
    - Procedures

- **Types**

    - Simple Types
    - Arrays
    - Strings
    - Pointers
    - Structures
    - Type Conversions

        - Implicit Conversion
        - Explicit Conversion

- **Operators**

    - Introduction to Operators
    - Operators Precedence and Associativity
    - Arithmetic Operators
    - Relational Operators
    - Bitwise Operators
    - Boolean Operators

- **Expressions**

    - Expressions

- **Statements**
    - Introduction to Statements
    - Assignment Statements
    - Conditional Statements
        - If Statement
        - Select Case Statement
    - Iteration Statements (Loops)
        - For Statement
        - While Statement
        - Do Statement
    - Jump Statements
        - Break and Continue Statements
        - Exit Statement
        - Goto Statement
        - Gosub Statement
    - asm Statement
- **Directives**
    - Compler Directives
    - Linker Directives

Lexical Elements Overview

The following topics provide a formal definition of the mikroBasic PRO for dsPIC30/33 and PIC24 lexical elements. They describe different categories of word-like units (tokens) recognized by the language.

In the tokenizing phase of compilation, the source code file is parsed (i.e. broken down) into tokens and whitespace. The tokens in mikroBasic PRO for dsPIC30/33 and PIC24 are derived from a series of operations performed on your programs by the compiler.

A mikroBasic PRO for dsPIC30/33 and PIC24 program starts as a sequence of ASCII characters representing the source code, created by keystrokes using a suitable text editor (such as the mikroBasic PRO for dsPIC30/33 and PIC24 Code Editor). The basic program unit in mikroBasic PRO for dsPIC30/33 and PIC24 is a file. This usually corresponds to a named file located in RAM or on disk, having the extension `.mbas`.

Whitespace

Whitespace is a collective name given to spaces (blanks), horizontal and vertical tabs, newline characters and comments. Whitespace can serve to indicate where tokens start and end, but beyond this function, any surplus whitespace is discarded.

For example, the two sequences

```
dim tmp as byte
dim j as word
```

and

```
dim  tmp  as  byte
dim    j  as  word
```

are lexically equivalent and parse identically.

Newline Character

Newline character (CR/LF) is not a whitespace in BASIC, and serves as a statement terminator/separator. In mikroBasic PRO for dsPIC30/33 and PIC24, however, you *may* use newline to break long statements into several lines. Parser will first try to get the longest possible expression (across lines if necessary), and then check for statement terminators.

Whitespace in Strings

The ASCII characters representing whitespace can occur within string literals, in which case they are protected from the normal parsing process (they remain as a part of the string). For example, statement

```
some_string = “mikro foo”
```

parses to four tokens, including a single string literal token:

```
some_string
=
“mikro foo”
newline character
```

Comments

Comments are pieces of text used to annotate a program, and are technically another form of whitespace. Comments are for the programmer's use only; they are stripped from the source text before parsing.

Use the apostrophe to create a comment:

```
' Any text between an apostrophe and the end of the
' line constitutes a comment. May span one line only.
```

There are no multi-line comments in mikroBasic PRO for dsPIC30/33 and PIC24

Tokens

Token is the smallest element of a mikroBasic PRO for dsPIC30/33 and PIC24 program, meaningful to the compiler. The parser separates tokens from the input stream by creating the longest token possible using the input characters in a left–to–right scan.

mikroBasic PRO for dsPIC30/33 and PIC24 recognizes the following kinds of tokens:

- keywords
- identifiers
- constants
- operators
- punctuators (also known as separators)

Token Extraction Example

Here is an example of token extraction. Take a look at the following example code sequence:

```
end_flag = 0
```

First, note that `end_flag` would be parsed as a single identifier, rather than as the keyword `end` followed by the identifier `_flag`.

The compiler would parse it as the following four tokens:

```
end_flag    ' variable identifier
=           ' assignment operator
0           ' literal
newline     ' statement terminator
```

Literals

Literals are tokens representing fixed numeric or character values.

The data type of a constant is deduced by the compiler using such clues as numeric value and format used in the source code.

Integer Literals

Integral values can be represented in decimal, hexadecimal or binary notation.

In decimal notation, numerals are represented as a sequence of digits (without commas, spaces or dots), with optional prefix `+` or `-` operator to indicate the sign. Values default to positive (`6258` is equivalent to `+6258`).

The dollar-sign prefix (`$`) or the prefix 0x indicates a hexadecimal numeral (for example, `$8F` or `0x8F`).

The percent-sign prefix (`%`) indicates a binary numeral (for example, `%0101`).

Here are some examples:

```
11        ' decimal literal
$11       ' hex literal, equals decimal 17
0x11      ' hex literal, equals decimal 17
%11       ' binary literal, equals decimal 3
```

The allowed range of values is imposed by the largest data type in mikroBasic PRO for dsPIC30/33 and PIC24 – `longword`. The compiler will report an error if the literal exceeds `4294967295 ($FFFFFFFF)`.

Floating Point Literals

A floating-point value consists of:

- Decimal integer
- Decimal point
- Decimal fraction
- e or E and a signed integer exponent (optional)

You can omit either decimal integer or decimal fraction (but not both).

Negative floating constants are taken as positive constants with the unary operator minus (-) prefixed.

mikroBasic PRO for dsPIC30/33 and PIC24 limits floating-point constants to the range of ±1.17549435082 * 10^{-38} .. ±6.80564774407 * 10^{38}.

Here are some examples:

```
0.         ' = 0.0
-1.23      ' = -1.23
23.45e6    ' = 23.45 * 10^6
```

```
2e-5      ' = 2.0 * 10^-5
3E+10     ' = 3.0 * 10^10
.09E34    ' = 0.09 * 10^34
```

Character Literals

Character literal is one character from the extended ASCII character set, enclosed with quotes (for example, `"A"`). Character literal can be assigned to variables of `byte` and `char` type (variable of `byte` will be assigned the ASCII value of the character). Also, you can assign character literal to a string variable.

String Literals

String literal is a sequence of characters from the extended ASCII character set, enclosed with quotes. Whitespace is preserved in string literals, i.e. parser does not "go into" strings but treats them as single tokens.

Length of string literal is a number of characters it consists of. String is stored internally as the given sequence of characters plus a final `null` character. This `null` character is introduced to terminate the string, it does not count against the string's total length.

String literal with nothing in between the quotes (*null string*) is stored as a single `null` character.

You can assign string literal to a string variable or to an array of `char`.

Here are several string literals:

```
"Hello world!"               ' message, 12 chars long
"Temperature is stable"      ' message, 21 chars long
"  "                         ' two spaces, 2 chars long
"C"                          ' letter, 1 char long
""                           ' null string, 0 chars long
```

The quote itself cannot be a part of the string literal, i.e. there is no escape sequence. You could use the built-in function Chr to print a quote: `Chr(34)`. Also, see String Splicing.

Keywords

Keywords are special-purpose words which cannot be used as normal identifier names.

Beside standard BASIC keywords, all relevant SFR are defined as global variables and represent reserved words that cannot be redefined (for example: `P0, TMR1, T1CON`, etc). Probe Code Assistant for specific letters (Ctrl+Space in Editor) or refer to Predefined Globals and Constants.

Here is the alphabetical listing of keywords in mikroBasic PRO for dsPIC30/33 and PIC24:

- `absolute`
- `abstract`
- `and`
- `array`
- `as`
- `asm`
- `assembler`
- `at`
- `automated`
- `bdata`
- `begin`
- `bit`
- `case`
- `cdecl`
- `class`
- `code`
- `compact`
- `const`
- `constructor`
- `contains`
- `data`
- `default`
- `deprecated`
- `destructor`
- `dispid`
- `dispinterface`
- `div`
- `dma`
- `do`
- `downto`
- `dynamic`
- `end`
- `except`
- `export`
- `exports`
- `external`
- `far`
- `file`
- `final`
- `finalization`
- `finally`
- `for`

- forward
- goto
- helper
- idata
- if
- ilevel
- implementation
- implements
- in
- index
- inherited
- initialization
- inline
- interface
- io
- is
- iv
- label
- library
- message
- mod
- name
- near
- nil
- nodefault
- not
- object
- of
- on
- operator
- or
- org
- out
- overload
- override
- package
- packed
- pascal
- pdata
- platform
- private
- procedure
- program
- property
- protected
- public
- published
- raise
- read
- readonly
- record
- register

- `reintroduce`
- `repeat`
- `requires`
- `rx`
- `safecall`
- `sbit`
- `sealed`
- `set`
- `sfr`
- `shl`
- `shr`
- `small`
- `stdcall`
- `stored`
- `string`
- `threadvar`
- `to`
- `try`
- `type`
- `unit`
- `until`
- `uses`
- `var`
- `virtual`
- `volatile`
- `while`
- `with`
- `write`
- `writeonly`
- `xdata`
- `xor`
- `ydata`

Also, mikroBasic PRO for dsPIC30/33 and PIC24 includes a number of predefined identifiers used in libraries. You can replace them by your own definitions, if you plan to develop your own libraries. For more information, see mikroBasic PRO for dsPIC30/33 and PIC24 Libraries.

Identifiers

Identifiers are arbitrary names of any length given to functions, variables, symbolic constants, user-defined data types and labels. All these program elements will be referred to as objects throughout the help (don't be confused with the meaning of *object* in object-oriented programming).

Identifiers can contain letters from `a` to `z` and `A` to `Z`, the underscore character "_" and digits from `0` to `9`. First character must be a letter or an underscore, i.e. identifier cannot begin with a numeral.

Case Sensitivity

mikroBasic PRO for dsPIC30/33 and PIC24 is not case sensitive, so `Sum`, `sum`, and `suM` are equivalent identifiers.

Uniqueness and Scope

Although identifier names are arbitrary (within the rules stated), errors result if the same name is used for more than one identifier within the same scope. Simply, duplicate names are *illegal* within the same scope. For more information, refer to Scope and Visibility.

Identifier Examples

Here are some valid identifiers:

```
temperature_V1
Pressure
no_hit
dat2string
SUM3
_vtext
```

… and here are some invalid identifiers:

```
7temp        ' NO -- cannot begin with a numeral
%higher      ' NO -- cannot contain special characters
xor          ' NO -- cannot match reserved word
j23.07.04    ' NO -- cannot contain special characters (dot)
```

Punctuators

The mikroBasic PRO for dsPIC30/33 and PIC24 punctuators (also known as separators) are:

- [ ] – Brackets
- ( ) – Parentheses
- , – Comma
- : – Colon
- . – Dot

Brackets

Brackets `[ ]` indicate single and multidimensional array subscripts:

```
dim alphabet as byte[30]
` ...
alphabet[2] = "c"
```

For more information, refer to Arrays.

Parentheses

Parentheses `( )` are used to group expressions, isolate conditional expressions and indicate function calls and function declarations:

```
d = c * (a + b)                  ' Override normal precedence
if (d = z) then ...              ' Useful with conditional statements
func()                           ' Function call, no arguments
sub function func2(dim n as word) ' Function declaration w/ parameters
```

For more information, refer to Operators Precedence and Associativity, Expressions and Functions and Procedures.

Comma

Comma (`,`) separates the arguments in function calls:

```
Lcd_Out(1, 1, txt)
```

Furthermore, the comma separates identifiers in declarations:

```
dim i, j, k as word
```

The comma also separates elements in initialization lists of constant arrays:

```
const MONTHS as byte[12] = (31,28,31,30,31,30,31,31,30,31,30,31)
```

Colon

Colon (`:`) is used to indicate a labeled statement:

```
start:  nop
  '...
goto start
```

For more information, refer to Labels.

Dot

Dot (`.`) indicates access to a structure member. For example:

```
person.surname = "Smith"
```

For more information, refer to Structures.

Dot is a necessary part of floating point literals. Also, dot can be used for accessing individual bits of registers in mikroBasic PRO for dsPIC30/33 and PIC24.

Program Organization

mikroBasic PRO for dsPIC30/33 and PIC24 imposes strict program organization. Below you can find models for writing legible and organized source files. For more information on file inclusion and scope, refer to Modules and to Scope and Visibility.

Organization of Main Module

Basically, the main source file has two sections: declaration and program body. Declarations should be in their proper place in the code, organized in an orderly manner. Otherwise, the compiler may not be able to comprehend the program correctly.

When writing code, follow the model presented below. The main module should look like this:

```
program <program name>
include <include other modules>

'********************************************************
'* Declarations (globals):
'********************************************************

' symbols declarations
symbol ...

' constants declarations
const ...
```

```
' structures declarations
structure ...

' variables declarations
dim  Name[,  Name2...]  as  [^]type  [absolute  0x123]  [external]  [volatile]  [register]
[sfr]

' procedures declarations
sub procedure procedure_name(...)
  <local declarations>
  ...
end sub

' functions declarations
sub function function_name(...) as return_type
  <local declarations>
  ...
end sub

'*****************************************************
'* Program body:
'*****************************************************

main:
  ' write your code here
end.
```

Organization of Other Modules

Modules other than main start with the keyword `module`. Implementation section starts with the keyword `implements`. Follow the model presented below:

```
module  <module name>
include <include other modules>

'*****************************************************
'* Interface (globals):
'*****************************************************

' symbols declarations
symbol ...

' constants declarations
const ...

' structures declarations
structure ...

' variables declarations
dim  Name[,  Name2...]  as  [^]type  [absolute  0x123]  [external]  [volatile]  [register]
[sfr]
```

```
' procedures prototypes
sub procedure sub_procedure_name([dim byref] [const] ParamName as [^]type, [dim byref]
[const] ParamName2, ParamName3 as [^]type)

' functions prototypes
sub function sub_function_name([dim byref] [const] ParamName as [^]type, [dim byref]
[const] ParamName2, ParamName3 as [^]type) as [^]type

'*****************************************************
'* Implementation:
'*****************************************************

implements

' constants declarations
const ...

' variables declarations
dim ...

' procedures declarations
sub procedure sub_procedure_name([dim byref] [const] ParamName as [^]type, [dim byref]
[const] ParamName2, ParamName3 as [^]type) [ilevel 0x123] [overload] [forward]
  <local declarations>
  ...
end sub

' functions declarations
sub function sub_function_name([dim byref] [const] ParamName as [^]type, [dim byref]
[const] ParamName2, ParamName3 as [^]type) as [^]type [ilevel 0x123] [overload]
[forward]
  <local declarations>
  ...
end sub

end.
```

Note: Sub functions and sub procedures must have the same declarations in the interface and implementation section. Otherwise, compiler will report an error.

Scope and Visibility

Scope

The scope of an identifier is a part of the program in which the identifier can be used to access its object. There are different categories of scope, which depends on how and where identifiers are declared:

Place of declaration	Scope
Identifier is declared in the declaration section of the main module, out of any function or procedure	Scope extends from the point where it is declared to the end of the current file, including all routines enclosed within that scope. These identifiers have a *file scope* and are referred to as *globals*.
Identifier is declared in the function or procedure	Scope extends from the point where it is declared to the end of the current routine. These identifiers are referred to as *locals*.
Identifier is declared in the interface section of the module	Scope extends the interface section of a module from the point where it is declared to the end of the module, and to any other module or program that uses that module. The only exception are symbols which have a scope limited to the file in which they are declared.
Identifier is declared in the implementation section of the module, but not within any function or procedure	Scope extends from the point where it is declared to the end of the module. The identifier is available to any function or procedure in the module.

Visibility

The visibility of an identifier is that region of the program source code from which legal access to the identifier's associated object can be made.

Scope and visibility usually coincide, though there are circumstances under which an object becomes temporarily hidden by the appearance of a duplicate identifier, i.e. the object still exists but the original identifier cannot be used to access it until the scope of the duplicate identifier is ended.

Technically, visibility cannot exceed scope, but scope *can* exceed visibility.

Name Spaces

Name space is a scope within which an identifier must be unique. The mikroBasic PRO for dsPIC30/33 and PIC24 uses two distinct categories of identifiers:

1. Global variables are visible throughout the whole unit, from the place of declaration. Also. they can be seen in other units, if they are declared above the Implementation section.
2. Local variables, parameters, types, function results - must be unique within the block in which they are declared.

For example:

```
dim level as byte

sub procedure control(dim sens as byte)
  dim location as byte
  location = 1
  sens = location
  level = 123
end sub

sub procedure temperature
  location = 0        ' ILLEGAL
  sens = 23           ' ILLEGAL: redefinition of sens
  level = 95
end sub
```

Modules

In mikroBasic PRO for dsPIC30/33 and PIC24, each project consists of a single project file and one or more module files. The project file, with extension `.mbpds` contains information on the project, while modules, with extension `.mbas`, contain the actual source code. See Program Organization for a detailed look at module arrangement.

Modules allow you to:

- break large programs into encapsulated modules that can be edited separately,
- create libraries that can be used in different projects,
- distribute libraries to other developers without disclosing the source code.

Each module is stored in its own file and compiled separately; compiled modules are linked to create an application. To build a project, the compiler needs either a source file or a compiled module file for each module.

Include Clause

mikroBasic PRO for dsPIC30/33 and PIC24 includes modules by means of the `include` clause. It consists of the reserved word `include`, followed by a quoted module name. Extension of the file should not be included.

You can include one file per `include` clause. There can be any number of the `include` clauses in each source file, but they all must be stated immediately after the program (or module) name.

Here's an example:

```
program MyProgram

include "utils"
include "strings"
include "MyUnit"

...
```

For the given module name, the compiler will check for the presence of `.mcl` and `.mbas` files, in order specified by search paths.

- If both `.mbas` and `.mcl` files are found, the compiler will check their dates and include the newer one in the project. If the `.mbas` file is newer than the `.mcl`, then `.mbas` file will be recompiled and new `.mcl` will be created, overwriting the old `.mcl`.
- If only the `.mbas` file is found, the compiler will create the `.mcl` file and include it in the project;
- If only the `.mcl` file is present, i.e. no source code is available, the compiler will include it as found;
- If none of the files found, the compiler will issue a "File not found" warning.

Main Module

Every project in mikroBasic PRO for dsPIC30/33 and PIC24 requires a single main module file. The main module is identified by the keyword `program` at the beginning. It instructs the compiler where to "start".

After you have successfully created an empty project with Project Wizard, Code Editor will display a new main module. It contains the bare-bones of the program:

```
program MyProject

' main procedure
main:
  ' Place program code here
end.
```

Other than comments, nothing should precede the keyword `program`. After the program name, you can optionally place the `include` clauses.

Place all global declarations (constants, variables, labels, routines, structures) before the label `main`.

Other Modules

Modules other than main start with the keyword `module`. Newly created blank module contains the bare-bones:

```
module MyModule

implements

end.
```

Other than comments, nothing should precede the keyword `module`. After the module name, you can optionally place the `include` clauses.

Interface Section

Part of the module above the keyword `implements` is referred to as *interface* section. Here, you can place global declarations (constants, variables, labels, routines, structures) for the project.

Do *not* define routines in the interface section. Instead, state the prototypes of routines (from implementation section) that you want to be visible outside the module. Prototypes must exactly match the declarations.

Implementation Section

Implementation section hides all the irrelevant innards from other modules, allowing encapsulation of code.

Everything declared below the keyword `implements` is *private*, i.e. has its scope limited to the file. When you declare an identifier in the implementation section of a module, you cannot use it outside the module, but you can use it in any block or routine defined within the module.

By placing the prototype in the interface section of the module (above the `implements`) you can make the routine *public*, i.e. visible outside of module. Prototypes must exactly match the declarations.

Variables

Variable is an object whose value can be changed during the runtime. Every variable is declared under unique name which must be a valid identifier. This name is used for accessing the memory location occupied by a variable.

Variables are declared in the declaration part of the file or routine — each variable needs to be declared before being used. Global variables (those that do not belong to any enclosing block) are declared below the `include` statements, above the label `main`.

Specifying a data type for each variable is mandatory. Syntax for variable declaration is:

```
dim identifier_list as type
```

Here, `identifier_list` is a comma-delimited list of valid identifiers, and `type` can be any data type.

For more details refer to Types and Types Conversions. For more information on variables' scope refer to the chapter Scope and Visibility.

Here are a few examples:

```
dim i, j, k as byte
dim counter, temp as word
dim samples as longint[100]
```

External Modifier

Use the `external` modifier to indicate that the actual place and initial value of the variable, sub function or sub procedure body, is defined in a separate source code module.

For example, lets create a project which will calculate circle area and will have sub function and sub procedure definition in two different modules, and a call to these routines in the third, separate module.
So, the project will be consisted of the main module, `Main_Module.mpas` and `First_Module.mpas` and `Second_Module.mpas` modules.

In the `Main_Module` we will define routine called `r_squared` (calculates radius squared). Also, both modules must be included in the `Main_Module`:

```
program Main_Module

include First_Module
include Second_Module   ' Include both used modules

sub function r_square(dim r as float) as float   ' Definition of the r_square routine
  result = r*r;
end sub

main:
  CircleArea()   ' CircleArea routine call
end.
end.
```

In the `First_Module` we will define and declare routine called `pi_r_squared` (calculates pi multiplied by the radius squared):

```
module First_Module

sub procedure pi_r_square(dim rr as float)   ' Declaration of the pi_r_square routine

implements

sub procedure pi_r_square(dim rr as float)   ' Definition of the pi_r_square routine
  dim res as float
  res = rr*3.14
end sub

end.
```

In the `Second_Module` we will make a call to the routines defined externally (`r_squared` and `pi_r_squared`). First of all, we must declare their prototypes followed with a `external` modifier. Then, we can proceed to the routine call :

```
module Second_Module

sub procedure CircleArea()
sub function r_square(dim r as float) as float external   ' Declaration of the r_square
routine (defined in Main_Module) followed with a external modifier
sub procedure pi_r_square(dim rr as float) external      ' Declaration of the pi_r_square
routine (defined in Second_Module) followed with a external modifier

implements

sub procedure CircleArea()  ' Definition of the CircleArea routine
  dim res as real
  res = r_square(5)   ' Calculate r*r
  pi_r_square(res)    ' Calculate pi*r*r
end sub

end.
```

Variables and dsPIC30/33 and PIC24

Every declared variable consumes part of RAM memory. Data type of variable determines not only the allowed range of values, but also the space a variable occupies in RAM memory. Bear in mind that operations using different types of variables take different time to be completed. mikroBasic PRO for dsPIC30/33 and PIC24 recycles local variable memory space – local variables declared in different functions and procedures share the same memory space, if possible.

There is no need to declare SFRs explicitly, as mikroBasic PRO for dsPIC30/33 and PIC24 automatically declares relevant registers as global variables of `word`. For example: `W0, TMR1`, etc.

Constants

Constant is a data whose value cannot be changed during the runtime. Using a constant in a program consumes no RAM memory. Constants can be used in any expression, but cannot be assigned a new value.

Constants are declared in the declaration part of a program or routine. You can declare any number of constants after the keyword `const`:

```
const constant_name [as type] = value
```

Every constant is declared under unique *`constant_name`* which must be a valid identifier. It is a tradition to write constant names in uppercase. Constant requires you to specify *`value`*, which is a literal appropriate for the given type. *`type`* is optional and in the absence of it , the compiler assumes the “smallest” type that can accommodate *`value`*.

Note: You cannot omit type when declaring a constant array.

Here are a few examples:

```
const MAX as longint = 10000
const MIN = 1000       ' compiler will assume word type
const SWITCH = "n"     ' compiler will assume char type
const MSG = "Hello"    ' compiler will assume string type
const MONTHS as byte[12] = (31,28,31,30,31,30,31,31,30,31,30,31)
```

Labels

Labels serve as targets for goto and gosub statements. Mark the desired statement with label and colon like this:

```
label_identifier : statement
```

No special declaration of label is necessary in mikroBasic PRO for dsPIC30/33 and PIC24.

Name of the label needs to be a valid identifier. The labeled statement and `goto/gosub` statement must belong to the same block. Hence it is not possible to jump into or out of routine. Do not mark more than one statement in a block with the same label.

Note:

- The label `main` marks the entry point of a program and must be present in the main module of every project. See Program Organization for more information.
- Label should be followed by end of line (CR) otherwise compiler will report an error.

Here is an example of an infinite loop that calls the procedure `Beep` repeatedly:

```
loop:
  Beep
goto loop
```

Symbols

mikroBasic PRO fordsPIC30/33 and PIC24 symbols allow you to create simple macros without parameters. You can replace any line of code with a single identifier alias. Symbols, when properly used, can increase code legibility and reusability.

Symbols need to be declared at the very beginning of the module, right after the module name and (optional) `include` clauses. Check Program Organization for more details. Scope of a symbol is always limited to the file in which it has been declared.

Symbol is declared as:

```
symbol alias = code
```

Here, `alias` must be a valid identifier which you will use throughout the code. This identifier has a file scope. The `code` can be any line of code (literals, assignments, function calls, etc).

Using a symbol in the program consumes no RAM – the compiler will simply replace each instance of a symbol with the appropriate line of code from the declaration.

Here is an example:

```
symbol MAXALLOWED = 216           ' Symbol as alias for numeric value
symbol PORT = PORTC               ' Symbol as alias for SFR
symbol MYDELAY = Delay_ms(1000)   ' Symbol as alias for procedure call

dim cnt as byte  ' Some variable

'...
main:

if cnt > MAXALLOWED then
  cnt = 0
  PORT.1 = 0
  MYDELAY
end if
```

Note: Symbols do not support macro expansion in a way the C preprocessor does.

Functions and Procedures

Functions and procedures, collectively referred to as *routines*, are subprograms (self-contained statement blocks) which perform a certain task based on a number of input parameters. When executed, a function returns value while procedure does not.

Functions

A function is declared like this:

```
sub function function_name(parameter_list) as return_type
  [ local declarations ]
  function body
end sub
```

`function_name` represents a function's name and can be any valid identifier. `return_type` is a type of return value and can be any simple type or complex type. Within parentheses, `parameter_list` is a formal parameter list very similar to variable declaration. In mikroBasic PRO for PIC, parameters are always passed to a function by the value. To pass an argument by address, add the keyword `byref` ahead of identifier.

`Local declarations` are optional declarations of variables and/or constants, local for the given function. `Function body` is a sequence of statements to be executed upon calling the function.

Calling a function

A function is called by its name, with actual arguments placed in the same sequence as their matching formal parameters. The compiler is able to coerce mismatching arguments to the proper type according to implicit conversion rules. Upon a function call, all formal parameters are created as local objects initialized by values of actual arguments. Upon return from a function, a temporary object is created in the place of the call and it is initialized by the value of the function result. This means that function call as an operand in complex expression is treated as the function result.

In standard Basic, a `function_name` is automatically created local variable that can be used for returning a value of a function. mikroBasic PRO for dsPIC30/33 and PIC24 also allows you to use the automatically created local variable `result` to assign the return value of a function if you find function name to be too ponderous. If the return value of a function is not defined the compiler will report an error.

Function calls are considered to be *primary expressions* and can be used in situations where expression is expected. A function call can also be a self-contained statement and in that case the return value is discarded.

Example

Here's a simple function which calculates x^n based on input parameters `x` and `n` (`n > 0`):

```
sub function power(dim x, n as byte) as longint
dim i as byte
  result = 1
  if n > 0 then
    for i = 1 to n
      result = result*x
    next i
  end if
end sub
```

Now we could call it to calculate, say, 3^{12}:

```
tmp = power(3, 12)
```

Procedures

Procedure is declared like this:

```
sub procedure procedure_name(parameter_list)
  [ local declarations ]
  procedure body
end sub
```

`procedure_name` represents a procedure's name and can be any valid identifier. Within parentheses, *`parameter_list`* is a formal parameter list similar to variable declaration. In mikroBasic PRO for dsPIC30/33 and PIC24, parameters are always passed to procedure by value; to pass argument by address, add the keyword `byref` ahead of identifier.

`Local declarations` are optional declaration of variables and/or constants, local for the given procedure. *`Procedure body`* is a sequence of statements to be executed upon calling the procedure.

Calling a procedure

A procedure is called by its name, with actual arguments placed in the same sequence as their matching formal parameters. The compiler is able to coerce mismatching arguments to the proper type according to implicit conversion rules. Upon procedure call, all formal parameters are created as local objects initialized by the values of actual arguments.

Procedure call is a self-contained statement.

Example:

Here's an example procedure which transforms its input time parameters, preparing them for output on Lcd:

```
sub procedure time_prep(dim byref sec, min, hr as byte)
  sec  = ((sec and $F0) >> 4)*10 + (sec and $0F)
  min  = ((min and $F0) >> 4)*10 + (min and $0F)
  hr   = ((hr  and $F0) >> 4)*10 + (hr  and $0F)
end sub
```

A function can return a complex type. Follow the example bellow to learn how to declare and use a function which returns a complex type.

Example:

This example shows how to declare a function which returns a complex type.

```
program Example

structure TCircle        ' Structure
  dim CenterX, CenterY as word
  dim Radius as byte
end structure

dim MyCircle as TCircle ' Global variable

sub function DefineCircle(dim x, y as word, dim r as byte) as TCircle ' DefineCircle
function returns a Structure
  result.CenterX = x
  result.CenterY = y
  result.Radius = r
end sub

main:
  MyCircle = DefineCircle(100, 200, 30)          ' Get a Structure via function call
  MyCircle.CenterX = DefineCircle(100, 200, 30).CenterX + 20   ' Access a Structure field
via function call
  '                   |------------------------| |-----|
  '                      |                          |
  '                   Function returns TCircle     Access to one field of TCircle
end.
```

Forward declaration

A function can be declared without having it followed by it's implementation, by having it followed by the forward procedure. The effective implementation of that function must follow later in the module. The function can be used after a forward declaration as if it had been implemented already. The following is an example of a forward declaration:

```
program Volume

dim Volume as word

sub function First(dim a as word, dim b as word) as word forward

sub function Second(dim c as word) as word
dim tmp as word
  tmp = First(2, 3)
  result = tmp * c
end sub

sub function First(dim a, b as word) as word
  result = a * b
end sub

main:
  Volume = Second(4)
end.
```

Functions reentrancy

Functions reentrancy is allowed. Remember that the dsPIC30/33 and PIC24 have memory limitations that can vary between MCUs.

Types

Basic is strictly typed language, which means that every variable and constant need to have a strictly defined type, known at the time of compilation.

The type serves:

- to determine the correct memory allocation required,
- to interpret the bit patterns found in the object during subsequent accesses,
- in many type-checking situations, to ensure that illegal assignments are trapped.

mikroBasic PRO for dsPIC30/33 and PIC24 supports many standard (predefined) and user-defined data types, including signed and unsigned integers of various sizes, arrays, strings, pointers and structures.

Type Categories

Types can be divided into:

- simple types
- arrays
- strings
- pointers
- structures

Simple Types

Simple types represent types that cannot be divided into more basic elements and are the model for representing elementary data on machine level. Basic memory unit in mikroBasic PRO for dsPIC30/33 and PIC24 has 16 bits.

Here is an overview of simple types in mikroBasic PRO for dsPIC30/33 and PIC24:

Type	Size	Range
`bit`	1–bit	0 or 1
`sbit`	1–bit	0 or 1
`byte, char`	8–bit	0 .. 255
`short`	8–bit	-127 .. 128
`word`	16–bit	0 .. 65535
`integer`	16–bit	32768 .. 32767
`longword`	32–bit	0 .. 4294967295
`longint`	32–bit	2147483648 .. 2147483647
`float`	32–bit	$\pm 1.17549435082 * 10^{-38}$ .. $\pm 6.80564774407 * 10^{38}$

You can assign signed to unsigned or vice versa only using the explicit conversion. Refer to Types Conversions for more information.

Derived Types

The derived types are also known as *structured types*. They are used as elements in creating more complex user-defined types.

The derived types include:

- arrays
- pointers
- structures

Arrays

An array represents an indexed collection of elements of the same type (called the base type). Since each element has a unique index, arrays, unlike sets, can meaningfully contain the same value more than once.

Array Declaration

Array types are denoted by constructions in the following form:

```
type[array_length]
```

Each of the elements of an array is numbered from `0` through `array_length - 1`.

Every element of an array is of `type` and can be accessed by specifying array name followed by element's index within brackets.

Here are a few examples of array declaration:

```
dim weekdays as byte[7]
dim samples  as word[50]

main:
  ' Now we can access elements of array variables, for example:
  samples[0] = 1
  if samples[37] = 0 then
    ' ...
```

Constant Arrays

Constant array is initialized by assigning it a comma-delimited sequence of values within parentheses. For example:

```
' Declare a constant array which holds number of days in each month:
const MONTHS as byte[12] = (31,28,31,30,31,30,31,31,30,31,30,31)
```

Note that indexing is zero based; in the previous example, number of days in January is `MONTHS[0]` and number of days in December is `MONTHS[11]`.

The number of assigned values must not exceed the specified length. Vice versa is possible, when the trailing “excess” elements will be assigned zeroes.

For more information on arrays of `char`, refer to Strings.

Multi-dimensional Arrays

Multidimensional arrays are constructed by declaring arrays of array type. These arrays are stored in memory in such way that the right most subscript changes fastest, i.e. arrays are stored “in rows”. Here is a sample 2-dimensional array:

```
dim m as byte[5][10]    ' 2-dimensional array of size 5x10
```

A variable `m` is an array of 5 elements, which in turn are arrays of 10 byte each. Thus, we have a matrix of 5x10 elements where the first element is `m[0][0]` and last one is `m[4][9]`. The first element of the 4th row would be `m[3][0]`.

Strings

A string represents a sequence of characters equivalent to an array of `char`. It is declared like this:

```
string[string_length]
```

The specifier *`string_length`* is a number of characters a string consists of. The string is stored internally as the given sequence of characters plus a final `null` character (zero). This appended “stamp” does not count against string’s total length.

A null string (`“”`) is stored as a single `null` character.

You can assign string literals or other strings to string variables. The string on the right side of an assignment operator has to be shorter than another one, or of equal length. For example:

```
dim msg1 as string[20]
dim msg2 as string[19]

main:
  msg1 = “This is some message”
  msg2 = “Yet another message”

  msg1 = msg2   ‘ this is ok, but vice versa would be illegal
```

Alternately, you can handle strings element–by–element. For example:

```
dim s as string[5]
‘ ...
s = “mik”
‘ s[0] is char literal “m”
‘ s[1] is char literal “i”
‘ s[2] is char literal “k”
‘ s[3] is zero
‘ s[4] is undefined
‘ s[5] is undefined
```

Be careful when handling strings in this way, since overwriting the end of a string will cause an unpredictable behavior.

Array of string is declared in this manner:

```
typedef str as string[5]    ‘ first, declare str as a string of 5 elements

dim buffer as str[5]        ‘ now, declare buffer as a array of str elements
```

String Concatenating

mikroBasic PRO for dsPIC30/33 and PIC24 allows you to concatenate strings by means of plus operator. This kind of concatenation is applicable to string variables/literals, character variables/literals. For control characters, use the non-quoted hash sign and a numeral (e.g. `#13` for CR).

Here is an example:

```
dim msg as string[20]
    res_txt as string[5]
    res, channel as word

main:
  '...

  ' Get result of ADC
  res = Adc_Read(channel)

  ' Create string out of numeric result
  WordToStr(res, res_txt)

  ' Prepare message for output
  msg = "Result is " +        ' Text "Result is"
          res_txt             ' Result of ADC

'...
```

Notes:

- In current version plus operator for concatenating strings will accept at most two operands.
- mikroBasic PRO for dsPIC30/33 and PIC24 includes a String Library which automatizes string related tasks.

Pointers

A pointer is a data type which holds a memory address. While a variable accesses that memory address directly, a pointer can be thought of as a reference to that memory address.

To declare a pointer data type, add a carat prefix (`^`) before type. For example, in order to create a pointer to an `integer`, write:

```
^integer
```

In order to access data at the pointer's memory location, add a carat after the variable name. For example, let's declare variable `p` which points to a `word`, and then assign value 5 to the pointed memory location:

```
dim p as ^word
'...
p^ = 5
```

A pointer can be assigned to another pointer. However, note that only the address, not the value, is copied. Once you modify the data located at one pointer, the other pointer, when dereferenced, also yields modified data.

Pointers and memory spaces

Pointers can point to data in any available memory space.
Pointers can reside in any available memory space except in program (code) memory space.

```
dim ptr1 as ^const byte  ' ptr1 pointer in data space pointing to a byte in code space
dim ptr2 as ^const ^volatile sfr byte rx ' ptr2 is pointer in rx space pointing to a
pointer in code space pointing to volatile byte in sfr space
dim ptr3 as ^data byte code   ' error, pointers can not be placed in code space
```

Due to backward compatibility, pointers to program memory space can also be declared within constant declaration block (using keyword `const`):

```
program const_ptr

' constant array will be stored in program memory
const b_array as byte[5] = (1,2,3,4,5)

const ptr as ^byte      ' ptr is pointer to program memory space

main:
  ptr   = @b_array      ' ptr now points to b_array[0]
  PORTA = ptr^
  ptr   = ptr + 3       ' ptr now points to b_array[3]
  PORTA = ptr
end.
```

This leads to equality of the following declarations:

```
dim   ptr1 as ^const byte  ' ptr1 pointer in data space pointing to a byte in code
space
const ptr1 as ^byte ' ptr1 pointer in data space pointing to a byte in code space
```

Therefore, when declaring a pointer within constant declaration block, const qualifier refers to pointed object, not to pointer itself.

Notes:

- Pointer to constant space (Flash memory) is allocated in RAM.
- Constants of a simple type are not allocated in the Flash memory nor in RAM, but changed in the compile time, and therefore address of a such constant can not be obtained.

Function Pointers

Function pointers are allowed in mikroBasic PRO for dsPIC30/33 and PIC24. The example shows how to define and use a function pointer:

Example:

Example demonstrates the usage of function pointers. It is shown how to declare a procedural type, a pointer to function and finally how to call a function via pointer.

```
program Example;

typedef TMyFunctionType = sub function (dim param1, param2 as byte, dim param3 as word) 
as word ' First, define the procedural type

dim MyPtr as ^TMyFunctionType  ' This is a pointer to previously defined type
dim sample as word

sub function Func1(dim p1, p2 as byte, dim p3 as word) as word ' Now, define few functions 
which will be pointed to. Make sure that parameters match the type definition
  result = p1 and p2 or p3
end sub

sub function Func2(dim abc, def as byte, dim ghi as word) as word  ' Another function of 
the same kind. Make sure that parameters match the type definition
  result =  abc * def + ghi
end sub

sub function Func3(dim first, yellow as byte, dim monday as word) as word  ' Yet another 
function. Make sure that parameters match the type definition
  result =  monday - yellow - first
end sub

' main program:
main:
  MyPtr = @Func1              ' MyPtr now points to Func1
  Sample = MyPtr^(1, 2, 3)   ' Perform function call via pointer, call Func1, the return 
value is 3
  MyPtr = @Func2              ' MyPtr now points to Func2
  Sample = MyPtr^(1, 2, 3)   ' Perform function call via pointer, call Func2, the return 
value is 5
  MyPtr = @Func3              ' MyPtr now points to Func3
  Sample = MyPtr^(1, 2, 3)   ' Perform function call via pointer, call Func3, the return 
value is 0
end.
```

@ Operator

The `@` operator constructs a pointer to its operand. The following rules are applied to `@`:

- If `X` is a variable, `@X` returns a pointer to `X`.

Note: If variable `X` is of array type, the `@` operator will return pointer to it's first basic element, except when the left side of the statement in which `X` is used is an array pointer.

In this case, the `@` operator will return pointer to array, not to it's first basic element.

```
program example

dim w  as word
    ptr_b   as ^byte
    ptr_arr as ^byte[10]
    arr     as  byte[10]

main:
  ptr_b   = @arr ' @ operator will return ^byte
  w       = @arr ' @ operator will return ^byte
  ptr_arr = @arr ' @ operator will return ^byte[10]
end.
```

If `F` is a routine (a function or procedure), `@F` returns a pointer to `F`.

Related topics: Pointer Arithmetic

Pointer Arithmetic

Pointer arithmetic in the mikroBasic PRO for dsPIC30/33 and PIC24 is limited to:

- assigning one pointer to another,
- comparing two pointers,
- comparing pointer to zero,
- adding/subtracting pointer and an integer value,
- subtracting two pointers.

Assignment and Comparison

The simple assignment operator (=) can be used to assign value of one pointer to another if they are of the same type.

Assigning the integer constant 0 to a pointer assigns a null pointer value to it.

Two pointers pointing to the same array may be compared by using relational operators `=, <>, <, <=, >,` and `>=`. Results of these operations are the same as if they were used on subscript values of array elements in question:

```
dim ptr1 as ^byte
    ptr2 as ^byte
    a as byte[10]  ' array a containing 10 elements of type byte

main:
  ptr1 = @a[4]
  ptr2 = @a[2]

  if (ptr1 = ptr2) then ...   ' won't be executed as 4 is not equal to 2
  if (ptr1 > ptr2) then ...   ' will be executed as 4 is greater than 2

  if (ptr1^ = ptr2^) then ... ' if the value pointed to by ptr1 is equal to the value 
pointed to by ptr2 ...
  if (ptr1^ > ptr2^) then ... ' if the value pointed to by ptr1 is greater to the value 
pointed to by ptr2 ...
end.
```

Note: Comparing pointers pointing to different objects/arrays can be performed at programmer's own responsibility — a precise overview of data's physical storage is required.

Pointer Addition

You can use `Inc` to add an integral value to a pointer. The result of addition is defined only if the pointer points to an element of an array *and* if the result is a pointer pointing to the same array (or one element beyond it).

If a pointer is declared to point to `type`, adding an integral value n to the pointer increments the pointer value by `n * sizeof(type)` as long as the pointer remains within the legal range (first element to one beyond the last element). If `type` has a size of 10 bytes, then adding 5 to a pointer to `type` advances the pointer 50 bytes in memory.

For example:

```
dim
  a as byte[10]       ' array a containing 10 elements of type byte
  ptr as ^byte        ' pointer to byte

main:
  ptr = @a[0]         ' ptr is pointer to byte, pointing to a[0]
  ptr = ptr + 3       ' ptr+3 is a pointer pointing to a[3]
  ptr^ = 6            ' a[3] now equals 6
  Inc(ptr)            ' ptr now points to the next element of array a: a[4]
end.
```

Also, you may sum values pointed to by pointers.

For example:

```
dim
  i, j, x as byte   ' variables
  ptr1 as ^byte     ' pointers to byte
  ptr2 as ^byte

main
  i = 10          ' assign value 10 to variable; i is at the address 0x0038
  j = 5           ' assign value 10 to variable; j is at the address 0x003A

  ptr1 = @i       ' ptr1 is pointer to byte, pointing to i
  ptr2 = @j       ' ptr2 is a pointer pointing to j

  x = ptr1^ + ptr2^    ' result is equal to the sum of the values pointed to; x = 5
end.
```

Pointer Subtraction

Similar to addition, you can use `Dec` to subtract an integral value from a pointer.

If a pointer is declared to point to `type`, subtracting an integral value `n` from the the pointer decrements the pointer value by `n * sizeof(type)` as long as the pointer remains within the legal range (first element to one beyond the last element). If `type` has a size of 10 bytes, then subtracting 5 from a pointer to `type` pushes back the pointer 50 bytes in memory.

For example:

```
dim
  a as byte[10]       ' array a containing 10 elements of type byte
  ptr as ^byte        ' pointer to byte

main:
  ptr = @a[6]         ' ptr is pointer to byte, pointing to a[6]
  ptr = ptr - 3       ' ptr-3 is a pointer pointing to a[3]
  ptr^ = 6             ' a[3] now equals 6
  Dec(ptr)            ' ptr now points to the previous element of array a: a[2]
end.
```

Also, you may subtract two pointers. The difference will be equal to the distance between two pointed addresses, and is calculated regarding to the type which the pointer points to.

For example:

```
dim
  i, j, x as byte   ' variables
  ptr1 as ^byte     ' pointers to byte
  ptr2 as ^byte

main:
  i = 10        ' assign value 10 to variable; i is at the address 0x0039
  j = 5         ' assign value 5 to variable;  j is at the address 0x003A

  ptr1 = @i     ' ptr1 is pointer to byte, pointing to i
  ptr2 = @j     ' ptr2 is a pointer pointing to j

  x = ptr2 - ptr1   ' result is equal to the distance between the two pointed addresses;
x = 1 (1 byte)
  x = ptr1^ - ptr2^   ' result is equal to the difference of the values pointed to; x = 5
end.
```

Structures

A structure represents a heterogeneous set of elements. Each element is called a *member*; the declaration of a structure type specifies a name and type for each member. The syntax of a structure type declaration is

```
structure structname
  dim member1 as type1
  '...
  dim membern as typen
end structure
```

where `structname` is a valid identifier, each `type` denotes a type, and each `member` is a valid identifier. The scope of a member identifier is limited to the structure in which it occurs, so you don't have to worry about naming conflicts between member identifiers and other variables.

For example, the following declaration creates a structure type called `Dot`:

```
structure Dot
  dim x as float
  dim y as float
end structure
```

Each `Dot` contains two members: `x` and `y` coordinates; memory is allocated when you instantiate the structure, like this:

```
dim m, n as Dot
```

This variable declaration creates two instances of `Dot`, called `m` and `n`.

A member can be of the previously defined structure type. For example:

```
' Structure defining a circle:
structure Circle
  dim radius as float
  dim center as Dot
end structure
```

Structure Member Access

You can access the members of a structure by means of dot (`.`) as a direct member selector. If we had declared the variables `circle1` and `circle2` of the previously defined type `Circle`:

```
dim circle1, circle2 as Circle
```

we could access their individual members like this:

```
circle1.radius = 3.7
circle1.center.x = 0
circle1.center.y = 0
```

You can also commit assignments between complex variables, if they are of the same type:

```
circle2 = circle1  ' This will copy values of all members
```

Types Conversions

Conversion of variable of one type to a variable of another type is typecasting. mikroBasic PRO for dsPIC30/33 and PIC24 supports both implicit and explicit conversions for built-in types.

Implicit Conversion

Compiler will provide an automatic implicit conversion in the following situations:

- statement requires an expression of particular type (according to language definition) and we use an expression of different type,
- operator requires an operand of particular type and we use an operand of different type,
- function requires a formal parameter of particular type and we pass it an object of different type,
- `result` does not match the declared function return type.

Promotion

When operands are of different types, implicit conversion promotes the less complex type to more complex type taking the following steps:

```
bit       →  byte/char
byte/char →  word
short     →  integer
short     →  longint
integer   →  longint
integral  →  float
```

Higher bytes of extended unsigned operand are filled with zeroes. Higher bytes of extended signed operand are filled with bit sign (if number is negative, fill higher bytes with one, otherwise with zeroes). For example:

```
dim a as byte
dim b as word
'...
a = $FF
b = a   ' a is promoted to word, b becomes $00FF
```

Clipping

In assignments and statements that require an expression of particular type, destination will store the correct value only if it can properly represent the result of expression, i.e. if the result fits in destination range.

If expression evaluates to a more complex type than expected, excess of data will be simply clipped (higher bytes are lost).

```
dim i as byte
dim j as word
'...
j = $FF0F
i = j   ' i becomes $0F, higher byte $FF is lost
```

Explicit Conversion

Explicit conversion can be executed at any point by inserting type keyword (`byte, word, short, integer, longint,` or `float`) ahead of the expression to be converted. The expression must be enclosed in parentheses. Explicit conversion can be performed only on the operand left of the assignment operator.

Special case is the conversion between signed and unsigned types. Explicit conversion between signed and unsigned data does not change binary representation of data — it merely allows copying of source to destination.

For example:

```
dim a as byte
dim b as short
'...
b = -1
a = byte(b)   ' a is 255, not 1

' This is because binary representation remains
' 11111111; it's just interpreted differently now
```

You can't execute explicit conversion on the operand left of the assignment operator:

```
word(b) = a    ' Compiler will report an error
```

Conversions Examples

Here is an example of conversion:

```
program test

typedef TBytePtr as ^byte

dim arr as word[10]
    ptr as TBytePtr

dim a, b, cc as byte
dim dd as word

main:
  a = 241
  b = 128

  cc  = a + b         ' equals 113
  cc  = word(a + b)   ' equals 113
  dd  = a + b         ' equals 369

  ptr = TBytePtr(@arr)
  ptr = ^byte(@arr)
end.
```

Typedef Specifier

The specifier `typedef` introduces a synonym for a specified type. The `typedef` declarations are used to construct shorter or more convenient names for types already defined by the language or declared by the user.

The specifier `typedef` stands first in the declaration:

```
typedef synonym as <type_definition>
```

The `typedef` keyword assigns *synonym* to *<type_definition>*. The *synonym* needs to be a valid identifier.

A declaration starting with the `typedef` specifier does not introduce an object or a function of a given type, but rather a new name for a given type. In other words, the `typedef` declaration is identical to a “normal” declaration, but instead of objects, it declares types. It is a common practice to name custom type identifiers with starting capital letter — this is not required by the mikroBasic PRO for dsPIC.

For example:

```
' Let's declare a synonym for "word"
typedef Distance as word

' Now, synonym "Distance" can be used as type identifier:
dim i as Distance ' declare variable i of word
```

In the `typedef` declaration, as in any other declaration, several types can be declared at once. For example:

```
typedef ^Pti, Array[10] as byte
```

Here, `Pti` is a synonym for type “pointer to `int`”, and `Array` is a synonym for type “array of 10 `byte` elements”.

Type Qualifiers

The type qualifiers `const` and `volatile` are optional in declarations and do not actually affect the type of declared object.

Qualifier const

The qualifier `const` implies that a declared object will not change its value during runtime. In declarations with the `const` qualifier all objects need to be initialized.

The mikroBasic PRO for dsPIC30/33 and PIC24 treats objects declared with the `const` qualifier the same as literals or preprocessor constants. If the user tries to change an object declared with the `const` qualifier compiler will report an error.

For example:

```
const PI as byte = 3.14159
```

Qualifier volatile

The qualifier `volatile` implies that a variable may change its value during runtime independently from the program. Use the volatile modifier to indicate that a variable can be changed by a background routine, an interrupt routine, or I/O port. Declaring an object to be volatile warns the compiler not to make assumptions concerning the value of an object while evaluating expressions in which it occurs because the value could be changed at any moment.

Operators

Operators are tokens that trigger some computation when being applied to variables and other objects in an expression.

There are four types of operators in mikroBasic PRO for dsPIC30/33 and PIC24:

- Arithmetic Operators
- Bitwise Operators
- Boolean Operators
- Relational Operators

Operators Precedence and Associativity

There are 4 precedence categories in mikroBasic PRO for dsPIC30/33 and PIC24. Operators in the same category have equal precedence with each other.

Each category has an associativity rule: left-to-right (→), or right-to-left (←). In the absence of parentheses, these rules resolve the grouping of expressions with operators of equal precedence.

Precedence	Operands	Operators	Associativity
4	1	`@ not + -`	←
3	2	`* / div mod and << >>`	→
2	2	`+ - or xor`	→
1	2	`= <> < > <= >=`	→

Arithmetic Operators

Arithmetic operators are used to perform mathematical computations. They have numerical operands and return numerical results. Since the `char` operators are technically `bytes`, they can be also used as unsigned operands in arithmetic operations.

All arithmetic operators associate from left to right.

Arithmetic Operators Overview

Operator	Operation	Operands	Result
`+`	addition	`byte, short, word, integer, longint, longword, float`	`byte, short, word, integer, longint, longword, float`
`-`	subtraction	`byte, short, word, integer, longint, longword, float`	`byte, short, word, integer, longint, longword, float`
`*`	multiplication	`byte, short, word, integer, longint, longword, float`	`word, integer, longint, longword, float`
`/`	division, floating-point	`byte, short, word, integer, longint, longword, float`	`float`
`div`	division, rounds down to nearest integer	`byte, short, word, integer, longint, longword`	`byte, short, word, integer, longint, longword`
`mod`	modulus, returns the remainder of integer division (cannot be used with floating points)	`byte, short, word, integer, longint, longword`	`byte, short, word, integer, longint, longword`

Division by Zero

If 0 (zero) is used explicitly as the second operand (i.e. `x div 0`), the compiler will report an error and will not generate code.
But in case of implicit division by zero: `x div y`, where `y` is 0 (zero), the result will be the maximum integer (i.e `255`, if the result is `byte` type; `65536`, if the result is `word` type, etc.).

Unary Arithmetic Operators

Operator `-` can be used as a prefix unary operator to change sign of a signed value. Unary prefix operator `+` can be used, but it doesn't affect data.

For example:

```
b = -a
```

Relational Operators

Use relational operators to test equality or inequality of expressions. All relational operators return TRUE or FALSE.

All relational operators associate from left to right.

Relational Operators Overview

Operator	Operation
=	equal
<>	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal

Relational Operators in Expressions

The equal sign (`=`) can also be an assignment operator, depending on context.

Precedence of arithmetic and relational operators was designated in such a way to allow complex expressions without parentheses to have expected meaning:

```
if aa + 5 >= bb - 1.0 / cc then    ' same as: if (aa + 5) >= (bb - (1.0 / cc)) then
  dd = My_Function()
end if
```

Bitwise Operators

Use bitwise operators to modify individual bits of numerical operands.

Bitwise operators associate from left to right. The only exception is the bitwise complement operator `not` which associates from right to left.

Bitwise Operators Overview

Operator	Operation
`and`	bitwise AND; compares pairs of bits and returns 1 if both bits are 1, otherwise it returns 0
`or`	bitwise (inclusive) OR; compares pairs of bits and generates a 1 result if either or both bits are 1, otherwise it returns 0
`xor`	bitwise exclusive OR (XOR); compares pairs of bits and generates a 1 result if the bits are complementary, otherwise it returns 0
`not`	bitwise complement (unary); inverts each bit
`<<`	bitwise shift left; moves the bits to the left, discards the far left bit and assigns 0 to the right most bit.
`>>`	bitwise shift right; moves the bits to the right, discards the far right bit and if unsigned assigns 0 to the left most bit, otherwise sign extends

Logical Operations on Bit Level

and	**0**	**1**
0	0	0
1	0	1

or	**0**	**1**
0	0	1
1	1	1

xor	**0**	**1**
0	0	1
1	1	0

not	**0**	**1**
	1	0

The bitwise operators `and,` `or,` and `xor` perform logical operations on the appropriate pairs of bits of their operands. The operator `not` complements each bit of its operand. For example:

```
$1234 and $5678          ' equals $1230

' because ..

' $1234 : 0001 0010 0011 0100
' $5678 : 0101 0110 0111 1000
' ----------------------------
'   and : 0001 0010 0011 0000

' .. that is, $1230

' Similarly:

$1234 or  $5678          ' equals $567C
$1234 xor $5678          ' equals $444C
not $1234                ' equals $EDCB
```

Unsigned and Conversions

If a number is converted from less complex to more complex data type, the upper bytes are filled with zeroes. If a number is converted from more complex to less complex data type, the data is simply truncated (the upper bytes are lost).

For example:

```
dim a as byte
dim b as word
' ...
  a = $AA
  b = $F0F0
  b = b and a
   ' a is extended with zeroes; b becomes $00A0
```

Signed and Conversions

If number is converted from less complex to more complex data type, the upper bytes are filled with ones if sign bit is 1 (number is negative); the upper bytes are filled with zeroes if sign bit is 0 (number is positive). If number is converted from more complex to less complex data type, the data is simply truncated (the upper bytes are lost).

For example:

```
dim a as byte
dim b as word
' ...
  a = -12
  b = $70FF
  b = b and a

  ' a is sign extended, upper byte is $FF;
  ' b becomes $70F4
```

Bitwise Shift Operators

The binary operators `<<` and `>>` move the bits of the left operand by a number of positions specified by the right operand, to the left or right, respectively. Right operand has to be positive and less than 255.

With shift left (`<<`), left most bits are discarded, and "new" bits on the right are assigned zeroes. Thus, shifting unsigned operand to the left by *n* positions is equivalent to multiplying it by 2^n if all discarded bits are zero. This is also true for signed operands if all discarded bits are equal to the sign bit.

With shift right (`>>`), right most bits are discarded, and the "freed" bits on the left are assigned zeroes (in case of unsigned operand) or the value of the sign bit (in case of signed operand). Shifting operand to the right by *n* positions is equivalent to dividing it by 2^n.

Boolean Operators

Although mikroBasic PRO for dsPIC30/33 and PIC24 does not support `boolean` type, you have Boolean operators at your disposal for building complex conditional expressions. These operators conform to standard Boolean logic, and return either `TRUE` (all ones) or `FALSE` (zero):

Operator	Operation
`and`	logical AND
`or`	logical OR
`xor`	logical exclusive OR (XOR)
`not`	logical negation

Boolean operators associate from left to right. Negation operator `not` associates from right to left.

Unary Operators

Unary operators are operators that take exactly one argument.

Unary Arithmetic Operator

Operator `-` can be used as a prefix unary operator to change sign of a signed value. Unary prefix operator `+` can be used also, but it doesn't affect data.

For example:

```
b = -a
```

Unary Bitwise Operator

The result of the `not` (bitwise negation) operator is the bitwise complement of the operand. In the binary representation of the result, every bit has the opposite value of the same bit in the binary representation of the operand.

Operator	Operation
`not`	bitwise complement (unary); inverts each bit

Example:

```
not 0x1234                 ' equals 0xEDCB
```

Address and Indirection Operator

In the mikroBasic PRO for dsPIC, address of an object in memory can be obtained by means of an unary operator `@`. To reach the pointed object, we use an indirection operator `^` on a pointer. See Pointers section for more details.

Operator	Operation
`^`	accesses a value indirectly, through a pointer; result is the value at the address to which operand points
`@`	constructs a pointer to its operand

See Pointers for more details on this subject

Note: Besides these, sizeof and explicit conversion unary operators are supported also.

Sizeof Operator

The prefix unary operator `sizeof` returns an integer constant that represents the size of memory space (in bytes) used by its operand (determined by its type, with some exceptions).

The operator `sizeof` can take either a type identifier or an unary expression as an operand. You *cannot* use `sizeof` with expressions of function type, incomplete types, parenthesized names of such types, or with lvalue that designates a bit field object.

Sizeof Applied to Expression

If applied to expression, the size of an operand is determined without evaluating the expression (and therefore without side effects). The result of the operation will be the size of the type of the expression's result.

Sizeof Applied to Type

If applied to a type identifier, `sizeof` returns the size of the specified type. The unit for type size is `sizeof(byte)` which is equivalent to one byte.

Thus:

```
sizeof(byte)             ' returns 1
sizeof(integer)          ' returns 2
sizeof(longword)         ' returns 4
sizeof(float)            ' returns 4
```

When the operand is a non-parameter of array type, the result is the total number of bytes in the array (in other words, an array name is not converted to a pointer type):

```
dim  i, j as integer
     samples as integer[7]
...
j = sizeof(samples[1])   ' j = sizeof(integer) = 2
i = sizeof(samples)      ' i = 10*sizeof(integer) = 20
```

If the operand is a parameter declared as array type or function type, `sizeof` gives the size of the pointer. When applied to structures, `sizeof` gives the total number of bytes, including any padding. The operator `sizeof` cannot be applied to a function.

Expressions

An expression is a sequence of operators, operands and punctuators that returns a value.

The *primary expressions* include: literals, constants, variables and function calls. From them, using operators, more complex expressions can be created. Formally, expressions are defined recursively: subexpressions can be nested up to the limits of memory.

Expressions are evaluated according to certain conversion, grouping, associativity and precedence rules which depend on the operators in use, presence of parentheses and data types of the operands. The precedence and associativity of the operators are summarized in Operator Precedence and Associativity. The way operands and subexpressions are grouped does not necessarily specify the actual order in which they are evaluated by mikroBasic PRO for dsPIC30/33 and PIC24.

Expression Evaluation

General Rule

Expression are evaluated according to the right side operands. Operations are done at higher operand level, with signed operands taking precedence.

Example:

```
a as byte
b as word
c as integer

a * b ' word level
a * c ' integer level
b * c ' integer level
```

Left side exception

In arithmetic expression left side is considered in the following manner: If the left side size in bytes is greater than higher operand size, then evaluation is done at one level above higher operand level (to get correct calculations).

Example:

```
a as longword
b as byte

a = b * 5 ' this is done at word level
```

Conditional expressions

Conditional expressions may differ from the same code in assignment expressions (due to left side exception).

Example:

```
a as longword
b as byte

if b*5 then... ' byte level - general rule will not give same result as

a = b * 5      ' word level - general rule + left side exception
if a then...

if b*5 exceeds byte range.
```

Explicit Typecasting

Any expression can be evaluated at specific level by using explicit typecasting. Having in mind previous example, in order to get same calculation in conditional and assignment expression, the following should be done:

```
if word(b*5) then... ' word level
```

Statements

Statements define algorithmic actions within a program. Each statement needs to be terminated with a semicolon (;). In the absence of specific jump and selection statements, statements are executed sequentially in the order of appearance in the source code.

The most simple statements are assignments, procedure calls and jump statements. These can be combined to form loops, branches and other structured statements.

Refer to:

- Assignment Statements
- Conditional Statements
- Iteration Statements (Loops)
- Jump Statements

- asm Statement

Assignment Statements

Assignment statements have the following form:

```
variable = expression
```

The statement evaluates `expression` and assigns its value to `variable`. All the rules of implicit conversion are applied. `Variable` can be any declared variable or array element, and `expression` can be any expression.

Do not confuse the assignment with relational operator = which tests for equality. mikroBasic PRO for dsPIC30/33 and PIC24 will interpret the meaning of the character = from the context.

Conditional Statements

Selection or flow-control statements select one of alternative courses of action by testing certain values. There are two types of selection statements:

- if
- select case

If Statement

Use the keyword `if` to implement a conditional statement. The syntax of the `if` statement has the following form:

```
if expression then
  statements
[else
  other statements]
end if
```

When `expression` evaluates to true, `statements` execute. If expression is false, `other statements` execute. The `expression` must convert to a boolean type; otherwise, the condition is ill-formed. The `else` keyword with an alternate block of statements (`other statements`) is optional.

Nested if statements

Nested if statements require additional attention. A general rule is that the nested conditionals are parsed starting from the innermost conditional, with each `else` bound to the nearest available if on its left:

```
if expression1 then
if expression2 then
statement1
else
statement2
end if
end if
```

The compiler treats the construction in this way:

```
if expression1 then
  if expression2 then
    statement1
  else
    statement2
  end if
end if
```

In order to force the compiler to interpret our example the other way around, we have to write it explicitly:

```
if expression1 then
  if expression2 then
    statement1
  end if
else
  statement2
end if
```

Select Case Statement

Use the `select case` statement to pass control to a specific program branch, based on a certain condition. The `select case` statement consists of selector expression (condition) and list of possible values. The syntax of the `select case` statement is:

```
select case selector
  case value_1
    statements_1
  ...
  case value_n
    statements_n
  [case else
    default_statements]
end select
```

`selector` is an expression which should evaluate as integral value. `values` can be literals, constants or expressions and `statements` can be any statements. The `case else` clause is optional.

First, the `selector` expression (condition) is evaluated. The `select case` statement then compares it against all available `values`. If the match is found, the `statements` following the match evaluate, and the `select case` statement terminates. In case there are multiple matches, the first matching statement will be executed. If none of the `values` matches the `selector`, then `default_statements` in the `case else` clause (if there is one) are executed.

Here is a simple example of the `select case` statement:

```
select case operator
  case "*"
    res = n1 * n2
  case "/"
    res = n1 / n2
  case "+"
    res = n1 + n2
  case "-"
    res = n1 - n2
  case else
    res = 0
    cnt = cnt + 1
end select
```

Also, you can group values together for a match. Simply separate the items by commas:

```
select case reg
  case 0
    opmode = 0
  case 1,2,3,4
    opmode = 1
  case 5,6,7
    opmode = 2
end select
```

Nested Case Statements

Note that the `select case` statements can be nested – *`values`* are then assigned to the innermost enclosing `select case` statement.

Iteration Statements

Iteration statements let you loop a set of statements. There are three forms of iteration statements in mikroBasic PRO for dsPIC30/33 and PIC24:

- for
- while
- do

You can use the statements break and continue to control the flow of a loop statement. `break` terminates the statement in which it occurs, while `continue` begins executing the next iteration of the sequence.

For Statement

The `for` statement implements an iterative loop and requires you to specify the number of iterations. The syntax of the `for` statement is:

```
for counter = initial_value to final_value [step step_value]
  statement_list
next counter
```

`counter` is a variable which increments with each iteration of the loop. Before the first iteration, `counter` is set to `initial_value` and will increment until it reaches `final_value`. `final_value` will be recalculated each time the loop is reentered.
This way number of loop iterations can be changed inside the loop by changing `final_value`. With each iteration, `statement_list` will be executed.

`initial_value` and `final_value` should be expressions compatible with `counter`; `statement_list` may be consisted of statements that don't change the value of the `counter`.

Note that the parameter `step_value` may be negative, allowing you to create a countdown.

If `final_value` is a complex expression whose value can not be calculated in compile time and number of loop iterations is not to be changed inside the loop by the means of `final_value`, it should be calculated outside the for statement and result should be passed as for statement's `final_value`. `statement_list` is a list of statements that do not change the value of counter.

Here is an example of calculating scalar product of two vectors, `a` and `b`, of length `10`, using the `for` statement:

```
s = 0
for i = 0 to 9
  s = s + a[i] * b[i]
next i
```

Endless Loop

The `for` statement results in an endless loop if `final_value` equals or exceeds the range of the `counter`'s type.

While Statement

Use the `while` keyword to conditionally iterate a statement. The syntax of the `while` statement is:

```
while expression
  statements
wend
```

`statements` are executed repeatedly as long as `expression` evaluates true. The test takes place before `statements` are executed. Thus, if `expression` evaluates false on the first pass, the loop does not execute.

Here is an example of calculating scalar product of two vectors, using the `while` statement:

```
s = 0
i = 0
while i < n
  s = s + a[i] * b[i]
  i = i + 1
wend
```

Probably the easiest way to create an endless loop is to use the statement:

```
while TRUE
  ' ...
wend
```

Do Statement

The `do` statement executes until the condition becomes true. The syntax of the `do` statement is:

```
do
  statements
loop until expression
```

`statements` are executed repeatedly until *`expression`* evaluates true. *`expression`* is evaluated *after* each iteration, so the loop will execute *`statements`* at least once.

Here is an example of calculating scalar product of two vectors, using the `do` statement:

```
s = 0
i = 0
do
  s = s + a[i] * b[i]
  i = i + 1
loop until i = n
```

Jump Statements

The jump statement, when executed, transfers control unconditionally. There are five such statements in mikroBasic PRO for dsPIC30/33 and PIC24:

- break
- continue
- exit
- goto
- gosub

Break and Continue Statements

Break Statement

Sometimes, you might need to stop the loop from within its body. Use the `break` statement within loops to pass control to the first statement following the innermost loop (`for, while,` or `do`).

For example:

```
Lcd_Out(1, 1, “No card inserted”)

' Wait for CF card to be plugged; refresh every second
while true
  if Cf_Detect() = 1 then
    break
  end if
  Delay_ms(1000)
wend

' Now we can work with CF card ...
Lcd_Out(1, 1, “Card detected   “)
```

Continue Statement

You can use the `continue` statement within loops to “skip the cycle”:

- `continue` statement in the `for` loop moves program counter to the line with keyword `for` after incrementing the counter,
- `continue` statement in the `while` loop moves program counter to the line with loop condition (top of the loop),
- `continue` statement in the `do` loop moves program counter to the line with loop condition (bottom of the loop).

```
` continue jumps here
for i = ...
  ...
  continue
  ...
next i
```

```
` continue jumps here
while condition
  ...
  continue
  ...
wend
```

```
do
  ...
  continue
  ...
` continue jumps here
loop until condition
```

Exit Statement

The `exit` statement allows you to break out of a routine (function or procedure). It passes the control to the first statement following the routine call.

Here is a simple example:

```
sub procedure Proc1()
dim error as byte
  ... ' we're doing something here
  if error = TRUE then
    exit
  end if
  ... ' some code, which won't be executed if error is true
end sub
```

Note: If breaking out of a function, return value will be the value of the local variable `result` at the moment of exit.

Return Statement

The return statement causes execution to leave the current subroutine and resume at the point in the code immediately after where the subroutine was called. It's mainly intended to be used with gosub statement.
Return statement suffers from the same sort of readability problems as the GOTO statement and like goto, the use of return statement is generally discouraged.

Here is a simple example:

```
sub procedure Proc1()
dim error as byte
  ... ' we're doing something here
  if error = TRUE then
    return
  end if
  ... ' some code, which won't be executed if error is true
end sub
```

Note: Return statements performs the same as exit statement except in functions. If breaking out of a function with return statement, return value will not be specified. In such cases exit statement should be used.

Goto Statement

Use the `goto` statement to unconditionally jump to a local label — for more information, refer to Labels. The syntax of the `goto` statement is:

```
goto label_name
```

This will transfer control to the location of a local label specified by *`label_name`*. The `goto` line can come before or after the label.

Label and `goto` statement must belong to the same block. Hence it is not possible to jump *into* or out of a procedure or function.

You can use `goto` to break out from any level of nested control structures. Never jump *into* a loop or other structured statement, since this can have unpredictable effects.

The use of `goto` statement is generally discouraged as practically every algorithm can be realized without it, resulting in legible structured programs. One possible application of the `goto` statement is breaking out from deeply nested control structures:

```
for i = 0 to n
  for j = 0 to m
    ...
    if disaster
      goto Error
    end if
    ...
  next j
next i
 .
 .
 .
Error: ' error handling code
```

Gosub Statement

Use the `gosub` statement to unconditionally jump to a local label — for more information, refer to Labels. The syntax of the `gosub` statement is:

```
gosub label_name
...
label_name:
...
return
```

This will transfer control to the location of a local label specified by *`label_name`*. Also, the calling point is remembered. Upon encountering the `return` statement, program execution will continue with the next statement (line) after `gosub`. The `gosub` line can come before or after the label.

It is not possible to jump into or out of routine by means of `gosub`. Never jump *into* a loop or other structured statement, since this can have unpredictable effects.

Note: Like with `goto`, the use of `gosub` statement is generally discouraged. mikroBasic PRO for dsPIC30/33 and PIC24 supports `gosub` only for the sake of backward compatibility. It is better to rely on functions and procedures, creating legible structured programs.

asm Statement

mikroBasic PRO for dsPIC30/33 and PIC24 allows embedding assembly in the source code by means of the `asm` statement. Note that you cannot use numerals as absolute addresses for register variables in assembly instructions. You may use symbolic names instead (listing will display these names as well as addresses).

You can group assembly instructions with the `asm` keyword:

```
asm
   block of assembly instructions
end asm
```

The only types whose name remains the same in asm as it is in the mikroBasic PRO for dsPIC30/33 and PIC24 are registers, e.g. INTCON, PORTB, WREG, GIE, etc.

mikroBasic PRO for dsPIC30/33 and PIC24 comments are allowed in embedded assembly code.

Accessing variables

Depending on the place of declaration, accessing a variable can be done in several ways :

- **Accessing global variable**:
  1. If declared under implementation section (visible only in the file where it was declared): `<source_file_name>_<variable_name>.`
  2. If declared in the interface section (visible throughout the whole project): `_<variable_name>`.
  3. If accessing registers (declared through `register,` `rx` or `sfr` specifiers, visible throughout the whole project): `<variable_name>`.
- **Accessing local variable**: `<routine_name>_<variable_name>.`
- **Accessing routine parameter**: `FARG_<routine_name>_<variable_name>.`

Here is an example of using asm instructions:

```
program asm_example

dim myvar as word absolute 0x2678
dim myvar1 as longword
const msg = "Hello" org 0x1234

sub procedure proc() org 0x2346
  asm
    nop
  end asm
end sub

main :
  myvar = 5
  myvar1 = 0xABCD1234
```

```
asm
   MOV _myvar, w0                      ; move myvar to W0
   nop
   MOV #6, W0                          ; move literal 6 to W0
   MOV W0, _myvar                      ; move contents of W0 to myvar
   MOV #lo_addr(_myvar), w1  ; retrieve low address word of _myvar and move it to W1 (0x2678 -> W1)
   MOV #hi_addr(_myvar), W1 ; retrieve high address word of _myvar and move it to W1 (0x0000 -> W1)
   MOV #lo_addr(_proc), W0 ; retrieve hi address byte of routine proc and move it to W0 (0x0001 -> W1)
   MOV #lo_addr(_msg), W0 ; retrieve low address word of constant msg and move it to W0 (0x3652 -> W1)
   MOV _myvar1+2, w0 ; accessing hi word of myvar1 variable and move it to W1 (0xABCD -> W1)
  end asm

end.
```

Asm code and SSA optimization

If asm code is mixed with the Basic code, keep in mind that the generated code can substantially differ when SSA optimization option is enabled or disabled.
This is due to the fact that SSA optimization uses certain working registers to store routine parameters (W10-W13), rather than storing them onto the function frame.

Because of this, user must be very careful when writing asm code as existing values in the working registers used by SSA optimization can be overwritten.
To avoid this, it is recommended that user includes desired asm code in a separate routine.

Directives

Directives are words of special significance which provide additional functionality regarding compilation and output.

The following directives are at your disposal:

- Compiler directives for conditional compilation,
- Linker directives for object distribution in memory.

Compiler Directives

Any line in source code with leading `#` is taken as a compiler directive. The initial `#` can be preceded or followed by whitespace (excluding new lines). The compiler directives are not case sensitive.

You can use conditional compilation to select particular sections of code to compile while excluding other sections. All compiler directives must be completed in the source file in which they begun.

Directives #DEFINE and #UNDEFINE

Use directive `#DEFINE` to define a conditional compiler constant (“flag”). You can use any identifier for a flag, with no limitations. No conflicts with program identifiers are possible because the flags have a separate name space. Only one flag can be set per directive.

For example:

```
#DEFINE extended_format
```

Use `#UNDEFINE` to undefine (“clear”) previously defined flag.

Directives #IFDEF, #IFNDEF, #ELSE and #ENDIF

Conditional compilation is carried out by the `#IFDEF` and `#IFNDEF` directives. `#IFDEF` tests whether a flag is currently defined, and `#IFNDEF` if the flag is not defined; i.e. whether a previous `#DEFINE` directive has been processed for that flag and is still in force.

Directives `#IFDEF` and `#IFNDEF` are terminated with the `#ENDIF` directive and can have an optional `#ELSE` clause:

```
#IFDEF flag THEN
  block of code
[ #ELSE
  alternate block of code ]
#ENDIF
```

First, `#IFDEF` checks if `flag` is defined by means of `#DEFINE`. If so, only *block of code* will be compiled. Otherwise, *alternate block of code* in `#ELSE` (if any) will be compiled. `#ENDIF` ends the conditional sequence. The result of the preceding scenario is that only one section of code (possibly empty) is passed on for further processing. The processed section can contain further conditional clauses, nested to any depth; each `#IFDEF` must be matched with a closing `#ENDIF`.

Here is an example:

```
' Uncomment the appropriate flag for your application:
'#DEFINE resolution10
'#DEFINE resolution12

#IFDEF resolution10 THEN
  // <code specific to 10-bit resolution>
  #ELSE
    #IFDEF resolution12 THEN
      // <code specific to 12-bit resolution>
    #ELSE
      // <default code>
    #ENDIF
#ENDIF
```

Unlike `#IFDEF,` `#IFNDEF` checks if `flag` is not defined by means of `#DEFINE`, thus producing the opposite results.

Include Directive #I

The `#I` parameter directive instructs mikroBasic PRO for dsPIC30/33 and PIC24 to include the named text file in the compilation. In effect, the file is inserted in the compiled text right after the `#I` filename directive. If `filename` does not specify a directory path, then, in addition to searching for the file in the same directory as the current unit, mikroBasic PRO for dsPIC30/33 and PIC24 will search for file in order specified by the search paths.

To specify a filename that includes a space, surround the file name with quotation marks: `#I “My file”`.

There is one restriction to the use of include files: An include file can’t be specified in the middle of a statement part. In fact, all statements between the begin and end of a statement part must exist in the same source file.

See also Predefined Project Level Defines.

Linker Directives

mikroBasic PRO for dsPIC30/33 and PIC24 uses internal algorithm to distribute objects within memory. If you need to have a variable or routine at the specific predefined address, use the linker directives `absolute` and `org`.

Directive absolute

Directive `absolute` specifies the starting address in RAM for a variable. If the variable is multi-byte, higher bytes will be stored at the consecutive locations.

Directive `absolute` is appended to declaration of a variable:

```
' Variable x will occupy 1 word (16 bits) at address 0x32
dim x as word absolute 0x32

' Variable y will occupy 2 words at addresses 0x34 and 0x36
dim y as longint absolute 0x34
```

Be careful when using `absolute` directive, as you may overlap two variables by accident. For example:

```
dim  i as word absolute 0x42
' Variable i will occupy 1 word at address 0x42;

dim jj as longint absolute 0x40
' Variable will occupy 2 words at 0x40 and 0x42; thus,
' changing i changes jj at the same time and vice versa
```

Directive org

Directive `org` specifies the starting address of a constant or a routine in ROM. It is appended to the constant or a routine declaration.

To place a constant array in Flash memory, write the following:

```
' Constant array MONTHS will be placed starting from the address 0x800
const MONTHS as byte[12] = (31,28,31,30,31,30,31,31,30,31,30,31) org 0x800
```

If you want to place simple type constant into Flash memory, instead of following declaration:

```
const SimpleConstant as byte = 0xAA org 0x2000
```

use an array consisting of single element:

```
const SimpleConstant as byte[1] = (0xAA) org 0x800
```

In first case, compiler will recognize your attempt, but in order to save Flash space, and boost performance, it will automatically replace all instances of this constant in code with it's literal value.
In the second case your constant will be placed in Flash in the exact location specified.

To place a routine on a specific address in Flash memory you should write the following:

```
sub procedure proc(dim par as word) org 0x200
' Procedure will start at the address 0x200;
...
end sub
```

`org` directive can be used with `main` routine too. For example:

```
program Led_Blinking

main: org 0x800                ' main procedure starts at 0x800
...
end
```

Directive orgall

Use the `orgall` directive to specify the address above which all routines and constants will be placed. Example:

```
main:
  orgall(0x200)  ' All the routines, constants in main program will be above the address 
0x200
  ...
end.
```

CHAPTER 9

mikroBasic PRO for dsPIC30/33 and PIC24 Libraries

mikroBasic PRO for dsPIC30/33 and PIC24 provides a set of libraries which simplify the initialization and use of dsPIC30/33 and PIC24 and their modules:

Use Library manager to include mikroBasic PRO for dsPIC30/33 and PIC24 Libraries in you project.

Hardware Libraries

- ADC Library
- CAN Library
- CANSPI Library
- Compact Flash Library
- Enhanced CAN Library
- EEPROM Library
- Epson S1D13700 Graphic Lcd Library
- Flash Memory Library
- Graphic Lcd Library
- I²C Library
- Keypad Library
- Lcd Library
- Manchester Code Library
- Multi Media Card Library
- OneWire Library
- Peripheral Pin Select Library
- Port Expander Library
- PS/2 Library
- PWM Library
- PWM Motor Library
- RS-485 Library
- Software I²C Library
- Software SPI Library
- Software UART Library
- Sound Library
- SPI Library
- SPI Ethernet Library
- SPI Ethernet ENC24J600 Library
- SPI Graphic Lcd Library
- SPI Lcd Library
- SPI Lcd8 Library
- SPI T6963C Graphic Lcd Library
- T6963C Graphic Lcd Library
- TFT Display Library
- Touch Panel Library
- Touch Panel TFT Library
- UART Library
- USB Library

Digital Signal Processing Libraries

- FIR Filter Library
- IIR Filter Library
- FFT Library
- Bit Reverse Complex Library
- Vectors Library
- Matrices Library

Miscellaneous Libraries

- Button Library
- C Type Library
- Conversions Library
- Setjmp Library
- String Library
- Time Library
- Trigon Library
- Trigonometry Library

See also Built-in Routines.

Hardware Libraries

- ADC Library
- CAN Library
- CANSPI Library
- Compact Flash Library
- Enhanced CAN Library
- EEPROM Library
- Epson S1D13700 Graphic Lcd Library
- Flash Memory Library
- Graphic Lcd Library
- I²C Library
- Keypad Library
- Lcd Library
- Manchester Code Library
- Multi Media Card Library
- OneWire Library
- Peripheral Pin Select Library
- Port Expander Library
- PS/2 Library
- PWM Library
- PWM Motor Library
- RS-485 Library
- Software I²C Library
- Software SPI Library
- Software UART Library
- Sound Library
- SPI Library
- SPI Ethernet Library
- SPI Ethernet ENC24J600 Library
- SPI Graphic Lcd Library
- SPI Lcd Library
- SPI Lcd8 Library
- SPI T6963C Graphic Lcd Library
- T6963C Graphic Lcd Library
- TFT Display Library
- Touch Panel Library
- Touch Panel TFT Library
- UART Library
- USB Library

ADC Library

ADC (Analog to Digital Converter) module is available with a number of dsPIC30/33 and PIC24 MCU modules. ADC is an electronic circuit that converts continuous signals to discrete digital numbers. ADC Library provides you a comfortable work with the module.

Library Routines

- ADCx_Init
- ADCx_Init_Advanced
- ADCx_Get_Sample
- ADCx_Read
- ADC_Set_Active

ADCx_Init

Prototype	`sub procedure ADCx_Init()`
Description	This routines configures ADC module to work with default settings. The internal ADC module is set to: - single channel conversion - 10-bit conversion resolution - unsigned integer data format - auto-convert - VRef+ : AVdd, VRef- : AVss - instruction cycle clock - conversion clock : 32*Tcy - auto-sample time : 31TAD
Parameters	None.
Returns	Nothing.
Requires	- MCU with built-in ADC module. - ADC library routines require you to specify the module you want to use. To select the desired ADC module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.
Example	`ADC1_Init()   ' Initialize ADC1 module with default settings`
Notes	- Number of ADC modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ADCx_Init_Advanced

Prototype

```
' dsPIC30F and PIC24FJ prototype
sub procedure ADC1_Init_Advanced(dim Reference as word)

' dsPIC33FJ and PIC24HJprototype
sub procedureADCx_Init_Advanced(dim ADCMode as word, dim Reference as word)
```

Description

This routine configures the internal ADC module to work with user defined settings.

Parameters

- `ADCMode`: resolution of the ADC module.
- `Reference`: voltage reference used in ADC process.

Description	Predefined library const
ADC mode:	
`10-bit resolution`	`_ADC_10bit`
`12-bit resolution`	`_ADC_12bit`
Voltage reference	
`Internal voltage reference`	`_ADC_INTERNAL_REF`
`External voltage reference`	`_ADC_EXTERNAL_REF`

Returns

Nothing.

Requires

- MCU with built-in ADC module.
- ADC library routines require you to specify the module you want to use. To select the desired ADC module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.

Example

```
ADC1_Init_Advanced(_ADC_10bit, _ADC_INTERNAL_REF)  ' sets ADC module in 12-bit resolution mode with internal reference used
```

Notes

- Number of ADC modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.
- Not all MCUs support advanced configuration. Please, read the appropriate datasheet before utilizing this library.

ADCx_Get_Sample

Prototype	`sub function ADCx_Get_Sample(dim channel as word) as word`
Description	The function enables ADC module and reads the specified analog channel input.
Parameters	- `channel` represents the channel from which the analog value is to be acquired.
Returns	10-bit or 12-bit (depending on selected mode by ADCx_Init_Advanced or MCU) unsigned value from the specified `channel`.
Requires	- The MCU with built-in ADC module. - Prior to using this routine, ADC module needs to be initialized. See ADCx_Init and ADCx_Init_Advanced. - ADC library routines require you to specify the module you want to use. To select the desired ADC module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Before using the function, be sure to configure the appropriate TRISx bits to designate pins as inputs.

Example	`dim adc_value as word`
	`...`
	`adc_value = ADC1_Get_Sample(10)  ' read analog value from ADC1 module channel 10`

Notes	- Number of ADC modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library. - The function sets the appropriate bit in the ADPCFG registers to enable analog function of the chosen pin. - Refer to the appropriate Datasheet for channel-to-pin mapping.

ADCx_Read

Prototype	`sub function ADCx_Read(dim channel as word) as word`
Description	The function initializes, enables ADC module and reads the specified analog channel input.
Parameters	- `channel` represents the channel from which the analog value is to be acquired.
Returns	10-bit or 12-bit (depending on the MCU) unsigned value from the specified `channel`.
Requires	- The MCU with built-in ADC module. - ADC library routines require you to specify the module you want to use. To select the desired ADC module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ADC modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library. - Before using the function, be sure to configure the appropriate TRISx bits to designate pins as inputs.

Example	`dim adc_value as word`
	`...`
	`adc_value = ADC1_Read(10)   ' read analog value from ADC1 module channel 10`

Notes	- This is a standalone routine, so there is no need for a previous initialization of ADC module. - The function sets the appropriate bit in the ADPCFG registers to enable analog function of the chosen pin. - Refer to the appropriate Datasheet for channel-to-pin mapping.

ADC_Set_Active

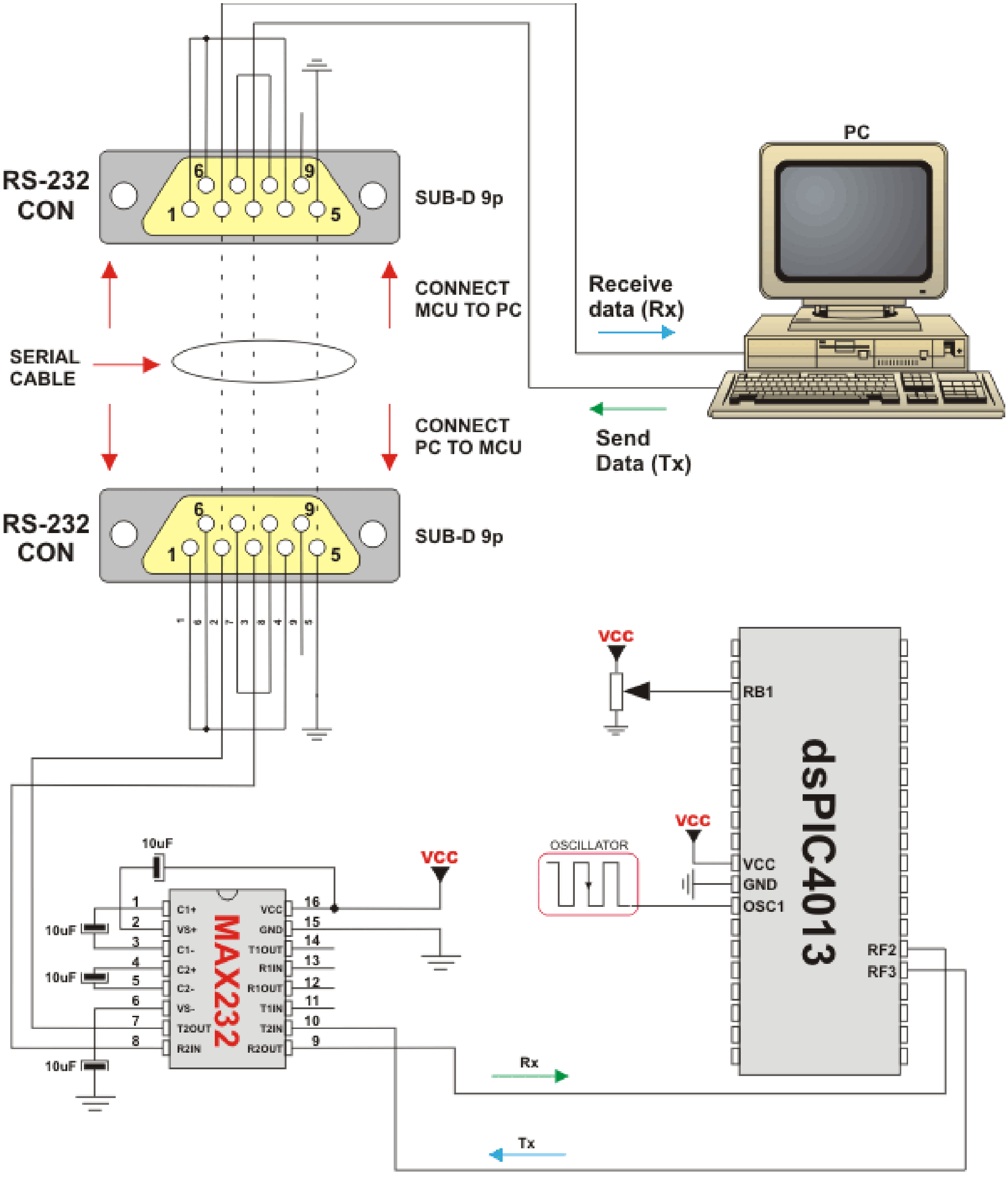
Prototype	`sub procedure ADC_Set_Active(dim adc_gs as ^TADC_Get_Sample)`
Description	Sets active ADC module.
Parameters	Parameters: - `adc_gs`: ADCx_Get_Sample handler.
Returns	Nothing.
Requires	Routine is available only for MCUs with multiple ADC modules. Used ADC module must be initialized before using this routine. See ADCx_Init and ADCx_Init_Advanced routines.
Example	`' Activate ADC2 module` `ADC_Set_Active(@ADC2_Get_Sample)`
Notes	None.

Library Example

This code snippet reads analog value from the channel 1 and sends readings as a text over UART1.

Copy Code To Clipboard

```
program ADC_on_LEDs
dim ADCresult as word
    txt as char[6]

main:
  PORTB = 0x0000        ' clear PORTB
  TRISB = 0xFFFF        ' PORTB is input
  ADC1_Init()           ' Enable ADC module
  UART1_Init(9600)      ' Initialize UART communication

  while TRUE
    ADCresult = ADC1_Get_Sample(1)    ' Acquire ADC sample
    WordToStr(ADCresult, txt)         ' convert its value to string
    UART1_Write_Text(txt)             ' and send it to UART terminal
    Delay_ms(50)
  wend
end.
```



ADC HW connection

CAN Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides a library (driver) for working with the dsPIC30F CAN module.

The CAN is a very robust protocol that has error detection and signalization, self–checking and fault confinement. Faulty CAN data and remote frames are re-transmitted automatically, similar to the Ethernet.

Data transfer rates depend on distance. For example, 1 Mbit/s can be achieved at network lengths below 40m while 250 Kbit/s can be achieved at network lengths below 250m. The greater distance the lower maximum bitrate that can be achieved. The lowest bitrate defined by the standard is 200Kbit/s. Cables used are shielded twisted pairs.

CAN supports two message formats:

- Standard format, with 11 identifier bits, and
- Extended format, with 29 identifier bits

Important:

- Consult the CAN standard about CAN bus termination resistance.
- CAN library routines require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.
- Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

Library Routines

- CANxSetOperationMode
- CANxGetOperationMode
- CANxInitialize
- CANxSetBaudRate
- CANxSetMask
- CANxSetFilter
- CANxRead
- CANxWrite

CANxSetOperationMode

Prototype	`sub procedure CANxSetOperationMode(dim mode, WAIT as word)`
Description	Sets the CAN module to requested mode.
Parameters	- `mode`: CAN module operation mode. Valid values: `CAN_OP_MODE` constants. See CAN_OP_MODE constants. - `WAIT`: CAN mode switching verification request. If `WAIT == 0`, the call is non-blocking. The function does not verify if the CAN module is switched to requested mode or not. Caller must use CANxGetOperationMode to verify correct operation mode before performing mode specific operation. If `WAIT != 0`, the call is blocking – the function won't "return" until the requested mode is set.
Returns	Nothing.
Requires	MCU with the CAN module. MCU must be connected to the CAN transceiver (MCP2551 or similar) which is connected to the CAN bus.

Example

```
' set the CAN1 module into configuration mode (wait inside CAN1SetOperationMode
until this mode is set)
CAN1SetOperationMode(_CAN_MODE_CONFIG, 0xFF)
```

Notes	- CAN library routine require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

CANxGetOperationMode

Prototype	`sub function CANxGetOperationMode() as word`
Description	The function returns current operation mode of the CAN module. See CAN_OP_MODE constants or device datasheet for operation mode codes.
Parameters	None.
Returns	Current operation mode.
Requires	MCU with the CAN module. MCU must be connected to the CAN transceiver (MCP2551 or similar) which is connected to the CAN bus.

Example

```
' check whether the CAN1 module is in Normal mode and if it is then do
something.
if (CAN1GetOperationMode() = _CAN_MODE_NORMAL) then
  ...
end if
```

Notes	- CAN library routine require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

CANxInitialize

Prototype	`sub procedure CANxInitialize(dim SJW, BRP, PHSEG1, PHSEG2, PROPSEG, CAN_CONFIG_FLAGS as word)`
Description	Initializes the CAN module. The internal dsPIC30F CAN module is set to: - Disable CAN capture - Continue CAN operation in Idle mode - Do not abort pending transmissions - Fcan clock : 4*Tcy (Fosc) - Baud rate is set according to given parameters - CAN mode is set to Normal - Filter and mask registers IDs are set to zero - Filter and mask message frame type is set according to `CAN_CONFIG_FLAGS` value `SAM`, `SEG2PHTS`, `WAKFIL` and `DBEN` bits are set according to `CAN_CONFIG_FLAGS` value.
Parameters	- `SJW` as defined in MCU's datasheet (CAN Module) - `BRP` as defined in MCU's datasheet (CAN Module) - `PHSEG1` as defined in MCU's datasheet (CAN Module) - `PHSEG2` as defined in MCU's datasheet (CAN Module) - `PROPSEG` as defined in MCU's datasheet (CAN Module) - `CAN_CONFIG_FLAGS` is formed from predefined constants. See CAN_CONFIG_FLAGS constants.
Returns	Nothing.
Requires	MCU with the CAN module. MCU must be connected to the CAN transceiver (MCP2551 or similar) which is connected to the CAN bus.
Example	See code below.
Notes	- CAN mode NORMAL will be set on exit. - CAN library routine require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

Example

```
' initialize the CAN1 module with appropriate baud rate and message acceptance
flags along with the sampling rules
dim can_config_flags as word
...
can_config_flags = _CAN_CONFIG_SAMPLE_THRICE and  ' Form value to be used
                   _CAN_CONFIG_PHSEG2_PRG_ON and  ' with CAN1Initialize
                   _CAN_CONFIG_STD_MSG            and
                   _CAN_CONFIG_DBL_BUFFER_ON      and
                   _CAN_CONFIG_MATCH_MSG_TYPE     and
                   _CAN_CONFIG_LINE_FILTER_OFF

CAN1Initialize(1,3,3,3,1,can_config_flags)   ' initialize the CAN1 module
```

CANxSetBaudRate

Prototype	`sub procedure CANxSetBaudRate(dim SJW, BRP, PHSEG1, PHSEG2, PROPSEG, CAN_CONFIG_FLAGS as word)`
Description	Sets CAN baud rate. Due to complexity of the CAN protocol, you can not simply force a bps value. Instead, use this function when CAN is in Config mode. Refer to datasheet for details. `SAM`, `SEG2PHTS` and `WAKFIL` bits are set according to `CAN_CONFIG_FLAGS` value. Refer to datasheet for details.
Parameters	- `SJW` as defined in MCU's datasheet (CAN Module) - `BRP` as defined in MCU's datasheet (CAN Module) - `PHSEG1` as defined in MCU's datasheet (CAN Module) - `PHSEG2` as defined in MCU's datasheet (CAN Module) - `PROPSEG` as defined in MCU's datasheet (CAN Module) - `CAN_CONFIG_FLAGS` is formed from predefined constants. See CAN_CONFIG_FLAGS constants.
Returns	Nothing.
Requires	MCU with the CAN module. MCU must be connected to the CAN transceiver (MCP2551 or similar) which is connected to the CAN bus. CAN must be in Config mode, otherwise the function will be ignored. See CANxSetOperationMode.

Example

```
' set required baud rate and sampling rules
dim can_config_flags as word
...
CAN1SetOperationMode(_CAN_MODE_CONFIG,0xFF)  ' set CONFIGURATION mode (CAN1 module must be in config mode for baud rate settings)

can_config_flags  = _CAN_CONFIG_SAMPLE_THRICE and ' Form value to be used
                    _CAN_CONFIG_PHSEG2_PRG_ON and ' with CAN1Initialize
                    _CAN_CONFIG_STD_MSG           and
                    _CAN_CONFIG_DBL_BUFFER_ON     and
                    _CAN_CONFIG_MATCH_MSG_TYPE    and
                    _CAN_CONFIG_LINE_FILTER_OFF

CAN1SetBaudRate(1,3,3,3,1,can_config_flags)  ' set the CAN1 module baud rate
```

Notes	- CAN library routine require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

CANxSetMask

Prototype	`sub procedure CANxSetMask(dim CAN_MASK as word, dim val as longint, dim CAN_CONFIG_FLAGS as word)`
Description	Function sets mask for advanced filtering of messages. Given `value` is bit adjusted to appropriate buffer mask registers.

Parameters	- `CAN_MASK`: CAN module mask number. Valid values: `CAN_MASK` constants. See CAN_MASK constants.
	- `val`: mask register value. This value is bit-adjusted to appropriate buffer mask registers
	- `CAN_CONFIG_FLAGS`: selects type of message to filter. Valid values:
	`- _CAN_CONFIG_ALL_VALID_MSG,`
	`- _CAN_CONFIG_MATCH_MSG_TYPE & _CAN_CONFIG_STD_MSG,`
	`- _CAN_CONFIG_MATCH_MSG_TYPE & _CAN_CONFIG_XTD_MSG.`
	See CAN_CONFIG_FLAGS constants.

Returns	Nothing.
Requires	MCU with the CAN module. MCU must be connected to the CAN transceiver (MCP2551 or similar) which is connected to the CAN bus. CAN must be in Config mode, otherwise the function will be ignored. See CANxSetOperationMode.

Example	`' set appropriate filter mask and message type value`
	`CAN1SetOperationMode(_CAN_MODE_CONFIG,0xFF)  ' set CONFIGURATION mode (CAN1 module must be in config mode for mask settings)`
	
	`' Set all B1 mask bits to 1 (all filtered bits are relevant)`
	`' Note that -1 is just a cheaper way to write 0xFFFFFFFF.`
	`' Complement will do the trick and fill it up with ones.`
	`CAN1SetMask(_CAN_MASK_B1, -1, _CAN_CONFIG_MATCH_MSG_TYPE and _CAN_CONFIG_XTD_MSG)`

Notes	- CAN library routine require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

CANxSetFilter

Prototype	`sub procedure CANxSetFilter(dim CAN_FILTER as word, dim val as longint, dim CAN_CONFIG_FLAGS as word)`
Description	Function sets message filter. Given `value` is bit adjusted to appropriate buffer mask registers.
Parameters	- `CAN_FILTER`: CAN module filter number. Valid values: `CAN_FILTER` constants. See CAN_FILTER constants. - `val`: filter register value. This value is bit-adjusted to appropriate filter registers - `CAN_CONFIG_FLAGS:` selects type of message to filter. Valid values: `_CAN_CONFIG_STD_MSG` and `_CAN_CONFIG_XTD_MSG`. See CAN_CONFIG_FLAGS constants.
Returns	Nothing.
Requires	MCU with the CAN module. MCU must be connected to the CAN transceiver (MCP2551 or similar) which is connected to the CAN bus. CAN must be in Config mode, otherwise the function will be ignored. See CANxSetOperationMode.

Example	`' set appropriate filter value and message type`
	`CAN1SetOperationMode(_CAN_MODE_CONFIG,0xFF) ' set CONFIGURATION mode (CAN1 module must be in config mode for filter settings)`
	
	`' Set id of filter B1_F1 to 3:`
	`CAN1SetFilter(_CAN_FILTER_B1_F1, 3, _CAN_CONFIG_XTD_MSG)`

Notes	- CAN library routine require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

CANxRead

Prototype

```
sub function CANxRead(dim byref id as longint, dim byref data as byte[1],
dim dataLen, CAN_RX_MSG_FLAGS as word) as word
```

Description

If at least one full Receive Buffer is found, it will be processed in the following way:

- Message ID is retrieved and stored to location pointed by `id` pointer
- Message data is retrieved and stored to array pointed by `data` pointer
- Message length is retrieved and stored to location pointed by `dataLen` pointer
- Message flags are retrieved and stored to location pointed by `CAN_RX_MSG_FLAGS` pointer

Parameters

- `id`: message identifier address
- `data`: an array of bytes up to 8 bytes in length
- `dataLen`: data length address
- `CAN_RX_MSG_FLAGS`: message flags address. For message receive flags format refer to `CAN_RX_MSG_FLAGS` constants. See CAN_RX_MSG_FLAGS constants.

Returns

- `0` if nothing is received
- `0xFFFF` if one of the Receive Buffers is full (message received)

Requires

MCU with the CAN module.
MCU must be connected to the CAN transceiver (MCP2551 or similar) which is connected to the CAN bus.
CAN must be in Config mode, otherwise the function will be ignored. See CANxSetOperationMode.

Example

```
' check the CAN1 module for received messages. If any was received do
something.
dim msg_rcvd, rx_flags, data_len as word
    data as byte[8]
    msg_id as longint
...
CAN1SetOperationMode(_CAN_MODE_NORMAL,0xFF)   ' set NORMAL mode (CAN1 module
must be in mode in which receive is possible)
...
rx_flags = 0                                  ' clear message flags
if (msg_rcvd = CAN1Read(msg_id, data, data_len, rx_flags)<>0) then
  ...
end if
```

Notes

- CAN library routine require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.
- Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

CANxWrite

Prototype	`sub function CANxWrite(dim id as longint, dim byref data_ as byte[1], dim dataLen, CAN_TX_MSG_FLAGS as word) as word`
Description	If at least one empty Transmit Buffer is found, the function sends message in the queue for transmission.
Parameters	- `id`: CAN message identifier. Valid values: 11 or 29 bit values, depending on message type (standard or extended) - `data`: data to be sent - `dataLen`: data length. Valid values: `0..8` - `CAN_RX_MSG_FLAGS`: message flags. Valid values: `CAN_TX_MSG_FLAGS` constants. See CAN_TX_MSG_FLAGS constants.
Returns	- `0` if all Transmit Buffers are busy - `0xFFFF` if at least one Transmit Buffer is available
Requires	MCU with the CAN module. MCU must be connected to the CAN transceiver (MCP2551 or similar) which is connected to the CAN bus. CAN must be in Config mode, otherwise the function will be ignored. See CANxSetOperationMode.

Example

```
' send message extended CAN message with appropriate ID and data
dim tx_flags as word
    data as byte[8]
    msg_id as longint
...
CAN1SetOperationMode(_CAN_MODE_NORMAL,0xFF)   ' set NORMAL mode (CAN1 must be in mode in which transmission is possible)

tx_flags = _CAN_TX_PRIORITY_0  and
           _CAN_TX_XTD_FRAME   and
           _CAN_TX_NO_RTR_FRAME              ' set message flags
CAN1Write(msg_id, data, 1, tx_flags)
```

Notes	- CAN library routine require you to specify the module you want to use. To use the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

CAN Constants

There is a number of constants predefined in CAN library. To be able to use the library effectively, you need to be familiar with these. You might want to check the example at the end of the chapter.

CAN_OP_MODE Constants

`CAN_OP_MODE` constants define CAN operation mode. Function CANxSetOperationMode expects one of these as its argument:

Copy Code To Clipboard

```
const     _CAN_MODE_BITS    as word = $E0   ' Use this to access opmode  bits
          _CAN_MODE_NORMAL     as word = 0x00
          _CAN_MODE_DISABLE    as word = 0x01
          _CAN_MODE_LOOP       as word = 0x02
          _CAN_MODE_LISTEN     as word = 0x03
          _CAN_MODE_CONFIG     as word = 0x04
          _CAN_MODE_LISTEN_ALL as word = 0x07
```

CAN_CONFIG_FLAGS Constants

`CAN_CONFIG_FLAGS` constants define flags related to CAN module configuration. Functions CANxInitialize and CANxSetBaudRate expect one of these (or a bitwise combination) as their argument:

Copy Code To Clipboard

```
const
    _CAN_CONFIG_DEFAULT          as word = 0xFF    ' 11111111

    _CAN_CONFIG_PHSEG2_PRG_BIT   as word = 0x01
    _CAN_CONFIG_PHSEG2_PRG_ON    as word = 0xFF    ' XXXXXXX1
    _CAN_CONFIG_PHSEG2_PRG_OFF   as word = 0xFE    ' XXXXXXX0

    _CAN_CONFIG_LINE_FILTER_BIT as word = 0x02
    _CAN_CONFIG_LINE_FILTER_ON   as word = 0xFF    ' XXXXXX1X
    _CAN_CONFIG_LINE_FILTER_OFF as word = 0xFD    ' XXXXXX0X

    _CAN_CONFIG_SAMPLE_BIT       as word = 0x04
    _CAN_CONFIG_SAMPLE_ONCE      as word = 0xFF    ' XXXXX1XX
    _CAN_CONFIG_SAMPLE_THRICE    as word = 0xFB    ' XXXXX0XX

    _CAN_CONFIG_MSG_TYPE_BIT     as word = 0x08
    _CAN_CONFIG_STD_MSG          as word = 0xFF    ' XXXX1XXX
    _CAN_CONFIG_XTD_MSG          as word = 0xF7    ' XXXX0XXX

    _CAN_CONFIG_DBL_BUFFER_BIT   as word = 0x10
    _CAN_CONFIG_DBL_BUFFER_ON    as word = 0xFF    ' XXX1XXXX
    _CAN_CONFIG_DBL_BUFFER_OFF   as word = 0xEF    ' XXX0XXXX
```

```
    _CAN_CONFIG_MATCH_TYPE_BIT  as word = 0x20
    _CAN_CONFIG_ALL_VALID_MSG   as word = 0xDF   ' XX0XXXXX
    _CAN_CONFIG_MATCH_MSG_TYPE  as word = 0xFF   ' XX1XXXXX
```

You may use bitwise `and` to form config byte out of these values. For example:

Copy Code To Clipboard

```
init  = _CAN_CONFIG_SAMPLE_THRICE    and
        _CAN_CONFIG_PHSEG2_PRG_ON    and
        _CAN_CONFIG_STD_MSG          and
        _CAN_CONFIG_DBL_BUFFER_ON    and
        _CAN_CONFIG_VALID_XTD_MSG    and
        _CAN_CONFIG_LINE_FILTER_OFF
...
CAN1Initialize(1, 1, 3, 3, 1, init)   ' initialize CAN
```

CAN_TX_MSG_FLAGS Constants

`CAN_TX_MSG_FLAGS` are flags related to transmission of a CAN message:

Copy Code To Clipboard

```
const
    _CAN_TX_PRIORITY_BITS as word = 0x03
    _CAN_TX_PRIORITY_0    as word = 0xFC   ' XXXXXX00
    _CAN_TX_PRIORITY_1    as word = 0xFD   ' XXXXXX01
    _CAN_TX_PRIORITY_2    as word = 0xFE   ' XXXXXX10
    _CAN_TX_PRIORITY_3    as word = 0xFF   ' XXXXXX11

    _CAN_TX_FRAME_BIT     as word = 0x08
    _CAN_TX_STD_FRAME     as word = 0xFF   ' XXXXX1XX
    _CAN_TX_XTD_FRAME     as word = 0xF7   ' XXXXX0XX

    _CAN_TX_RTR_BIT       as word = 0x40
    _CAN_TX_NO_RTR_FRAME  as word = 0xFF   ' X1XXXXXX
    _CAN_TX_RTR_FRAME     as word = 0xBF   ' X0XXXXXX
```

You may use bitwise `and` to adjust the appropriate flags. For example:

Copy Code To Clipboard

```
' form value to be used with CANSendMessage:
send_config = _CAN_TX_PRIORITY_0     and
              _CAN_TX_XTD_FRAME      and
              _CAN_TX_NO_RTR_FRAME
...
CANSendMessage(id, data, 1, send_config)
```

CAN_RX_MSG_FLAGS Constants

`CAN_RX_MSG_FLAGS` are flags related to reception of CAN message. If a particular bit is set; corresponding meaning is TRUE or else it will be FALSE.

Copy Code To Clipboard

```
const
    _CAN_RX_FILTER_BITS  as word = 0x0007   ' Use this to access filter bits
    _CAN_RX_FILTER_1     as word = 0x00
    _CAN_RX_FILTER_2     as word = 0x01
    _CAN_RX_FILTER_3     as word = 0x02
    _CAN_RX_FILTER_4     as word = 0x03
    _CAN_RX_FILTER_5     as word = 0x04
    _CAN_RX_FILTER_6     as word = 0x05
    _CAN_RX_OVERFLOW     as word = 0x08  ' Set if Overflowed else cleared
    _CAN_RX_INVALID_MSG  as word = 0x10  ' Set if invalid else cleared
    _CAN_RX_XTD_FRAME    as word = 0x20  ' Set if XTD message else cleared
    _CAN_RX_RTR_FRAME    as word = 0x40  ' Set if RTR message else cleared
   _CAN_RX_DBL_BUFFERED as word = 0x80' Set if this message was hardware double-
buffered
```

You may use bitwise `and` to adjust the appropriate flags. For example:

Copy Code To Clipboard

```
if (MsgFlag and _CAN_RX_OVERFLOW) <> 0 then
  ...
  ' Receiver overflow has occurred.
  ' We have lost our previous message.
end if
```

CAN_MASK Constants

`CAN_MASK` constants define mask codes. Function CANxSetMask expects one of these as its argument:

Copy Code To Clipboard

```
const
    _CAN_MASK_B1 as word = 0
    _CAN_MASK_B2 as word = 1
```

CAN_FILTER Constants

`CAN_FILTER` constants define filter codes. Function CANxSetFilter expects one of these as its argument:

Copy Code To Clipboard

```
const
    _CAN_FILTER_B1_F1 as word = 0
    _CAN_FILTER_B1_F2 as word = 1
    _CAN_FILTER_B2_F1 as word = 2
    _CAN_FILTER_B2_F2 as word = 3
    _CAN_FILTER_B2_F3 as word = 4
    _CAN_FILTER_B2_F4 as word = 5
```

Library Example

The example demonstrates CAN protocol. The 1st node initiates the communication with the 2nd node by sending some data to its address. The 2nd node responds by sending back the data incremented by 1. The 1st node then does the same and sends incremented data back to the 2nd node, etc.

Code for the first CAN node:

Copy Code To Clipboard

```
program CAN_1st

dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags, Rx_Data_Len as word
    RxTx_Data as byte[8]
    Rx_ID     as longint
    Msg_Rcvd  as word

const ID_1st as longint = 12111
const ID_2nd as longint = 3                                  ' node IDs

main:

  ADPCFG = 0xFFFF
  PORTB  = 0
  TRISB  = 0

  Can_Init_Flags   = 0
  Can_Send_Flags   = 0
  Can_Rcv_Flags    = 0

Can_Send_Flags   = _CAN_TX_PRIORITY_0 and             ' form value to be used
                     _CAN_TX_XTD_FRAME and            ' with CANSendMessage
                     _CAN_TX_NO_RTR_FRAME

Can_Init_Flags   = _CAN_CONFIG_SAMPLE_THRICE and      ' form value to be used
                     _CAN_CONFIG_PHSEG2_PRG_ON and    ' with CANInitialize
                     _CAN_CONFIG_XTD_MSG and
                     _CAN_CONFIG_DBL_BUFFER_ON and
                     _CAN_CONFIG_MATCH_MSG_TYPE and
                     _CAN_CONFIG_LINE_FILTER_OFF
```

```
RxTx_Data[0] = 9
  CAN1Initialize(1,3,3,3,1,Can_Init_Flags)             ' initialize CAN
  CAN1SetOperationMode(_CAN_MODE_CONFIG,0xFF)          ' set CONFIGURATION mode

  CAN1SetMask(_CAN_MASK_B1, -1, _CAN_CONFIG_MATCH_MSG_TYPE and _CAN_CONFIG_XTD_MSG)
' set all mask1 bits to ones
  CAN1SetMask(_CAN_MASK_B2, -1, _CAN_CONFIG_MATCH_MSG_TYPE and _CAN_CONFIG_XTD_MSG)
' set all mask2 bits to ones
   CAN1SetFilter(_CAN_FILTER_B2_F3,ID_2nd,_CAN_CONFIG_XTD_MSG)  ' set id of filter B2_F3
to 2nd node ID

  CAN1SetOperationMode(_CAN_MODE_NORMAL,0xFF)                  ' set NORMAL mode


  CAN1Write(ID_1st, RxTx_Data, 1, Can_Send_Flags)

  while TRUE
    Msg_Rcvd = CAN1Read(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)
    if ((Rx_ID = ID_2nd) and (Msg_Rcvd <> 0)) <> 0 then
      PORTB = RxTx_Data[0]                             ' output data at PORTB
      RxTx_Data[0] = RxTx_Data[0] + 1
      Delay_ms(10)
      CAN1Write(ID_1st, RxTx_Data, 1, Can_Send_Flags)  ' send incremented data back
    end if
  wend
end.
```

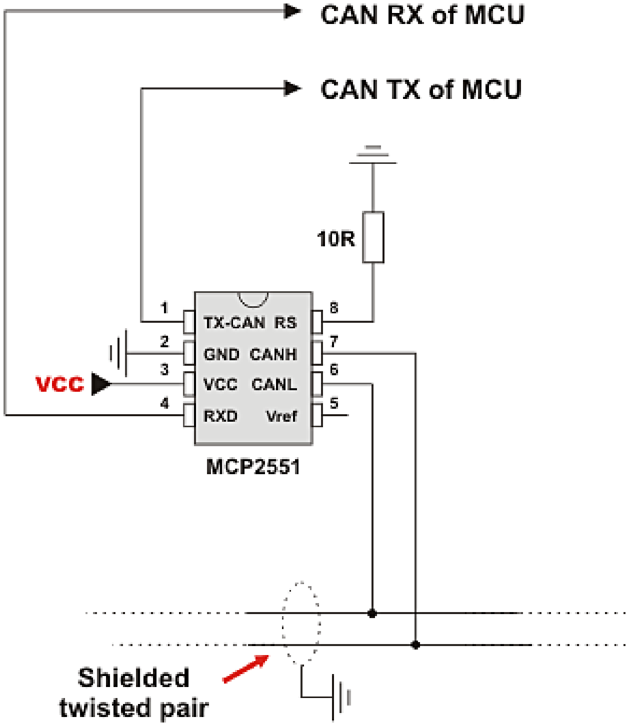
Code for the second CAN node:

Copy Code To Clipboard

```
program Can_2nd

dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags, Rx_Data_Len as word
    RxTx_Data as byte[8]
    Rx_ID     as longint
    Msg_Rcvd  as word

const ID_1st as longint = 12111
const ID_2nd as longint = 3                          ' node IDs

main:
  ADPCFG = 0xFFFF
  PORTB  = 0
  TRISB  = 0

  Can_Init_Flags = 0
  Can_Send_Flags = 0
  Can_Rcv_Flags  = 0
```

```
Can_Send_Flags =  _CAN_TX_PRIORITY_0 and           ' form value to be used
                  _CAN_TX_XTD_FRAME and            ' with CANSendMessage
                  _CAN_TX_NO_RTR_FRAME

Can_Init_Flags  =  _CAN_CONFIG_SAMPLE_THRICE and   ' form value to be used
                   _CAN_CONFIG_PHSEG2_PRG_ON and   ' with CANInitialize
                   _CAN_CONFIG_XTD_MSG and
                   _CAN_CONFIG_DBL_BUFFER_ON and
                   _CAN_CONFIG_MATCH_MSG_TYPE and
                   _CAN_CONFIG_LINE_FILTER_OFF

  CAN1Initialize(1,3,3,3,1,Can_Init_Flags)           ' initialize CAN
  CAN1SetOperationMode(_CAN_MODE_CONFIG,0xFF)        ' set CONFIGURATION mode

  CAN1SetMask(_CAN_MASK_B1, -1, _CAN_CONFIG_MATCH_MSG_TYPE and _CAN_CONFIG_XTD_MSG)
' set all mask1 bits to ones
  CAN1SetMask(_CAN_MASK_B2, -1, _CAN_CONFIG_MATCH_MSG_TYPE and _CAN_CONFIG_XTD_MSG)
' set all mask2 bits to ones
  CAN1SetFilter(_CAN_FILTER_B1_F1,ID_1st,_CAN_CONFIG_XTD_MSG)    ' set id of filter_B1_F1
to 1st node ID

  CAN1SetOperationMode(_CAN_MODE_NORMAL,0xFF)                    ' set NORMAL mode

  while TRUE
    Msg_Rcvd = CAN1Read(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)
    if ((Rx_ID = ID_1st) and (Msg_Rcvd <> 0)) <> 0 then
      PORTB   = RxTx_Data[0]                              ' output data at portB
      RxTx_Data[0] = RxTx_Data[0] + 1
      CAN1Write(ID_2nd, RxTx_Data, 1, Can_Send_Flags)     ' send incremented data back
    end if
  wend
end.
```

HW Connection



Example of interfacing CAN transceiver with MCU and CAN bus

CANSPI Library

The SPI module is available with a number of the dsPIC30/33 and PIC24 MCUs. The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library (driver) for working with mikroElektronika's CANSPI Add-on boards (with MCP2515 or MCP2510) via SPI interface.

The CAN is a very robust protocol that has error detection and signalization, self–checking and fault confinement. Faulty CAN data and remote frames are re-transmitted automatically, similar to the Ethernet.

In the mikroBasic PRO for dsPIC30/33 and PIC24, each routine of the CAN library has its own CANSPI counterpart with identical syntax. For more information on Controller Area Network, consult the CAN Library. Note that an effective communication speed depends on SPI and certainly is slower than "real" CAN.

CAN supports two message formats:

- Standard format, with 11 identifier bits and
- Extended format, with 29 identifier bits

Important:

- Consult the CAN standard about CAN bus termination resistance.
- An effective CANSPI communication speed depends on SPI and certainly is slower than "real" CAN.
- The library uses the SPI module for communication. User must initialize appropriate SPI module before using the CANSPI Library.
- For MCUs with multiple SPI modules it is possible to initialize both of them and then switch by using the SPI_Set_Active routine.
- Number of SPI modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.
- CANSPI module refers to mikroElektronika's CANSPI Add-on board connected to SPI module of MCU.

Library Dependency Tree

External dependencies of CANSPI Library

The following variables must be defined in all projects using CANSPI Library:	Description:	Example:
`dim CanSpi_CS as sbit sfr external`	Chip Select line.	`dim CanSpi_CS as sbit at LATF0_bit`
`dim CanSpi_Rst as sbit sfr external`	Reset line.	`dim CanSpi_Rst as sbit at LATF1_bit`
`dim CanSpi_CS_Direction as sbit sfr external`	Direction of the Chip Select pin.	`dim CanSpi_CS_Direction as sbit at TRISF0_bit`
`dim CanSpi_Rst_Direction as sbit sfr external`	Direction of the Reset pin.	`dim CanSpi_Rst_Direction as sbit at TRISF1_bit`

Library Routines

- CANSPISetOperationMode
- CANSPIGetOperationMode
- CANSPIInit
- CANSPISetBaudRate
- CANSPISetMask
- CANSPISetFilter
- CANSPIRead
- CANSPIWrite

CANSPISetOperationMode

Prototype	`sub procedure CANSPISetOperationMode(dim mode as byte, dim WAIT as byte)`
Description	Sets the CANSPI module to requested mode.
Parameters	`mode`: CANSPI module operation mode. Valid values: `CANSPI_OP_MODE` constants. See CANSPI_OP_MODE constants. `WAIT`: CANSPI mode switching verification request. If `WAIT == 0`, the call is non-blocking. The function does not verify if the CANSPI module is switched to requested mode or not. Caller must use `CANSPIGetOperationMode` to verify correct operation mode before performing mode specific operation. If `WAIT != 0`, the call is blocking – the function won't "return" until the requested mode is set.
Returns	Nothing.
Requires	The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.
Example	`' set the CANSPI module into configuration mode (wait inside CANSPISetOperationMode until this mode is set)` `CANSPISetOperationMode(_CANSPI_MODE_CONFIG, 0xFF)`
Notes	None.

CANSPIGetOperationMode

Prototype	`sub function CANSPIGetOperationMode() as byte`
Description	The function returns current operation mode of the CANSPI module. Check CANSPI_OP_MODE constants or device datasheet for operation mode codes.
Parameters	None.
Returns	Current operation mode.
Requires	The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.

Example

```
' check whether the CANSPI module is in Normal mode and if it is do something.
if (CANSPIGetOperationMode() = _CANSPI_MODE_NORMAL) then
  ...
end if
```

Notes	None.

CANSPIInit

Prototype	`sub procedure CANSPIInit(dim SJW, BRP, PHSEG1, PHSEG2, PROPSEG, CANSPI_CONFIG_FLAGS as char)`
Description	Initializes the CANSPI module. Stand-Alone CAN controller in the CANSPI module is set to: - Disable CAN capture - Continue CAN operation in Idle mode - Do not abort pending transmissions - Fcan clock: 4*Tcy (Fosc) - Baud rate is set according to given parameters - CAN mode: Normal - Filter and mask registers IDs are set to zero - Filter and mask message frame type is set according to CANSPI_CONFIG_FLAGS value `SAM`, `SEG2PHTS`, `WAKFIL` and `DBEN` bits are set according to CANSPI_CONFIG_FLAGS value.
Parameters	- `SJW` as defined in MCU's datasheet (CAN Module) - `BRP` as defined in MCU's datasheet (CAN Module) - `PHSEG1` as defined in MCU's datasheet (CAN Module) - `PHSEG2` as defined in MCU's datasheet (CAN Module) - `PROPSEG` as defined in MCU's datasheet (CAN Module) - `CANSPI_CONFIG_FLAGS` is formed from predefined constants. See CANSPI_CONFIG_FLAGS constants.
Returns	Nothing.

Requires	Global variables: - `CanSpi_CS`: Chip Select line - `CanSpi_Rst`: Reset line - `CanSpi_CS_Direction`: Direction of the Chip Select pin - `CanSpi_Rst_Direction`: Direction of the Reset pin must be defined before using this function. The CANSPI routines are supported only by MCUs with the SPI module. The SPI module needs to be initialized. See the SPIx_Init and SPIx_Init_Advanced routines. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.
Example	

```
' CANSPI module connections
dim CanSpi_CS     as sbit at  LATF0_bit
    CanSpi_CS_Direction  as sbit at TRISF0_bit
    CanSpi_Rst  as sbit at LATF12_bit
    CanSpi_Rst_Direction  as sbit at TRISF1_bit
' End CANSPI module connections

dim CANSPI_Init_Flags as word
...
CANSPI_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE and
                   _CANSPI_CONFIG_PHSEG2_PRG_ON  and
                   _CANSPI_CONFIG_STD_MSG        and
                   _CANSPI_CONFIG_DBL_BUFFER_ON  and
                   _CANSPI_CONFIG_VALID_XTD_MSG  and
                   _CANSPI_CONFIG_LINE_FILTER_OFF
...
SPI1_Init()                               ' initialize SPI1 module
CANSPI(1,3,3,3,1,CANSPI_Init_Flags)   ' initialize CANSPI
```

Notes	- CANSPI mode NORMAL will be set on exit.

CANSPISetBaudRate

Prototype	`sub procedure CANSPISetBaudRate(dim SJW, BRP, PHSEG1, PHSEG2, PROPSEG, CANSPI_CONFIG_FLAGS as byte)`
Returns	Nothing.
Description	Sets the CANSPI module baud rate. Due to complexity of the CAN protocol, you can not simply force a bps value. Instead, use this function when the CANSPI module is in Config mode. `SAM`, `SEG2PHTS` and `WAKFIL` bits are set according to CANSPI_CONFIG_FLAGS value. Refer to datasheet for details.
Parameters	- `SJW` as defined in MCU's datasheet (CAN Module) - `BRP` as defined in MCU's datasheet (CAN Module) - `PHSEG1` as defined in MCU's datasheet (CAN Module) - `PHSEG2` as defined in MCU's datasheet (CAN Module) - `PROPSEG` as defined in MCU's datasheet (CAN Module) - `CANSPI_CONFIG_FLAGS` is formed from predefined constants. See CANSPI_CONFIG_FLAGS constants.
Returns	Nothing.
Requires	The CANSPI module must be in Config mode, otherwise the function will be ignored. See CANSPISetOperationMode. The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.

Example

```
' set required baud rate and sampling rules
dim CANSPI_CONFIG_FLAGS as byte
...
CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)  ' set CONFIGURATION mode
(CANSPI1 module mast be in config mode for baud rate settings)
CANSPI_CONFIG_FLAGS = _CANSPI_CONFIG_SAMPLE_THRICE and
                   _CANSPI_CONFIG_PHSEG2_PRG_ON  and
                   _CANSPI_CONFIG_STD_MSG        and
                   _CANSPI_CONFIG_DBL_BUFFER_ON  and
                   _CANSPI_CONFIG_VALID_XTD_MSG  and
                   _CANSPI_CONFIG_LINE_FILTER_OFF
CANSPISetBaudRate(1, 1, 3, 3, 1, CANSPI_CONFIG_FLAGS)
```

Notes	None.

CANSPISetMask

Prototype	`sub procedure CANSPISetMask(dim CANSPI_MASK as byte, dim val as longint, dim CANSPI_CONFIG_FLAGS as byte)`
Description	Configures mask for advanced filtering of messages. The parameter `value` is bit-adjusted to the appropriate mask registers.
Parameters	- `CANSPI_MASK`: CAN module mask number. Valid values: `CANSPI_MASK` constants. See CANSPI_MASK constants. - `val`: mask register value. This value is bit-adjusted to appropriate buffer mask registers - `CANSPI_CONFIG_FLAGS`: selects type of message to filter. Valid values: - `_CANSPI_CONFIG_ALL_VALID_MSG`, - `_CANSPI_CONFIG_MATCH_MSG_TYPE & _CANSPI_CONFIG_STD_MSG`, - `_CANSPI_CONFIG_MATCH_MSG_TYPE & _CANSPI_CONFIG_XTD_MSG`. See CANSPI_CONFIG_FLAGS constants.
Returns	Nothing.
Requires	The CANSPI module must be in Config mode, otherwise the function will be ignored. See CANSPISetOperationMode. The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.

Example	`' set the appropriate filter mask and message type value`
	`CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)          ' set CONFIGURATION mode (CANSPI1 module must be in config mode for mask settings)`
	
	`' Set all B1 mask bits to 1 (all filtered bits are relevant):`
	`' Note that -1 is just a cheaper way to write 0xFFFFFFFF.`
	`' Complement will do the trick and fill it up with ones.`
	`CANSPISetMask(_CANSPI_MASK_B1, -1, _CANSPI_CONFIG_MATCH_MSG_TYPE and _CANSPI_CONFIG_XTD_MSG)`

Notes	None.

CANSPISetFilter

Prototype	`sub procedure CANSPISetFilter(dim CAN_FILTER as byte, dim val as longint, dim CANSPI_CONFIG_FLAGS as byte)`
Description	Configures message filter. The parameter `value` is bit-adjusted to the appropriate filter registers.
Parameters	- `CANSPI_FILTER`: CAN module filter number. Valid values: `CANSPI_FILTER` constants. See CANSPI_FILTER constants. - `val`: filter register value. This value is bit-adjusted to appropriate filter registers - `CANSPI_CONFIG_FLAGS`: selects type of message to filter. Valid values: `_CANSPI_CONFIG_STD_MSG` and `_CANSPI_CONFIG_XTD_MSG`. See CANSPI_CONFIG_FLAGS constants.
Returns	Nothing.
Requires	The CANSPI module must be in Config mode, otherwise the function will be ignored. See CANSPISetOperationMode. The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.

Example	`' set the appropriate filter value and message type`
	`CANSPI1SetOperationMode(_CANSPI_MODE_CONFIG,0xFF)   ' set CONFIGURATION mode (CANSPI1 module must be in config mode for filter settings)`
	
	`' Set id of filter B1_F1 to 3:`
	`CANSPI1SetFilter(_CANSPI_FILTER_B1_F1, 3, _CANSPI_CONFIG_XTD_MSG)`

Notes	None.

CANSPIRead

Prototype	`sub function CANSPIRead(dim byref id as longint, dim byref Data_ as byte[8], dim byref DataLen as byte, dim byref CAN_RX_MSG_FLAGS as byte) as byte`
Description	If at least one full Receive Buffer is found, it will be processed in the following way: - Message ID is retrieved and stored to location provided by the `id` parameter - Message data is retrieved and stored to a buffer provided by the `data` parameter - Message length is retrieved and stored to location provided by the `dataLen` parameter - Message flags are retrieved and stored to location provided by the `CANSPI_RX_MSG_FLAGS` parameter
Parameters	- `id`: message identifier address - `data`: an array of bytes up to 8 bytes in length - `dataLen`: data length address - `CANSPI_RX_MSG_FLAGS`: message flags address. For message receive flags format refer to `CANSPI_RX_MSG_FLAGS` constants. See CANSPI_RX_MSG_FLAGS constants.
Returns	- `0` if nothing is received - `0xFFFF` if one of the Receive Buffers is full (message received)
Requires	The CANSPI module must be in a mode in which receiving is possible. See CANSPISetOperationMode. The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.

Example

```
' check the CANSPI1 module for received messages. If any was received do
something.
dim msg_rcvd, rx_flags, data_len as byte
    data as byte[8]
    msg_id as longint
...
CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)   ' set NORMAL mode (CANSPI1
module must be in mode in which receive is possible)
...
rx_flags = 0                                        ' clear message flags
if (msg_rcvd = CANSPIRead(msg_id, data, data_len, rx_flags)) then
  ...
end if
```

Notes	None.

CANSPIWrite

Prototype	`sub function CANSPIWrite(dim id as longint, dim byref Data_ as byte[8], dim DataLen, CANSPI_TX_MSG_FLAGS as byte) as byte`
Description	If at least one empty Transmit Buffer is found, the function sends message in the queue for transmission.
Parameters	- `id`: CAN message identifier. Valid values: 11 or 29 bit values, depending on message type (standard or extended) - `Data`: data to be sent - `DataLen`: data length. Valid values: 0..8 - `CANSPI_TX_MSG_FLAGS`: message flags. Valid values: `CANSPI_TX_MSG_FLAGS` constants. See CANSPI_TX_MSG_FLAGS constants.
Returns	- `0` if all Transmit Buffers are busy - `0xFFFF` if at least one Transmit Buffer is available
Requires	The CANSPI module must be in mode in which transmission is possible. See CANSPISetOperationMode. The CANSPI routines are supported only by MCUs with the SPI module. MCU has to be properly connected to mikroElektronika's CANSPI Extra Board or similar hardware. See connection example at the bottom of this page.

Example

```
' send message extended CAN message with the appropriate ID and data
dim tx_flags as byte
    data as byte[8]
    msg_id as longint
...
CANSPISetOperationMode(_CAN_MODE_NORMAL,0xFF)    ' set NORMAL mode (CANSPI1 must be in mode in which transmission is possible)

tx_flags = _CANSPI_TX_PRIORITY_0 and _CANSPI_TX_XTD_FRAME    ' set message flags
CANSPIWrite(msg_id, data, 2, tx_flags)
```

Notes	None.

CANSPI Constants

There is a number of constants predefined in the CANSPI library. You need to be familiar with them in order to be able to use the library effectively. Check the example at the end of the chapter.

CANSPI_OP_MODE Constants

The `CANSPI_OP_MODE` constants define CANSPI operation mode. Function CANSPISetOperationMode expects one of these as it's argument:

Copy Code To Clipboard

```
const
    _CANSPI_MODE_BITS   as byte = $E0    Use this to access opmode  bits
    _CANSPI_MODE_NORMAL as byte = 0
    _CANSPI_MODE_SLEEP  as byte = $20
    _CANSPI_MODE_LOOP   as byte = $40
    _CANSPI_MODE_LISTEN as byte = $60
    _CANSPI_MODE_CONFIG as byte = $80
```

CANSPI_CONFIG_FLAGS Constants

The `CANSPI_CONFIG_FLAGS` constants define flags related to the CANSPI module configuration. The functions CANSPIInit, CANSPISetBaudRate, CANSPISetMask and CANSPISetFilter expect one of these (or a bitwise combination) as their argument:

Copy Code To Clipboard

```
const
    _CANSPI_CONFIG_DEFAULT        as byte  = $FF   ' 11111111

    _CANSPI_CONFIG_PHSEG2_PRG_BIT as byte  = $01
    _CANSPI_CONFIG_PHSEG2_PRG_ON  as byte  = $FF   ' XXXXXXX1
    _CANSPI_CONFIG_PHSEG2_PRG_OFF as byte  = $FE   ' XXXXXXX0

    _CANSPI_CONFIG_LINE_FILTER_BIT as byte = $02
    _CANSPI_CONFIG_LINE_FILTER_ON  as byte = $FF   ' XXXXXX1X
    _CANSPI_CONFIG_LINE_FILTER_OFF as byte = $FD   ' XXXXXX0X

    _CANSPI_CONFIG_SAMPLE_BIT      as byte = $04
    _CANSPI_CONFIG_SAMPLE_ONCE     as byte = $FF   ' XXXXX1XX
    _CANSPI_CONFIG_SAMPLE_THRICE   as byte = $FB   ' XXXXX0XX

    _CANSPI_CONFIG_MSG_TYPE_BIT    as byte = $08
    _CANSPI_CONFIG_STD_MSG         as byte = $FF   ' XXXX1XXX
    _CANSPI_CONFIG_XTD_MSG         as byte = $F7   ' XXXX0XXX

    _CANSPI_CONFIG_DBL_BUFFER_BIT  as byte = $10
    _CANSPI_CONFIG_DBL_BUFFER_ON   as byte = $FF   ' XXX1XXXX
    _CANSPI_CONFIG_DBL_BUFFER_OFF  as byte = $EF   ' XXX0XXXX

    _CANSPI_CONFIG_MSG_BITS        as byte = $60
    _CANSPI_CONFIG_ALL_MSG         as byte = $FF   ' X11XXXXX
    _CANSPI_CONFIG_VALID_XTD_MSG   as byte = $DF   ' X10XXXXX
    _CANSPI_CONFIG_VALID_STD_MSG   as byte = $BF   ' X01XXXXX
    _CANSPI_CONFIG_ALL_VALID_MSG   as byte = $9F   ' X00XXXXX
```

You may use bitwise `and` to form config byte out of these values. For example:

Copy Code To Clipboard

```
init  = _CANSPI_CONFIG_SAMPLE_THRICE    and
        _CANSPI_CONFIG_PHSEG2_PRG_ON    and
        _CANSPI_CONFIG_STD_MSG          and
        _CANSPI_CONFIG_DBL_BUFFER_ON    and
        _CANSPI_CONFIG_VALID_XTD_MSG    and
        _CANSPI_CONFIG_LINE_FILTER_OFF
...
CANSPIInit(1, 1, 3, 3, 1, init)    ' initialize CANSPI
```

CANSPI_TX_MSG_FLAGS Constants

`CANSPI_TX_MSG_FLAGS` are flags related to transmission of a CANSPI message:

Copy Code To Clipboard

```
const
    _CANSPI_TX_PRIORITY_BITS as byte = $03
    _CANSPI_TX_PRIORITY_0    as byte = $FC    ' XXXXXX00
    _CANSPI_TX_PRIORITY_1    as byte = $FD    ' XXXXXX01
    _CANSPI_TX_PRIORITY_2    as byte = $FE    ' XXXXXX10
    _CANSPI_TX_PRIORITY_3    as byte = $FF    ' XXXXXX11

    _CANSPI_TX_FRAME_BIT     as byte  = $08
    _CANSPI_TX_STD_FRAME     as byte  = $FF   ' XXXXX1XX
    _CANSPI_TX_XTD_FRAME     as byte  = $F7   ' XXXXX0XX

    _CANSPI_TX_RTR_BIT       as byte = $40
    _CANSPI_TX_NO_RTR_FRAME  as byte = $FF    ' X1XXXXXX
    _CANSPI_TX_RTR_FRAME     as byte = $BF    ' X0XXXXXX
```

You may use bitwise `and` to adjust the appropriate flags. For example:

Copy Code To Clipboard

```
' form value to be used with CANSendMessage:
send_config  = _CANSPI_TX_PRIORITY_0    and
               _CANSPI_TX_XTD_FRAME     and
               _CANSPI_TX_NO_RTR_FRAME
...
CANSPIWrite(id, data, 1, send_config)
```

CANSPI_RX_MSG_FLAGS Constants

`CANSPI_RX_MSG_FLAGS` are flags related to reception of CANSPI message. If a particular bit is set then corresponding meaning is TRUE or else it will be FALSE.

Copy Code To Clipboard

```
const
     _CANSPI_RX_FILTER_BITS  as byte = $07    ' Use this to access filter bits
     _CANSPI_RX_FILTER_1     as byte = $00
     _CANSPI_RX_FILTER_2     as byte = $01
     _CANSPI_RX_FILTER_3     as byte = $02
     _CANSPI_RX_FILTER_4     as byte = $03
     _CANSPI_RX_FILTER_5     as byte = $04
     _CANSPI_RX_FILTER_6     as byte = $05

     _CANSPI_RX_OVERFLOW     as byte = $08    ' Set if Overflowed else cleared
     _CANSPI_RX_INVALID_MSG  as byte = $10    ' Set if invalid else cleared
     _CANSPI_RX_XTD_FRAME    as byte = $20    ' Set if XTD message else cleared
     _CANSPI_RX_RTR_FRAME    as byte = $40    ' Set if RTR message else cleared
     _CANSPI_RX_DBL_BUFFERED as byte = $80    ' Set if this message was hardware double-
buffered
```

You may use bitwise `and` to adjust the appropriate flags. For example:

Copy Code To Clipboard

```
if (MsgFlag and _CANSPI_RX_OVERFLOW) <> 0 then
  ...
  ' Receiver overflow has occurred.
  ' We have lost our previous message.
end if
```

CANSPI_MASK Constants

The `CANSPI_MASK` constants define mask codes. Function CANSPISetMask expects one of these as it's argument:

Copy Code To Clipboard

```
const
     _CANSPI_MASK_B1 as byte = 0
     _CANSPI_MASK_B2 as byte = 1
```

CANSPI_FILTER Constants

The `CANSPI_FILTER` constants define filter codes. Functions CANSPISetFilter expects one of these as it's argument:

Copy Code To Clipboard

```
const
     _CANSPI_FILTER_B1_F1 as byte = 0
     _CANSPI_FILTER_B1_F2 as byte = 1
     _CANSPI_FILTER_B2_F1 as byte = 2
     _CANSPI_FILTER_B2_F2 as byte = 3
     _CANSPI_FILTER_B2_F3 as byte = 4
     _CANSPI_FILTER_B2_F4 as byte = 5
```

Library Example

The code is a simple demonstration of CANSPI protocol. This node initiates the communication with the 2nd node by sending some data to its address. The 2nd node responds by sending back the data incremented by 1. This (1st) node then does the same and sends incremented data back to the 2nd node, etc.

Code for the first CANSPI node:

Copy Code To Clipboard

```
program Can_Spi_1st

const ID_1st as longint = 12111
const ID_2nd as longint = 3

dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as word   ' can flags
    Rx_Data_Len as word                              ' received data length in bytes
    RxTx_Data   as byte[8]                                  ' can rx/tx data buffer
    Msg_Rcvd as byte                                        ' reception flag
    Tx_ID, Rx_ID as longword                                ' can rx and tx ID

' CANSPI module connections
dim CanSpi_CS            as sbit at LATF0_bit
    CanSpi_CS_Direction  as sbit at TRISF0_bit
    CanSpi_Rst           as sbit at LATF1_bit
    CanSpi_Rst_Direction as sbit at TRISF1_bit
' End CANSPI module connections

  ADPCFG = 0xFFFF                                  ' Configure AN pins as digital I/O

  PORTB = 0                                        ' clear PORTB
  TRISB = 0                                        ' set PORTB as output

  Can_Init_Flags = 0                               '
  Can_Send_Flags = 0                               ' clear flags
  Can_Rcv_Flags  = 0                               '

  Can_Send_Flags =  _CANSPI_TX_PRIORITY_0 and          ' form value to be used
                    _CANSPI_TX_XTD_FRAME and           '   with CANSPIWrite
                    _CANSPI_TX_NO_RTR_FRAME

  Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE and    ' Form value to be used
                    _CANSPI_CONFIG_PHSEG2_PRG_ON and   '  with CANSPIInit
                    _CANSPI_CONFIG_XTD_MSG and
                    _CANSPI_CONFIG_DBL_BUFFER_ON and
                    _CANSPI_CONFIG_VALID_XTD_MSG

' Initialize SPI1 module
  SPI1_Init()

CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)   ' initialize external CANSPI module
CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)      ' set CONFIGURATION mode
CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG)  ' set all mask1 bits to ones
CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG)  ' set all mask2 bits to ones
CANSPISetFilter(_CANSPI_FILTER_B2_F4,ID_2nd,_CANSPI_CONFIG_XTD_MSG)  ’set id of filter
B2_F4 to 2nd node ID
```

```
 CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)          ' set NORMAL mode

' Set initial data to be sent
  RxTx_Data[0] = 9

  CANSPIWrite(ID_1st, RxTx_Data, 1, Can_Send_Flags)         ' send initial message

  while (TRUE)                                              ' endless loop
    Msg_Rcvd = CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)    ' receive 
message
    if ((Rx_ID = ID_2nd) and Msg_Rcvd) then    ' if message received check id
      PORTB = RxTx_Data[0]                     ' id correct, output data at PORTD
      Inc(RxTx_Data[0])                        ' increment received data
      Delay_ms(10)
      CANSPIWrite(ID_1st, RxTx_Data, 1, Can_Send_Flags)   ' send incremented data back
    end if
  wend
end.
```

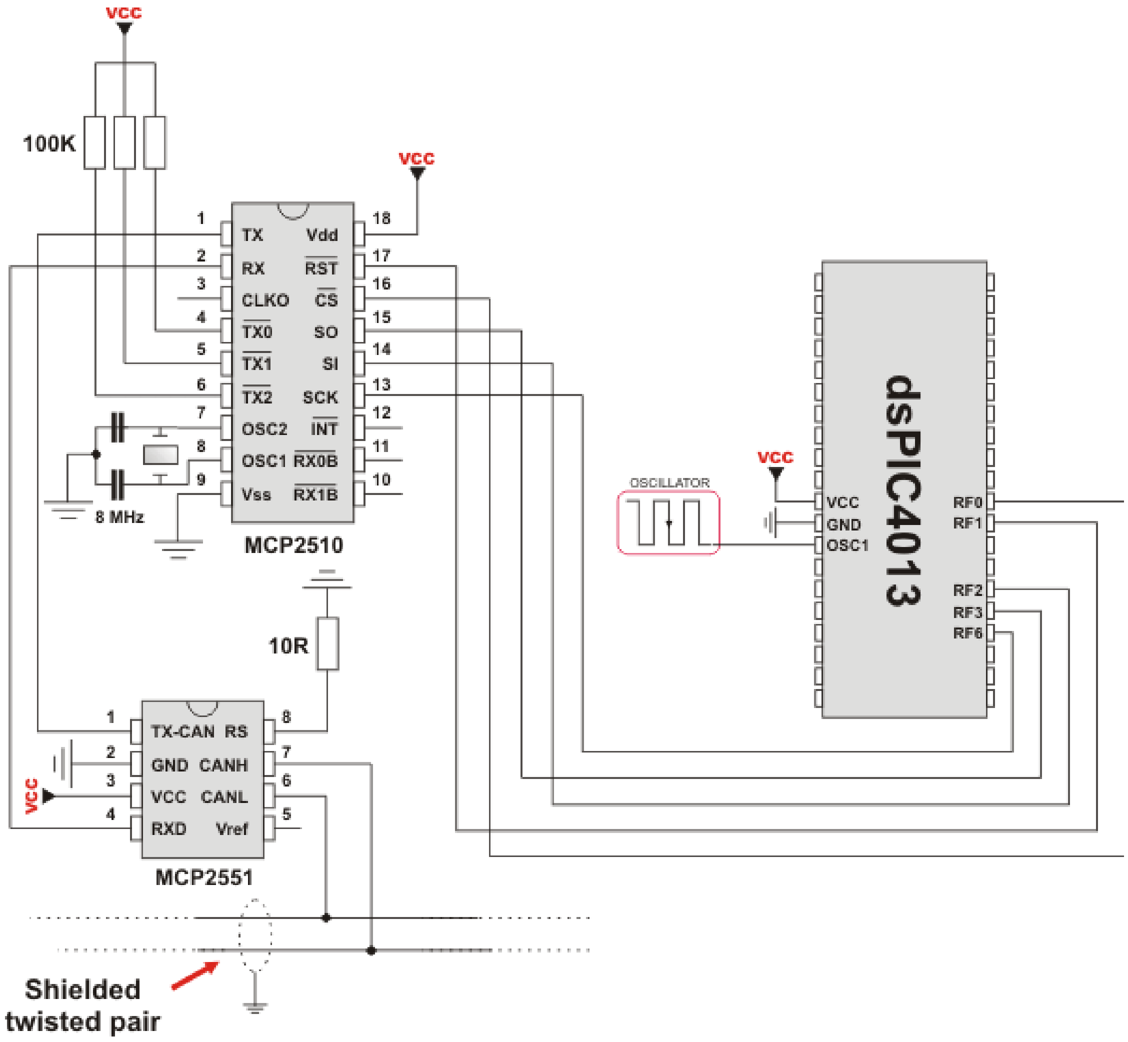
Code for the second CANSPI node:

Copy Code To Clipboard

```
program Can_Spi_2nd

const ID_1st as longint = 12111
const ID_2nd as longint = 3

dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as word   ' can flags
    Rx_Data_Len as word                                  ' received data length in bytes
    RxTx_Data   as byte[8]                                  ' can rx/tx data buffer
    Msg_Rcvd as byte                                        ' reception flag
    Tx_ID, Rx_ID as longword                                ' can rx and tx ID

' CANSPI module connections
dim CanSpi_CS            as sbit at  LATF0_bit
    CanSpi_CS_Direction  as sbit at TRISF0_bit
    CanSpi_Rst           as sbit at LATF1_bit
    CanSpi_Rst_Direction as sbit at TRISF1_bit
' End CANSPI module connections

  ADPCFG = 0xFFFF                                    ' Configure AN pins as digital I/O

  PORTB = 0                                          ' clear PORTB
  TRISB = 0                                          ' set PORTB as output

  Can_Init_Flags = 0                                 '
  Can_Send_Flags = 0                                 ' clear flags
  Can_Rcv_Flags  = 0                                 '
```

```
  Can_Send_Flags = _CANSPI_TX_PRIORITY_0 and           ' form value to be used
                   _CANSPI_TX_XTD_FRAME and            ' with CANSPIWrite
                   _CANSPI_TX_NO_RTR_FRAME

  Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE and    ' Form value to be used
                   _CANSPI_CONFIG_PHSEG2_PRG_ON and    ' with CANSPIInit
                   _CANSPI_CONFIG_XTD_MSG and
                   _CANSPI_CONFIG_DBL_BUFFER_ON and
                   _CANSPI_CONFIG_VALID_XTD_MSG and
                   _CANSPI_CONFIG_LINE_FILTER_OFF

' Initialize SPI1 module
  SPI1_Init()

  CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)      ' initialize external CANSPI module
  CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)  ' set CONFIGURATION mode
  CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG)  ' set all mask1 bits to ones
  CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG)  ' set all mask2 bits to ones
  CANSPISetFilter(_CANSPI_FILTER_B2_F3,ID_1st,_CANSPI_CONFIG_XTD_MSG) ' set id of filter
B2_F3 to 1st node ID

  CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)               ' set NORMAL mode

  while (TRUE)                                                   ' endless loop
     Msg_Rcvd = CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)  ' receive
message
    if ((Rx_ID = ID_1st) and Msg_Rcvd) then           ' if message received check id
      PORTB = RxTx_Data[0]                        ' id correct, output data at PORTB
      Inc(RxTx_Data[0])                                    ' increment received data
      CANSPIWrite(ID_2nd, RxTx_Data, 1, Can_Send_Flags)    ' send incremented data back
    end if
  wend
end.
```

HW Connection



Example of interfacing CAN transceiver MCP2510 with MCU via SPI interface

Compact Flash Library

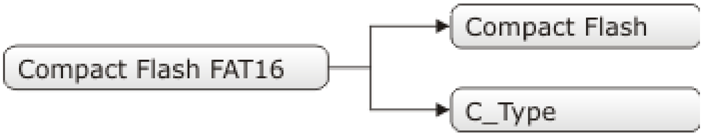
The Compact Flash Library provides routines for accessing data on Compact Flash card (abbr. CF further in text). CF cards are widely used memory elements, commonly used with digital cameras. Great capacity and excellent access time of only a few microseconds make them very attractive for microcontroller applications.

In CF card, data is divided into sectors. One sector usually comprises 512 bytes. Routines for file handling, the `Cf_Fat` routines, are not performed directly but successively through 512B buffer.

Important:

- Routines for file handling can be used only with FAT16 file system.
- Library functions create and read files from the root directory only.
- Library functions populate both FAT1 and FAT2 tables when writing to files, but the file data is being read from the FAT1 table only; i.e. there is no recovery if the FAT1 table gets corrupted.
- If MMC/SD card has Master Boot Record (MBR), the library will work with the first available primary (logical) partition that has non-zero size. If MMC/SD card has Volume Boot Record (i.e. there is only one logical partition and no MBRs), the library works with entire card as a single partition. For more information on MBR, physical and logical drives, primary/secondary partitions and partition tables, please consult other resources, e.g. Wikipedia and similar.
- Before writing operation, make sure not to overwrite boot or FAT sector as it could make your card on PC or digital camera unreadable. Drive mapping tools, such as Winhex, can be of great assistance.

Library Dependency Tree



External dependencies of Compact Flash Library

The following variables must be defined in all projects using Compact Flash Library:	**Description:**	**Example:**
`dim CF_Data_Port as byte sfr external`	Compact Flash Data Port.	`dim CF_Data_Port as byte at PORTD`
`dim CF_RDY as sbit sfr external`	Ready signal line.	`dim CF_RDY as sbit at RB7_bit`
`dim CF_WE as sbit sfr external`	Write Enable signal line.	`dim CF_WE as sbit at LATB6_bit`
`dim CF_OE as sbit sfr external`	Output Enable signal line.	`dim CF_OE as sbit at LATB5_bit`
`dim CF_CD1 as sbit sfr external`	Chip Detect signal line.	`dim CF_CD1 as sbit at RB4_bit`
`dim CF_CE1 as sbit sfr external`	Chip Enable signal line.	`dim CF_CE1 as sbit at LATB3_bit`
`dim CF_A2 as sbit sfr external`	Address pin 2.	`dim CF_A2 as sbit at LATB2_bit`
`dim CF_A1 as sbit sfr external`	Address pin 1.	`dim CF_A1 as sbit at LATB1_bit`
`dim CF_A0 as sbit sfr external`	Address pin 0.	`dim CF_A0 as sbit at LATB0_bit`
`dim CF_RDY_direction as sbit sfr external`	Direction of the Ready pin.	`dim CF_RDY_direction as sbit at TRISB7_bit`
`dim CF_WE_direction as sbit sfr external`	Direction of the Write Enable pin.	`dim CF_WE_direction as sbit at TRISB6_bit`
`dim CF_OE_direction as sbit sfr external`	Direction of the Output Enable pin.	`dim CF_OE_direction as sbit at TRISB5_bit`
`dim CF_CD1_direction as sbit sfr external`	Direction of the Chip Detect pin.	`dim CF_CD1_direction as sbit at TRISB4_bit`
`dim CF_CE1_direction as sbit sfr external`	Direction of the Chip Enable pin.	`dim CF_CE1_direction as sbit at TRISB3_bit`
`dim CF_A2_direction as sbit sfr external`	Direction of the Address 2 pin.	`dim CF_A2_direction as sbit at TRISB2_bit`
`dim CF_A1_direction as sbit sfr external`	Direction of the Address 1 pin.	`dim CF_A1_direction as sbit at TRISB1_bit`
`dim CF_A0_direction as sbit sfr external`	Direction of the Address 0 pin.	`dim CF_A0_direction as sbit at TRISB0_bit`

Library Routines

- Cf_Init
- Cf_Detect
- Cf_Enable
- Cf_Disable
- Cf_Read_Init
- Cf_Read_Byte
- Cf_Write_Init
- Cf_Write_Byte
- Cf_Read_Sector
- Cf_Write_Sector

Routines for file handling:

- Cf_Fat_Init
- Cf_Fat_QuickFormat
- Cf_Fat_Assign
- Cf_Fat_Reset
- Cf_Fat_Read
- Cf_Fat_Rewrite
- Cf_Fat_Append
- Cf_Fat_Delete
- Cf_Fat_Write
- Cf_Fat_Set_File_Date
- Cf_Fat_Get_File_Date
- Cf_Fat_Get_File_Date_Modified
- Cf_Fat_Get_File_Size
- Cf_Fat_Get_Swap_File

The following routine is for the internal use by compiler only:

- Cf_Issue_ID_Command

Cf_Init

Prototype

```
sub procedure Cf_Init()
```

Description

Initializes ports appropriately for communication with CF card.

Parameters

None.

Returns

Nothing.

Requires

Global variables:

- `CF_Data_Port` : Compact Flash data port
- `CF_RDY` : Ready signal line
- `CF_WE` : Write enable signal line
- `CF_OE` : Output enable signal line
- `CF_CD1` : Chip detect signal line
- `CF_CE1` : Enable signal line
- `CF_A2` : Address pin 2
- `CF_A1` : Address pin 1
- `CF_A0` : Address pin 0
- `CF_RDY_direction` : Direction of the Ready pin
- `CF_WE_direction` : Direction of the Write enable pin
- `CF_OE_direction` : Direction of the Output enable pin
- `CF_CD1_direction` : Direction of the Chip detect pin
- `CF_CE1_direction` : Direction of the Chip enable pin
- `CF_A2_direction` : Direction of the Address 2 pin
- `CF_A1_direction` : Direction of the Address 1 pin
- `CF_A0_direction` : Direction of the Address 0 pin

must be defined before using this function.

Example

```
' set compact flash pinout
dim Cf_Data_Port as byte at PORTD

dim CF_RDY as sbit at RB7_bit
dim CF_WE  as sbit at LATB6_bit   ' for writing to output pin always use latch
dim CF_OE  as sbit at LATB5_bit   ' for writing to output pin always use latch
dim CF_CD1 as sbit at RB4_bit
dim CF_CE1 as sbit at LATB3_bit   ' for writing to output pin always use latch
dim CF_A2  as sbit at LATB2_bit   ' for writing to output pin always use latch
dim CF_A1  as sbit at LATB1_bit   ' for writing to output pin always use latch
dim CF_A0  as sbit at LATB0_bit   ' for writing to output pin always use latch

dim CF_RDY_direction as sbit at TRISB7_bit
dim CF_WE_direction  as sbit at TRISB6_bit
dim CF_OE_direction  as sbit at TRISB5_bit
dim CF_CD1_direction as sbit at TRISB4_bit
dim CF_CE1_direction as sbit at TRISB3_bit
dim CF_A2_direction  as sbit at TRISB2_bit
dim CF_A1_direction  as sbit at TRISB1_bit
dim CF_A0_direction  as sbit at TRISB0_bit
' end of cf pinout

'Init CF
Cf_Init()
```

Notes

None.

Cf_Detect

Prototype	`sub function CF_Detect() as word`
Description	Checks for presence of CF card by reading the `chip detect` pin.
Parameters	None.
Returns	- `1` - if CF card was detected - `0` - otherwise
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.

Example

```
' Wait until CF card is inserted:
while (Cf_Detect() = 0)
  nop
wend
```

Notes	dsPIC30 family MCU and CF card voltage levels are different. The user must ensure that MCU's pin connected to CD line can read CF card Logical One correctly.

Cf_Enable

Prototype	`sub procedure Cf_Enable()`
Description	Enables the device. Routine needs to be called only if you have disabled the device by means of the Cf_Disable routine. These two routines in conjunction allow you to free/occupy data line when working with multiple devices.
Parameters	None.
Returns	Nothing.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.
Example	`' enable compact flash` `Cf_Enable()`
Notes	None.

Cf_Disable

Prototype	`sub procedure Cf_Disable()`
Description	Routine disables the device and frees the data lines for other devices. To enable the device again, call Cf_Enable. These two routines in conjunction allow you to free/occupy data line when working with multiple devices.
Parameters	None.
Returns	Nothing.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.
Example	`' disable compact flash` `Cf_Disable()`
Notes	None.

Cf_Read_Init

Prototype	`sub procedure Cf_Read_Init(dim address as longword, dim sectcnt as byte)`
Description	Initializes CF card for reading.
Parameters	- `address`: the first sector to be prepared for reading operation. - `sector_count`: number of sectors to be prepared for reading operation.
Returns	Nothing.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.
Example	`' initialize compact flash for reading from sector 590` `Cf_Read_Init(590, 1)`
Notes	None.

Cf_Read_Byte

Prototype	`sub function CF_Read_Byte() as byte`
Description	Reads one byte from Compact Flash sector buffer location currently pointed to by internal read pointers. These pointers will be autoicremented upon reading.
Parameters	None.
Returns	Returns a byte read from Compact Flash sector buffer.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init. CF card must be initialized for reading operation. See Cf_Read_Init.

Example	`' Read a byte from compact flash:`
	`dim data_ as byte`
	`...`
	`data_ = Cf_Read_Byte()`

Notes	Higher byte of the `unsigned` return value is cleared.

Cf_Write_Init

Prototype	`sub procedure Cf_Write_Init(dim address as longword, dim sectcnt as word)`
Description	Initializes CF card for writing.
Parameters	- `address`: the first sector to be prepared for writing operation. - `sectcnt`: number of sectors to be prepared for writing operation.
Returns	Nothing.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.
Example	`' initialize compact flash for writing to sector 590` `Cf_Write_Init(590, 1)`
Notes	None.

Cf_Write_Byte

Prototype	`sub procedure Cf_Write_Byte(dim data_ as byte)`
Description	Writes a byte to Compact Flash sector buffer location currently pointed to by writing pointers. These pointers will be autoicremented upon reading. When sector buffer is full, its contents will be transfered to appropriate flash memory sector.
Parameters	- `data_`: byte to be written.
Returns	Nothing.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init. CF card must be initialized for writing operation. See Cf_Write_Init.

Example	`dim data_ as byte`
	`...`
	`data_ = 0xAA`
	`Cf_Write_Byte(data_)`

Notes	None.

Cf_Read_Sector

Prototype	`sub procedure Cf_Read_Sector(dim sector_number as longword, dim byref buffer as byte[512])`
Description	Reads one sector (512 bytes). Read data is stored into buffer provided by the `buffer` parameter.
Parameters	- `sector_number`: sector to be read. - `buffer`: data buffer of at least 512 bytes in length.
Returns	Nothing.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.

Example	`' read sector 22`
	`dim data_ as byte[512]`
	`...`
	`Cf_Read_Sector(22, data_)`

Notes	None.

Cf_Write_Sector

Prototype	`sub procedure Cf_Write_Sector(dim sector_number as longword, dim byref buffer as byte[512])`
Description	Writes 512 bytes of data provided by the `buffer` parameter to one CF sector.
Parameters	- `sector_number`: sector to be written to. - `buffer`: data buffer of 512 bytes in length.
Returns	Nothing.
Requires	The corresponding MCU ports must be appropriately initialized for CF card. See Cf_Init.

Example	`' write to sector 22`
	`dim data_ as byte[512]`
	`...`
	`Cf_Write_Sector(22, data_)`

Notes	None.

Cf_Fat_Init

Prototype	`sub function Cf_Fat_Init() as word`
Description	Initializes CF card, reads CF FAT16 boot sector and extracts necessary data needed by the library.
Parameters	None.
Returns	- `0` - if CF card was detected and successfully initialized - `1` - if FAT16 boot sector was not found - `255` - if card was not detected
Requires	Nothing.

Example

```
'  init the FAT library
if (Cf_Fat_Init() = 0) then
  ...
end if
```

Notes	None.

Cf_Fat_QuickFormat

Prototype	`sub function Cf_Fat_QuickFormat(dim byref cf_fat_label as string[11]) as word`
Description	Formats to FAT16 and initializes CF card.
Parameters	- `cf_fat_label`: volume label (11 characters in length). If less than 11 characters are provided, the label will be padded with spaces. If null string is passed, the volume will not be labeled.
Returns	- `0` - if CF card was detected, successfully formated and initialized - `1` - if FAT16 format was unsuccessful - `255` - if card was not detected
Requires	Nothing.

Example

```
' format and initialize the FAT library
if ( Cf_Fat_QuickFormat("mikroE") = 0) then
  ...
end if
```

Notes	- This routine can be used instead or in conjunction with Cf_Fat_Init routine. - If CF card already contains a valid boot sector, it will remain unchanged (except volume label field) and only FAT and ROOT tables will be erased. Also, the new volume label will be set.

Cf_Fat_Assign

<table>
<tr><td>Prototype</td><td><code>sub function Cf_Fat_Assign(dim byref filename as char[12], dim file_cre_attr as byte) as word</code></td></tr>
<tr><td>Description</td><td>Assigns file for file operations (read, write, delete...). All subsequent file operations will be applied over the assigned file.</td></tr>
<tr><td>Parameters</td><td>- <code>filename</code>: name of the file that should be assigned for file operations. The file name should be in DOS 8.3 (file_name.extension) format. The file name and extension will be automatically padded with spaces by the library if they have less than length required (i.e. “mikro.tx” -> “mikro .tx “), so the user does not have to take care of that. The file name and extension are case insensitive. The library will convert them to proper case automatically, so the user does not have to take care of that.
Also, in order to keep backward compatibility with the first version of this library, file names can be entered as UPPERCASE string of 11 bytes in length with no dot character between the file name and extension (i.e. “MIKROELETXT” -> MIKROELE.TXT). In this case the last 3 characters of the string are considered to be file extension.

- <code>file_cre_attr</code>: file creation and attributes flags. Each bit corresponds to the appropriate file attribute:
<table>
<tr><th>Bit</th><th>Mask</th><th>Description</th></tr>
<tr><td>0</td><td>0x01</td><td>Read Only</td></tr>
<tr><td>1</td><td>0x02</td><td>Hidden</td></tr>
<tr><td>2</td><td>0x04</td><td>System</td></tr>
<tr><td>3</td><td>0x08</td><td>Volume Label</td></tr>
<tr><td>4</td><td>0x10</td><td>Subdirectory</td></tr>
<tr><td>5</td><td>0x20</td><td>Archive</td></tr>
<tr><td>6</td><td>0x40</td><td>Device (internal use only, never found on disk)</td></tr>
<tr><td>7</td><td>0x80</td><td>File creation flag. If the file does not exist and this flag is set, a new file with specified name will be created.</td></tr>
</table></td></tr>
<tr><td>Returns</td><td>- <code>0</code> if file does not exist and no new file is created.
- <code>1</code> if file already exists or file does not exist but a new file is created.</td></tr>
<tr><td>Requires</td><td>CF card and CF library must be initialized for file operations. See Cf_Fat_Init.</td></tr>
<tr><td>Example</td><td><code>‘ create file with archive attribut if it does not already exist</code>
<code>Cf_Fat_Assign(“MIKRO007.TXT”,0xA0)</code></td></tr>
<tr><td>Notes</td><td>Long File Names (LFN) are not supported.</td></tr>
</table>

Cf_Fat_Reset

Prototype	`sub procedure Cf_Fat_Reset(dim byref size as longword)`
Description	Opens currently assigned file for reading.
Parameters	- `size`: buffer to store file size to. After file has been open for reading its size is returned through this parameter.
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign.

Example	`dim size as longword`
	`...`
	`Cf_Fat_Reset(size)`

Notes	None.

Cf_Fat_Read

Prototype	`sub procedure Cf_Fat_Read(dim byref bdata as byte)`
Description	Reads a byte from currently assigned file opened for reading. Upon function execution file pointers will be set to the next character in the file.
Parameters	- `bdata`: buffer to store read byte to. Upon this function execution read byte is returned through this parameter.
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign. File must be open for reading. See Cf_Fat_Reset.

Example	`dim bdata as byte`
	`...`
	`Cf_Fat_Read(bdata)`

Notes	None.

Cf_Fat_Rewrite

Prototype	`sub procedure Cf_Fat_Rewrite()`
Description	Opens currently assigned file for writing. If the file is not empty its content will be erased.
Parameters	None.
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. The file must be previously assigned. See Cf_Fat_Assign.
Example	`' open file for writing` `Cf_Fat_Rewrite()`
Notes	None.

Cf_Fat_Append

Prototype	`sub procedure Cf_Fat_Append()`
Description	Opens currently assigned file for appending. Upon this function execution file pointers will be positioned after the last byte in the file, so any subsequent file writing operation will start from there.
Parameters	None.
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign.
Example	`' open file for appending` `Cf_Fat_Append()`
Notes	None.

Cf_Fat_Delete

Prototype	`sub procedure Cf_Fat_Delete()`
Description	Deletes currently assigned file from CF card.
Parameters	None.
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign.
Example	`' delete current file` `Cf_Fat_Delete()`
Notes	None.

Cf_Fat_Write

Prototype	`sub procedure Cf_Fat_Write(dim byref fdata as byte[512], dim data_len as word)`
Description	Writes requested number of bytes to currently assigned file opened for writing.
Parameters	- `fdata`: data to be written. - `data_len`: number of bytes to be written.
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign. File must be open for writing. See Cf_Fat_Rewrite or Cf_Fat_Append.

Example	`dim file_contents as array[42]`
	`...`
	`Cf_Fat_Write(file_contents, 42)  ' write data to the assigned file`

Notes	None.

Cf_Fat_Set_File_Date

Prototype	`sub procedure Cf_Fat_Set_File_Date(dim year as word, dim month as byte, dim day as byte, dim hours as byte, dim mins as byte, dim seconds as byte)`
Description	Sets the date/time stamp. Any subsequent file writing operation will write this stamp to currently assigned file's time/date attributes.
Parameters	- `year`: year attribute. Valid values: 1980-2107 - `month`: month attribute. Valid values: 1-12 - `day`: day attribute. Valid values: 1-31 - `hours`: hours attribute. Valid values: 0-23 - `mins`: minutes attribute. Valid values: 0-59 - `seconds`: seconds attribute. Valid values: 0-59
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign. File must be open for writing. See Cf_Fat_Rewrite or Cf_Fat_Append.
Example	`Cf_Fat_Set_File_Date(2005,9,30,17,41,0)`
Notes	None.

Cf_Fat_Get_File_Date

Prototype	`sub procedure Cf_Fat_Get_File_Date(dim byref year as word, dim byref month as byte, dim byref day as byte, dim byref hours as byte, dim byref mins as byte)`
Description	Reads time/date attributes of currently assigned file.
Parameters	- `year`: buffer to store year attribute to. Upon function execution year attribute is returned through this parameter. - `month`: buffer to store month attribute to. Upon function execution month attribute is returned through this parameter. - `day`: buffer to store day attribute to. Upon function execution day attribute is returned through this parameter. - `hours`: buffer to store hours attribute to. Upon function execution hours attribute is returned through this parameter. - `mins`: buffer to store minutes attribute to. Upon function execution minutes attribute is returned through this parameter.
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign.

Example	`dim year as word`
	`dim month, day, hours, mins as byte`
	`...`
	`Cf_Fat_Get_File_Date_Modified(year, month, day, hours, mins)`

Notes	None.

Cf_Fat_Get_File_Date_Modified

Prototype	`sub procedure Cf_Fat_Get_File_Date_Modified(dim byref year as word, dim byref month, day, hours, mins as byte)`
Description	Retrieves the last modification date/time of the currently assigned file.
Parameters	- `year`: buffer to store year of modification attribute to. Upon function execution year of modification attribute is returned through this parameter. - `month`: buffer to store month of modification attribute to. Upon function execution month of modification attribute is returned through this parameter. - `day`: buffer to store day of modification attribute to. Upon function execution day of modification attribute is returned through this parameter. - `hours`: buffer to store hours of modification attribute to. Upon function execution hours of modification attribute is returned through this parameter. - `mins`: buffer to store minutes of modification attribute to. Upon function execution minutes of modification attribute is returned through this parameter.
Returns	Nothing.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign.

Example	`dim year as word`
	`dim month, day, hours, mins as byte`
	`...`
	`Cf_Fat_Get_File_Date_Modified(year, month, day, hours, mins)`

Notes	None.

Cf_Fat_Get_File_Size

Prototype	`sub function Cf_Fat_Get_File_Size() as longword`
Description	This function reads size of currently assigned file in bytes.
Parameters	None.
Returns	Size of the currently assigned file in bytes.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init. File must be previously assigned. See Cf_Fat_Assign.

Example	`dim my_file_size as longword`
	`...`
	`my_file_size = Cf_Fat_Get_File_Size()`

Notes	None.

Cf_Fat_Get_Swap_File

Prototype	`sub function Cf_Fat_Get_Swap_File(dim sectors_cnt as longword, dim byref filename as string[11], dim file_attr as byte) as longword`
Description	This function is used to create a swap file of predefined name and size on the CF media. If a file with specified name already exists on the media, search for consecutive sectors will ignore sectors occupied by this file. Therefore, it is recommended to erase such file if it exists before calling this function. If it is not erased and there is still enough space for a new swap file, this function will delete it after allocating new memory space for a new swap file. The purpose of the swap file is to make reading and writing to CF media as fast as possible, by using the Cf_Read_Sector() and Cf_Write_Sector() functions directly, without potentially damaging the FAT system. Swap file can be considered as a “window” on the media where the user can freely write/read data. It’s main purpose in the this library is to be used for fast data acquisition; when the time-critical acquisition has finished, the data can be re-written into a “normal” file, and formatted in the most suitable way.
Parameters	- `sectors_cnt`: number of consecutive sectors that user wants the swap file to have. - `filename`: name of the file that should be assigned for file operations. The file name should be in DOS 8.3 (file_name.extension) format. The file name and extension will be automatically padded with spaces by the library if they have less than length required (i.e. “mikro.tx” -> “mikro .tx “), so the user does not have to take care of that. The file name and extension are case insensitive. The library will convert them to proper case automatically, so the user does not have to take care of that. Also, in order to keep backward compatibility with the first version of this library, file names can be entered as UPPERCASE string of 11 bytes in length with no dot character between the file name and extension (i.e. “MIKROELETXT” -> MIKROELE.TXT). In this case the last 3 characters of the string are considered to be file extension. - `file_attr`: file creation and attributes flags. Each bit corresponds to the appropriate file attribute:

Parameters	

Bit	Mask	Description
0	0x01	Read Only
1	0x02	Hidden
2	0x04	System
3	0x08	Volume Label
4	0x10	Subdirectory
5	0x20	Archive
6	0x40	Device (internal use only, never found on disk)
7	0x80	Not used

Returns	- Number of the start sector for the newly created swap file, if there was enough free space on CF card to create file of required size. - `0` - otherwise.
Requires	CF card and CF library must be initialized for file operations. See Cf_Fat_Init.
Example	

```
'  Try to create a swap file with archive atribute, whose size will be at least 1000 sectors.
'  If it succeeds, it sends the No. of start sector over UART
dim size as longword
...
size = Cf_Fat_Get_Swap_File(1000, “mikroE.txt”, 0x20)
if (size <> 0) then
  UART1_Write(0xAA)
  UART1_Write(Lo(size))
  UART1_Write(Hi(size))
  UART1_Write(Higher(size))
  UART1_Write(Highest(size))
  UART1_Write(0xAA)
end if
```

Notes	Long File Names (LFN) are not supported.

Library Example

This project consists of several blocks that demonstrate various aspects of usage of the Cf_Fat16 library. These are:

- Creation of new file and writing down to it;
- Opening existing file and re-writing it (writing from start-of-file);
- Opening existing file and appending data to it (writing from end-of-file);
- Opening a file and reading data from it (sending it to USART terminal);
- Creating and modifying several files at once;
- Reading file contents;
- Deleting file(s);
- Creating the swap file (see Help for details);

Copy Code To Clipboard

```
program CF_Fat16_Test

dim
   ' set compact flash pinout
  Cf_Data_Port as byte at PORTD

  CF_RDY as sbit at RB7_bit
  CF_WE  as sbit at LATB6_bit    ' for writing to output pin always use latch
  CF_OE  as sbit at LATB5_bit    ' for writing to output pin always use latch
  CF_CD1 as sbit at RB4_bit
  CF_CE1 as sbit at LATB3_bit    ' for writing to output pin always use latch
  CF_A2  as sbit at LATB2_bit    ' for writing to output pin always use latch
  CF_A1  as sbit at LATB1_bit    ' for writing to output pin always use latch
  CF_A0  as sbit at LATB0_bit    ' for writing to output pin always use latch

  CF_RDY_direction as sbit at TRISB7_bit
  CF_WE_direction  as sbit at TRISB6_bit
  CF_OE_direction  as sbit at TRISB5_bit
  CF_CD1_direction as sbit at TRISB4_bit
  CF_CE1_direction as sbit at TRISB3_bit
  CF_A2_direction  as sbit at TRISB2_bit
  CF_A1_direction  as sbit at TRISB1_bit
  CF_A0_direction  as sbit at TRISB0_bit
 ' end of cf pinout

const LINE_LEN = 37
dim
  err_txt as string[20]
  file_contents as string[LINE_LEN]

  filename as string[14]     ' File names

  character as byte
  loop1, loop2 as byte
  i, size as longint

  Buffer as byte[512]
```

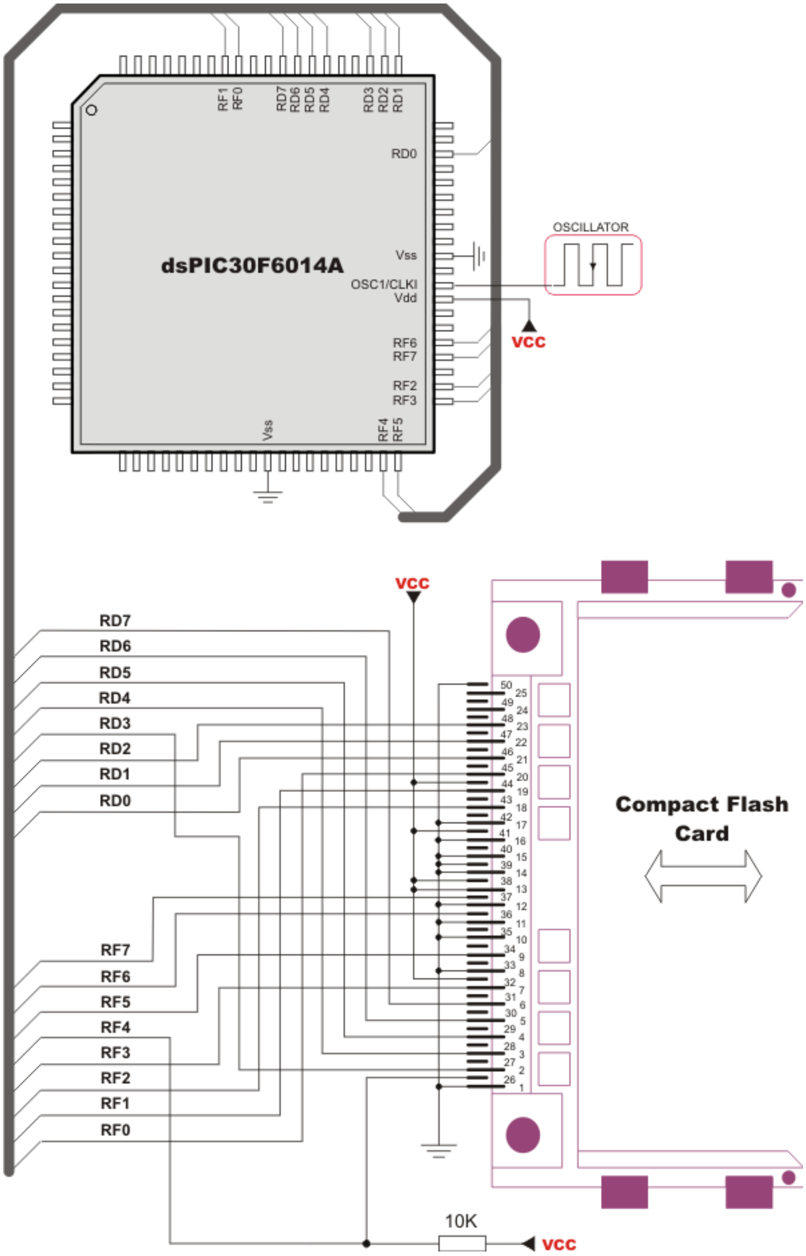
```
  ' UART write text and new line (carriage return + line feed)
sub procedure UART1_Write_Line( dim byref uart_text as string )
  UART1_Write_Text(uart_text)
  UART1_Write(13)
  UART1_Write(10)
end sub

'-------------- Creates new file and writes some data to it
sub procedure M_Create_New_File()
  filename[7] = "A"
  Cf_Fat_Set_File_Date(2005,6,21,10,35,0)         ' Set file date & time info
  Cf_Fat_Assign(filename, 0xA0)                  ' Will not find file and then create file
  Cf_Fat_Rewrite()                              ' To clear file and start with new data
  for loop1=1 to 90                               '  We want 5 files on the MMC card
    UART1_Write(".")
    file_contents[0] = loop1 div 10 + 48
    file_contents[1] = loop1 mod 10 + 48
    Cf_Fat_Write(file_contents, LINE_LEN-1)      ' write data to the assigned file
  next loop1
end sub

'-------------- Creates many new files and writes data to them
sub procedure M_Create_Multiple_Files()
  for loop2 = "B" to "Z"
    UART1_Write(loop2)                       ' this line can slow down the performance
    filename[7] = loop2                           ' set filename
    Cf_Fat_Set_File_Date(2005,6,21,10,35,0)       ' Set file date & time info
    Cf_Fat_Assign(filename, 0xA0)                ' find existing file or create a new one
    Cf_Fat_Rewrite()                           ' To clear file and start with new data
    for loop1 = 1 to 44
      file_contents[0] = loop1 div 10 + 48
      file_contents[1] = loop1 mod 10 + 48
      Cf_Fat_Write(file_contents, LINE_LEN-1)    ' write data to the assigned file
    next loop1
  next loop2
end sub

'-------------- Opens an existing file and rewrites it
sub procedure M_Open_File_Rewrite()
  filename[7] = "C"                              ' Set filename for single-file tests
  Cf_Fat_Assign(filename, 0)
  Cf_Fat_Rewrite()
  for loop1 = 1 to 55
    file_contents[0] = byte(loop1 div 10 + 48)
    file_contents[1] = byte(loop1 mod 10 + 48)
    Cf_Fat_Write(file_contents, LINE_LEN-1)      ' write data to the assigned file
  next loop1
end sub

'-------------- Opens an existing file and appends data to it
'               (and alters the date/time stamp)
sub procedure M_Open_File_Append()
```

```
   filename[7] = “B”
   Cf_Fat_Assign(filename, 0)
   Cf_Fat_Set_File_Date(2009, 1, 23, 17, 22, 0)
   Cf_Fat_Append
   file_contents = “ for mikroElektronika 2007”  ' Prepare file for append
   file_contents[26] = 10                        ' LF
   Cf_Fat_Write(file_contents, 27)               ' Write data to assigned file
end sub

'-------------- Opens an existing file, reads data from it and puts it to USART
sub procedure M_Open_File_Read()
  filename[7] = “B”
  Cf_Fat_Assign(filename, 0)
  Cf_Fat_Reset(size)               ' To read file, procedure returns size of file
  while size > 0
    Cf_Fat_Read(character)
    UART1_Write(character)        ' Write data to USART
    Dec(size)
  wend
end sub

'-------------- Deletes a file. If file doesn’t exist, it will first be created
'               and then deleted.
sub procedure M_Delete_File()
  filename[7] = “F”
  Cf_Fat_Assign(filename, 0)
  Cf_Fat_Delete()
end sub

'-------------- Tests whether file exists, and if so sends its creation date
'               and file size via USART
sub procedure M_Test_File_Exist()
dim
  fsize as longint
  year as word
  month_, day, hour_, minute_ as byte
  outstr as char[12]
  filename[7] = “B”        ' uncomment this line to search for file that DOES exists
'  filename[7] = “F”       ' uncomment this line to search for file that DOES NOT exist
  if Cf_Fat_Assign(filename, 0) <> 0 then
    '--- file has been found - get its date
    Cf_Fat_Get_File_Date(year,month_,day,hour_,minute_)
    UART1_Write_Text(“ created: “)
    WordToStr(year, outstr)
    UART1_Write_Text(outstr)
    ByteToStr(month_, outstr)
    UART1_Write_Text(outstr)
    WordToStr(day, outstr)

    UART1_Write_Text(outstr)

    WordToStr(hour_, outstr)
    UART1_Write_Text(outstr)
    WordToStr(minute_, outstr)

    UART1_Write_Text(outstr)
```

```
'--- file has been found - get its modified date
    Cf_Fat_Get_File_Date_Modified(year, month_, day, hour_, minute_)
    UART1_Write_Text(" modified: ")
    WordToStr(year, outstr)
    UART1_Write_Text(outstr)
    ByteToStr(month_, outstr)
    UART1_Write_Text(outstr)
    WordToStr(day, outstr)
    UART1_Write_Text(outstr)
    WordToStr(hour_, outstr)
    UART1_Write_Text(outstr)
    WordToStr(minute_, outstr)
    UART1_Write_Text(outstr)

    '--- get file size
    fsize = Cf_Fat_Get_File_Size
    LongIntToStr(fsize, outstr)
    UART1_Write_Line(outstr)
  else
    '--- file was not found - signal it
    UART1_Write(0x55)
    Delay_ms(1000)
    UART1_Write(0x55)
  end if
end sub


'-------------- Tries to create a swap file, whose size will be at least 100
'               sectors (see Help for details)
sub procedure M_Create_Swap_File()
  dim i as word

  for i=0 to 511
    Buffer[i] = i
  next i

  size = Cf_Fat_Get_Swap_File(5000, "mikroE.txt", 0x20)    ' see help on this function
for details

  if (size <> 0) then
    LongIntToStr(size, err_txt)
    UART1_Write_Line(err_txt)

    for i=0 to 4999
      Cf_Write_Sector(size, Buffer)
      Inc(size)
      UART1_Write(".")
    next i
  end if

end sub
```

```
'-------------- Main. Uncomment the function(s) to test the desired operation(s)
main:
  err_txt = "FAT16 not found"
  file_contents = "XX CF FAT16 library by Anton Rieckert"
  filename = "MIKRO00xTXT"

  #define COMPLETE_EXAMPLE        ' comment this line to make simpler/smaller example
  ADPCFG = 0xFFFF                 ' disable A/D inputs

  ' Initialize UART1 module
  UART1_Init(19200)
  Delay_ms(10)

  UART1_Write_Line("dsPIC-Started")         ' dsPIC present report

  ' --- Init the FAT library
  ' --- use Cf_Fat_QuickFormat instead of init routine if a format is needed
  if Cf_Fat_Init() = 0 then
    Delay_ms(2000)                    ' wait for a while until the card is stabilized
                                      ' period depends on used CF card

    '--- Test start
    UART1_Write_Line("Test Start.")
    M_Create_New_File()
    #IFDEF COMPLETE_EXAMPLE
    M_Create_Multiple_Files()
    M_Open_File_Rewrite()
    M_Open_File_Append()
    M_Open_File_Read()
    M_Delete_File()
    M_Test_File_Exist()
    M_Create_Swap_File()
    #ENDIF
    UART1_Write_Line("Test End.")
  else
   UART1_Write_Line(err_txt)      ' Note: Cf_Fat_Init tries to initialize a card more
than once.
                                  '  If card is not present, initialization may last
longer (depending on clock speed)
  end if
end.
```

HW Connection



Pin diagram of CF memory card

ECAN Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides a library (driver) for working with the dsPIC33FJ and pic24HJ ECAN module.

ECAN is a very robust protocol that has error detection and signalling, self–checking and fault confinement. Faulty ECAN data and remote frames are re-transmitted automatically, similar to the Ethernet.

Data transfer rates depend on distance. For example, 1 Mbit/s can be achieved at network lengths below 40m while 250 Kbit/s can be achieved at network lengths below 250m. The greater distance the lower maximum bitrate that can be achieved . The lowest bitrate defined by the standard is 200Kbit/s. Cables used are shielded twisted pairs.

ECAN supports two message formats:

- Standard format, with 11 identifier bits, and
- Extended format, with 29 identifier bits

ECAN message format and DMA RAM buffer definiton can be found in the `ECan_Defs.mbas` header file located in the ECAN project folder. Read this file carefully and make appropriate adjustments for mcu in use. Also, if a new project is to be created this file has to be copied, adjusted and included into the project via include pragma directive with corresponding Search Path updating.

Important:

- ECAN buffers are located in DMA RAM, so two DMA channels are used for message transfer, one for each direction (ECAN->DMA RAM, DMA RAM->ECAN). See the ECANxDmaChannelInit routine.
- Consult CAN standard about CAN bus termination resistance.
- CAN library routines require you to specify the module you want to use. To select the desired CAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.
- Number of CAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

Library Routines

- ECANxDmaChannelInit
- ECANxSetOperationMode
- ECANxGetOperationMode
- ECANxInitialize
- ECANxSelectTxBuffers
- ECANxFilterDisable
- ECANxFilterEnable
- ECANxSetBufferSize
- ECANxSetBaudRate
- ECANxSetMask
- ECANxSetFilter
- ECANxRead
- ECANxWrite

ECANxDmaChannelInit

Prototype	`sub function ECANxDmaChannelInit(dim DmaChannel as word, dim ChannelDir as word, dim DmaRamBuffAdd as word) as word`
Description	The function preforms initialization of the DMA module for ECAN.
Parameters	- `DmaChannel`: DMA Channel number. Valid values: `0..7`. - `ChannelDir`: transfer direction. Valid values: `1` (DMA RAM to peripheral) and `0` (peripheral to DMA RAM). - `DmaRamBuffAdd`: DMA RAM buffer address. DMA RAM location is MCU dependent, refer to datasheet for valid address range.
Returns	- `0` - if DMA channel parameter is valid - `0x0001` - if DMA channel is already in use (busy) - `0xFFFF` - if DMA channel parameter is invalid
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus.
Example	`' channel 0 will transfer 8 words from dma ram at 0x4000 to ECAN1` `ECAN1DmaChannelInit(0, 1, 0x4000)`
Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxSetOperationMode

Prototype	`sub procedure ECANxSetOperationMode(dim mode as word, dim WAIT as word)`
Description	Sets the ECAN module to requested mode.
Parameters	- `mode`: ECAN module operation mode. Valid values: `ECAN_OP_MODE` constants. See ECAN_OP_MODE constants. - `WAIT`: ECAN mode switching verification request. If `WAIT == 0`, the call is non-blocking. The function does not verify if the ECAN module is switched to requested mode or not. Caller must use ECANxGetOperationMode to verify correct operation mode before performing mode specific operation. If `WAIT != 0`, the call is blocking – the function won't "return" until the requested mode is set and no additional verification is necessary.
Returns	Nothing.
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus.
Example	`'set the ECAN1 module into configuration mode (wait inside ECAN1SetOperationMode until this mode is set)` `ECAN1SetOperationMode(_ECAN_MODE_CONFIG, 0xFF)`
Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxGetOperationMode

Prototype	`sub function ECANxGetOperationMode() as word`
Description	The function returns current operation mode of the ECAN module. See ECAN_OP_MODE constants or device datasheet for operation mode codes.
Parameters	None.
Returns	Current operation mode.
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus.
Example	see below
Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

Example:

```
' check whether the ECAN1 module is in Normal mode and if it is do
something.
if (ECAN1GetOperationMode() = _ECAN_MODE_NORMAL) then
  ...
end if
```

ECANxInitialize

Prototype	`sub procedure ECANxInitialize(dim SJW, BRP, PHSEG1, PHSEG2, PROPSEG, ECAN_CONFIG_FLAGS as word)`
Description	Initializes the ECAN module. The internal ECAN module is set to: - Disable ECAN capture - Continue ECAN operation in Idle mode - Abort all pending transmissions - Clear all transmit control registers - Fcan clock : Fcy (Fosc/2) - Baud rate is set according to given parameters - ECAN mode is set to Normal - Filter and mask registers remain unchanged `SAM, SEG2PHTS, WAKFIL` and `DBEN` bits are set according to the `ECAN_CONFIG_FLAGS` value.
Parameters	- `SJW` as defined in MCU's datasheet (ECAN Module) - `BRP` as defined in MCU's datasheet (ECAN Module) - `PHSEG1` as defined in MCU's datasheet (ECAN Module) - `PHSEG2` as defined in MCU's datasheet (ECAN Module) - `PROPSEG` as defined in MCU's datasheet (ECAN Module) - `ECAN_CONFIG_FLAGS` ECAN module configuration flags. Each bit corresponds to the appropriate ECAN module parameter. Should be formed out of predefined ECAN flag constants. See ECAN_CONFIG_FLAGS constants.
Returns	Nothing.
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus.

Example

```
' initialize the ECAN1 module with appropriate baud rate and message
acceptance flags along with the sampling rules
dim ecan_config_flags as word
...
ecan_config_flags = _ECAN_CONFIG_SAMPLE_THRICE and ' Form value to be used
                    _ECAN_CONFIG_PHSEG2_PRG_ON and ' with ECANInitialize
                    _ECAN_CONFIG_XTD_MSG           and
                    _ECAN_CONFIG_MATCH_MSG_TYPE    and
                    _ECAN_CONFIG_LINE_FILTER_OFF

ECAN1Initialize(1, 3, 3, 3, 1, ecan_config_flags) ' initialize the ECAN1 module
```

Notes	- ECAN mode NORMAL will be set on exit. - ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxSelectTxBuffers

Prototype	`sub function ECANxSelectTxBuffers(dim txselect as word) as word`
Description	The function designates the ECAN module's transmit buffers.
Parameters	- `txselect`: transmit buffer select. By setting bits in the txselect lower byte corresponding buffers are enabled for transmition. The ECAN module supports up to 8 transmit buffers. Also, by clearing bits in the txselect lower byte corresponding buffers are enabled for reception.
Returns	- `0` - if input parameter is valid - `0xFFFF` - if input parameter is invalid
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be initialized. See the ECANxInitialize routine.
Example	`' Buffers 0 and 2 are enabled for transmition:` `ECAN1SelectTxBuffers(0x0005)`
Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxFilterDisable

Prototype	`sub procedure ECANxFilterDisable(dim fltdis as word)`
Description	The function disables receive filters.
Parameters	- `fltdis`: filter disable selection parameter. Each bit corresponds to appropriate filter. By setting bit the corresponding filter is to be disabled.
Returns	Nothing.
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be initialized. See the ECANxInitialize routine.
Example	`' Filters 0, 4, 8, 12 are to be disabled:` `ECAN1FilterDisable(0x1111)`
Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxFilterEnable

Prototype	`sub procedure ECANxFilterEnable(dim flten as word)`
Description	The function enables receive filters.
Parameters	- `flten`: filter enable selection parameter. Each bit corresponds to appropriate filter. By setting bit the corresponding filter will be enabled.
Returns	Nothing.
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be initialized. See the ECANxInitialize routine.
Example	`' Filters 0, 4, 8, 12 are to be enabled:` `ECAN1FilterEnable(0x1111)`
Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxSetBufferSize

Prototype	`sub function ECANxSetBufferSize(dim Ecan1BuffSize as word) as word`
Description	The function configures the total number of receive and transmit buffers in DMA RAM.
Parameters	- `Ecan1BuffSize`: Number of ECAN DMA RAM receive and transmit buffers. Valid values: 4, 6, 8, 12, 16, 24, 32. Each buffer is 16 bytes long.
Returns	- `0` - if input parameter is valid - `0xFFFF` - if input parameter is invalid
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be initialized. See the ECANxInitialize routine.
Example	`' DMA RAM will have 16 rx+tx buffers` `ECAN1SetBufferSize(16)`
Notes	- The same value should be used for DMA RAM buffer definition in the `ECan_Defs.mbas` header file located in the ECAN project folder. - ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxSetBaudRate

Prototype	`sub procedure ECANxSetBaudRate(dim SJW, BRP, PHSEG1, PHSEG2, PROPSEG, ECAN_CONFIG_FLAGS as word)`
Description	Sets ECAN module baud rate. Due to complexity of the ECAN protocol, you can not simply force the bps value. Instead, use this function when ECAN is in Config mode. Refer to datasheet for details. `SAM`, `SEG2PHTS` and `WAKFIL` bits are set according to the `ECAN_CONFIG_FLAGS` value.
Parameters	- `SJW` as defined in MCU's datasheet (ECAN Module) - `BRP` as defined in MCU's datasheet (ECAN Module) - `PHSEG1` as defined in MCU's datasheet (ECAN Module) - `PHSEG2` as defined in MCU's datasheet (ECAN Module) - `PROPSEG` as defined in MCU's datasheet (ECAN Module) - `ECAN_CONFIG_FLAGS` ECAN module configuration flags. Each bit corresponds to the appropriate ECAN module parameter. Should be formed out of predefined ECAN flag constants. See ECAN_CONFIG_FLAGS constants
Returns	Nothing.
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be in Config mode, otherwise the function will be ignored. See ECANxSetOperationMode.

Example

```
' set required baud rate and sampling rules
dim ecan_config_flags as word
...
ECAN1SetOperationMode(_ECAN_MODE_CONFIG,0xFF)     ' set CONFIGURATION mode (ECAN1 module mast be in config mode for baud rate settings)

ecan_config_flags = _ECAN_CONFIG_SAMPLE_THRICE and ' Form value to be used
                    _ECAN_CONFIG_PHSEG2_PRG_ON and ' with ECAN1SetBaudRate
                    _ECAN_CONFIG_XTD_MSG           and
                    _ECAN_CONFIG_MATCH_MSG_TYPE    and
                    _ECAN_CONFIG_LINE_FILTER_OFF

ECAN1SetBaudRate(1, 3, 3, 3, 1, ecan_config_flags)  ' set ECAN1 module baud rate
```

Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxSetMask

Prototype	`sub procedure ECANxSetMask(dim ECAN_MASK as word, dim val as longint, dim ECAN_CONFIG_FLAGS as word)`
Description	The function configures appropriate mask for advanced message filtering.

Parameters	- `ECAN_MASK`: ECAN module mask number. Valid values: ECAN_MASK constants. See ECAN_MASK constants.
	- `val`: mask register value. This value is bit-adjusted to appropriate buffer mask registers
	- `ECAN_CONFIG_FLAGS`: selects type of messages to filter. Valid values:
	  - `_ECAN_CONFIG_ALL_VALID_MSG`,
	  - `_ECAN_CONFIG_MATCH_MSG_TYPE & _ECAN_CONFIG_STD_MSG`,
	  - `_ECAN_CONFIG_MATCH_MSG_TYPE & _ECAN_CONFIG_XTD_MSG`.
	
	See ECAN_CONFIG_FLAGS constants.

Returns	Nothing.
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be in Config mode, otherwise the function will be ignored. See ECANxSetOperationMode.

Example	`' set appropriate filter mask and message type value`
	`ECAN1SetOperationMode(_ECAN_MODE_CONFIG,0xFF)    ' set CONFIGURATION mode (ECAN1 module must be in config mode for mask settings)`
	
	`' Set all mask0 bits to 1 (all filtered bits are relevant):`
	`' Note that -1 is just a cheaper way to write 0xFFFFFFFF.`
	`' Complement will do the trick and fill it up with ones.`
	`ECAN1SetMask(_ECAN_MASK_0, -1, _ECAN_CONFIG_MATCH_MSG_TYPE and _ECAN_CONFIG_XTD_MSG)`

Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxSetFilter

Prototype	`sub procedure ECANxSetFilter(dim ECAN_FILTER as word, dim val as longint, dim ECAN_FILTER_MASK as word, dim ECAN_FILTER_RXBUFF as word, dim ECAN_CONFIG_FLAGS as word)`
Description	The function configures and enables appropriate message filter.
Parameters	- `ECAN_FILTER`: ECAN module filter number. Valid values: `ECAN_FILTER` constants. See ECAN_FILTER constants. - `val`: filter register value. This value is bit-adjusted to appropriate filter registers - `ECAN_FILTER_MASK`: mask register corresponding to filter. Valid values: `ECAN_MASK` constants. See ECAN_MASK constants. - `ECAN_FILTER_RXBUFF`: receive buffer corresponding to filter. Valid values: `ECAN_RX_BUFFER` constants. See ECAN_RX_BUFFER constants. - `ECAN_CONFIG_FLAGS`: selects type of messages to filter. Valid values: `_ECAN_CONFIG_XTD_MSG` and `_ECAN_CONFIG_STD_MSG`. See ECAN_CONFIG_FLAGS constants.
Returns	Nothing.
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be in Config mode, otherwise the function will be ignored. See ECANxSetOperationMode.

Example	`' set appropriate filter value and message type`
	`ECAN1SetOperationMode(_ECAN_MODE_CONFIG,0xFF)      ' set CONFIGURATION mode (ECAN1 module must be in config mode for filter settings)`
	
	`' Set id of filter 10 to 3, mask2, receive buffer 7, extended messages:`
	`ECAN1SetFilter(_ECAN_FILTER_10, 3, _ECAN_MASK_2, _ECAN_RX_BUFFER_7, _ECAN_CONFIG_XTD_MSG)`

Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxRead

Prototype	`sub function ECANxRead(dim byref id as longint, dim byref data as byte[8], dim byref dataLen as word, dim byref ECAN_RX_MSG_FLAGS as word) as word`
Description	If at least one full Receive Buffer is found, it will be processed in the following way: - Message ID is retrieved and stored to location pointed by the `id` pointer - Message data is retrieved and stored to array pointed by the `data` pointer - Message length is retrieved and stored to location pointed by the `dataLen` pointer - Message flags are retrieved and stored to location pointed by the `ECAN_RX_MSG_FLAGS` pointer
Parameters	- `id`: message identifier address - `data`: an array of bytes up to 8 bytes in length - `dataLen`: data length address - `ECAN_RX_MSG_FLAGS`: message flags address. For message receive flags format refer to the `ECAN_RX_MSG_FLAGS` constants. See ECAN_RX_MSG_FLAGS constants.
Returns	- `0` if none of Receive Buffers is full - `0xFFFF` if at least one of Receive Buffers is full (message received)
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be in a mode in which receiving is possible. See ECANxSetOperationMode.

Example

```
' check the ECAN1 module for received messages. If any was received do something.
dim msg_rcvd, rx_flags, data_len as word
    data as byte[8]
    msg_id as longint
...
ECAN1SetOperationMode(_ECAN_MODE_NORMAL,0xFF)       ' set NORMAL mode (ECAN1 module must be in a mode in which receiving is possible)
...
rx_flags = 0                                         ' clear message flags
if (msg_rcvd = ECAN1Read(msg_id, data, data_len, rx_flags)) then
  ...
end if
```

Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECANxWrite

Prototype	`sub function ECANxWrite(dim id as longint, dim byref Data as byte[8], dim DataLen, ECAN_TX_MSG_FLAGS as word) as word`
Description	If at least one empty Transmit Buffer is found, the function sends message in the queue for transmission.
Parameters	- `id`: ECAN message identifier. Valid values: all 11 or 29 bit values, depending on message type (standard or extended) - `Data`: data to be sent - `DataLen`: data length. Valid values: `0..8` - `ECAN_TX_MSG_FLAGS`: message flags. Valid values: `ECAN_TX_MSG_FLAGS` constants. See ECAN_TX_MSG_FLAGS constants.
Returns	- `0` if all Transmit Buffers are busy - `0xFFFF` if at least one Transmit Buffer is empty and available for transmition
Requires	The ECAN routines are supported only by MCUs with the ECAN module. Microcontroller must be connected to ECAN transceiver which is connected to the ECAN bus. The ECAN module must be in a mode in which transmission is possible. See ECANxSetOperationMode.

Example

```
' send message extended ECAN message with appropriate ID and data
dim tx_flags as word
    data as byte[8]
    msg_id as longint
...
ECAN1SetOperationMode(_ECAN_MODE_NORMAL,0xFF)      ' set NORMAL mode (ECAN1 must be in a mode in which transmission is possible)

tx_flags = _ECAN_TX_PRIORITY_0   and
            _ECAN_TX_XTD_FRAME    and
            _ECAN_TX_NO_RTR_FRAME                   ' set message flags
ECAN1Write(msg_id, data, 1, tx_flags)
```

Notes	- ECAN library routine require you to specify the module you want to use. To select the desired ECAN module, simply change the letter **x** in the routine prototype for a number from **1** to **2**. - Number of ECAN modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

ECAN Constants

There is a number of constants predefined in the ECAN library. You need to be familiar with them in order to be able to use the library effectively. Check the example at the end of the chapter.

ECAN_OP_MODE Constants

The `ECAN_OP_MODE` constants define ECAN operation mode. The routine ECANxSetOperationMode expect one of these as their argument:

Copy Code To Clipboard

```
const
    _ECAN_MODE_BITS        as word = 0x00E0    ' Use this to access opmode  bits
    _ECAN_MODE_NORMAL      as word = 0x00
    _ECAN_MODE_DISABLE     as word = 0x01
    _ECAN_MODE_LOOP        as word = 0x02
    _ECAN_MODE_LISTEN      as word = 0x03
    _ECAN_MODE_CONFIG      as word = 0x04
    _ECAN_MODE_LISTEN_ALL  as word = 0x07
```

ECAN_CONFIG_FLAGS Constants

The `ECAN_CONFIG_FLAGS` constants define flags related to the ECAN module configuration. The routines ECANxInitialize and ECANxSetBaudRate expect one of these (or a bitwise combination) as their argument:

Copy Code To Clipboard

```
const
    _ECAN_CONFIG_DEFAULT          as word = 0xFF    ' 11111111

    _ECAN_CONFIG_PHSEG2_PRG_BIT   as word = 0x01
    _ECAN_CONFIG_PHSEG2_PRG_ON    as word = 0xFF    ' XXXXXXX1
    _ECAN_CONFIG_PHSEG2_PRG_OFF   as word = 0xFE    ' XXXXXXX0

    _ECAN_CONFIG_LINE_FILTER_BIT  as word = 0x02
    _ECAN_CONFIG_LINE_FILTER_ON   as word = 0xFF    ' XXXXXX1X
    _ECAN_CONFIG_LINE_FILTER_OFF  as word = 0xFD    ' XXXXXX0X

    _ECAN_CONFIG_SAMPLE_BIT       as word = 0x04
    _ECAN_CONFIG_SAMPLE_ONCE      as word = 0xFF    ' XXXXX1XX
    _ECAN_CONFIG_SAMPLE_THRICE    as word = 0xFB    ' XXXXX0XX

    _ECAN_CONFIG_MSG_TYPE_BIT     as word = 0x08
    _ECAN_CONFIG_STD_MSG          as word = 0xFF    ' XXXX1XXX
    _ECAN_CONFIG_XTD_MSG          as word = 0xF7    ' XXXX0XXX

    _ECAN_CONFIG_MATCH_TYPE_BIT   as word = 0x20
    _ECAN_CONFIG_ALL_VALID_MSG    as word = 0xDF    ' XX0XXXXX
    _ECAN_CONFIG_MATCH_MSG_TYPE   as word = 0xFF    ' XX1XXXXX
```

You may use bitwise `and` to adjust the appropriate flags. For example:

Copy Code To Clipboard

```
init   = _ECAN_CONFIG_SAMPLE_THRICE    and
         _ECAN_CONFIG_PHSEG2_PRG_ON    and
         _ECAN_CONFIG_STD_MSG          and
         _ECAN_CONFIG_MATCH_MSG_TYPE   and
         _ECAN_CONFIG_LINE_FILTER_OFF
...
ECAN1Initialize(1, 1, 3, 3, 1, init)    ' initialize ECAN1
```

ECAN_TX_MSG_FLAGS Constants

`ECAN_TX_MSG_FLAGS` are flags related to transmission of ECAN message. The routine ECANxWrite expect one of these (or a bitwise combination) as their argument:

```
const
    _ECAN_TX_PRIORITY_BITS as word = 0x03
    _ECAN_TX_PRIORITY_0    as word = 0xFC    ' XXXXXX00
    _ECAN_TX_PRIORITY_1    as word = 0xFD    ' XXXXXX01
    _ECAN_TX_PRIORITY_2    as word = 0xFE    ' XXXXXX10
    _ECAN_TX_PRIORITY_3    as word = 0xFF    ' XXXXXX11

    _ECAN_TX_FRAME_BIT     as word = 0x08
    _ECAN_TX_STD_FRAME     as word = 0xFF    ' XXXXX1XX
    _ECAN_TX_XTD_FRAME     as word = 0xF7    ' XXXXX0XX

    _ECAN_TX_RTR_BIT       as word = 0x40
    _ECAN_TX_NO_RTR_FRAME  as word = 0xFF    ' X1XXXXXX
    _ECAN_TX_RTR_FRAME     as word = 0xBF    ' X0XXXXXX
```

You may use bitwise `and` to extract received message status. For example:

Copy Code To Clipboard

```
' form value to be used with CANSendMessage:
send_config  = _ECAN_TX_PRIORITY_0 and
               _ECAN_TX_XTD_FRAME  and
               _ECAN_TX_NO_RTR_FRAME
...
ECAN1SendMessage(id, data, 1, send_config)
```

ECAN_RX_MSG_FLAGS Constants

`ECAN_RX_MSG_FLAGS` are flags related to reception of ECAN message. If a particular bit is set then corresponding meaning is TRUE or else it will be FALSE.

```
const
    _ECAN_RX_FILTER_BITS as word = 0x000F ' Use this to access filter bits
    _ECAN_RX_FILTER_0    as word = 0x00   ' filter0 match
    _ECAN_RX_FILTER_1    as word = 0x01   ' filter1 match
    _ECAN_RX_FILTER_2    as word = 0x02   ' ...
    _ECAN_RX_FILTER_3    as word = 0x03
    _ECAN_RX_FILTER_4    as word = 0x04
    _ECAN_RX_FILTER_5    as word = 0x05
    _ECAN_RX_FILTER_6    as word = 0x06
    _ECAN_RX_FILTER_7    as word = 0x07
    _ECAN_RX_FILTER_8    as word = 0x08
    _ECAN_RX_FILTER_9    as word = 0x09
    _ECAN_RX_FILTER_10   as word = 0x0A
    _ECAN_RX_FILTER_11   as word = 0x0B
    _ECAN_RX_FILTER_12   as word = 0x0C
    _ECAN_RX_FILTER_13   as word = 0x0D
    _ECAN_RX_FILTER_14   as word = 0x0E   ' ...
    _ECAN_RX_FILTER_15   as word = 0x0F   ' filter15 match

    _ECAN_RX_OVERFLOW    as word = 0x10   ' Set if Overflowed else cleared
    _ECAN_RX_INVALID_MSG as word = 0x20   ' Set if invalid else cleared
    _ECAN_RX_XTD_FRAME   as word = 0x40   ' Set if XTD message else cleared
    _ECAN_RX_RTR_FRAME   as word = 0x80   ' Set if RTR message else cleared
```

You may use bitwise `and` to extract received message status. For example:

Copy Code To Clipboard

```
if (MsgFlag and _ECAN_RX_OVERFLOW <> 0) then
  ...
  ' Receiver overflow has occurred.
  ' We have lost our previous message.
end if
```

ECAN_MASK Constants

The `ECAN_MASK` constants define mask codes. The routine ECANxSetMask expect one of these as their argument:

Copy Code To Clipboard

```
const
    _ECAN_MASK_0 as word = 0
    _ECAN_MASK_1 as word = 1
    _ECAN_MASK_2 as word = 2
```

ECAN_FILTER Constants

The `ECAN_FILTER` constants define filter codes. The routine ECANxSetFilter expect one of these as their argument:

Copy Code To Clipboard

```
const
    _ECAN_FILTER_0  as word =  0
    _ECAN_FILTER_1  as word =  1
    _ECAN_FILTER_2  as word =  2
    _ECAN_FILTER_3  as word =  3
    _ECAN_FILTER_4  as word =  4
    _ECAN_FILTER_5  as word =  5
    _ECAN_FILTER_6  as word =  6
    _ECAN_FILTER_7  as word =  7
    _ECAN_FILTER_8  as word =  8
    _ECAN_FILTER_9  as word =  9
    _ECAN_FILTER_10 as word = 10
    _ECAN_FILTER_11 as word = 11
    _ECAN_FILTER_12 as word = 12
    _ECAN_FILTER_13 as word = 13
    _ECAN_FILTER_14 as word = 14
    _ECAN_FILTER_15 as word = 15
```

ECAN_RX_BUFFER Constants

The `ECAN_RX_BUFFER` constants define RX bufer codes codes. The routine ECANxSetFilter expect one of these as their argument:

Copy Code To Clipboard

```
const
    _ECAN_RX_BUFFER_0  as word =  0
    _ECAN_RX_BUFFER_1  as word =  1
    _ECAN_RX_BUFFER_2  as word =  2
    _ECAN_RX_BUFFER_3  as word =  3
    _ECAN_RX_BUFFER_4  as word =  4
    _ECAN_RX_BUFFER_5  as word =  5
    _ECAN_RX_BUFFER_6  as word =  6
    _ECAN_RX_BUFFER_7  as word =  7
    _ECAN_RX_BUFFER_8  as word =  8
    _ECAN_RX_BUFFER_9  as word =  9
    _ECAN_RX_BUFFER_10 as word = 10
    _ECAN_RX_BUFFER_11 as word = 11
    _ECAN_RX_BUFFER_12 as word = 12
    _ECAN_RX_BUFFER_13 as word = 13
    _ECAN_RX_BUFFER_14 as word = 14
    _ECAN_RX_BUFFER_15 as word = 15
```

Library Example

The example demonstrates ECAN protocol. The 1st node initiates the communication with the 2nd node by sending some data to its address. The 2nd node responds by sending back the data incremented by 1. The 1st node then does the same and sends incremented data back to the 2nd node, etc.

Code for the first ECAN node:

Copy Code To Clipboard

```
program ECan_1st

include ECAN_Defs

dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as word ' can flags
    Rx_Data_Len   as word                              ' received data length in bytes
    RxTx_Data     as byte[8]                           ' can rx/tx data buffer
    Msg_Rcvd      as word                              ' reception flag
    Rx_ID         as longint

const ID_1st as longint = 12111
const ID_2nd as longint = 3                              ' node IDs


sub procedure C1Interrupt() org 0x005A                   ' ECAN event iterrupt

  IFS2.C1IF = 0                        ' clear ECAN interrupt flag
  if(C1INTF.TBIF <> 0) then            ' was it tx interrupt?
    C1INTF.TBIF = 0                    ' if yes clear tx interrupt flag
  end if

  if(C1INTF.RBIF <> 0) then            ' was it rx interrupt?
    C1INTF.RBIF = 0                    ' if yes clear rx interrupt flag
  end if
end sub

main:
  ' Set PLL : Fosc = ((Fin/PLLPRE)*PLLDIV)/PLLPOST ; (((10MHz/2)*32)/4) = 20MHz
  ' refer the pic33 family datasheet for more details
  CLKDIV = CLKDIV and 0xFFE0   ' CLKDIVbits.PLLPRE = 0
  PLLFBD = 0x1E                ' PLLFBDbits.PLLDIV = 0x1E
  CLKDIV = CLKDIV and 0xFF3F   ' CLKDIVbits.PLLPOST = 1
  CLKDIV = CLKDIV or 0x00C0

  AD1PCFGH = 0xFFFF                                   '
  AD1PCFGL = 0xFFFF                                   ' all ports digital I/O
  AD2PCFGL = 0xFFFF                                   '

' Clear Interrupt Flags

  IFS0 = 0
  IFS1 = 0
```

```
  IFS2 = 0
  IFS3 = 0
  IFS4 = 0

  ' Enable ECAN1 Interrupt

  IEC2.C1IE   = 1                                    ' enable ECAN1 interrupts
  C1INTE.TBIE = 1                                    ' enable ECAN1 tx interrupt
  C1INTE.RBIE = 1                                    ' enable ECAN1 rx interrupt

  PORTB    = 0                                     ' clear PORTB
  TRISB    = 0                                     ' set PORTB as output,
                                                   ' for received message data displaying

  Can_Init_Flags = 0                                 '
  Can_Send_Flags = 0                                 ' clear flags
  Can_Rcv_Flags  = 0                                 '

  Can_Send_Flags = _ECAN_TX_PRIORITY_0 and           ' form value to be used
                   _ECAN_TX_XTD_FRAME and            ' with CANSendMessage
                   _ECAN_TX_NO_RTR_FRAME

  Can_Init_Flags = _ECAN_CONFIG_SAMPLE_THRICE and    ' form value to be used
                   _ECAN_CONFIG_PHSEG2_PRG_ON and    ' with CANInitialize
                   _ECAN_CONFIG_XTD_MSG and
                   _ECAN_CONFIG_MATCH_MSG_TYPE and
                   _ECAN_CONFIG_LINE_FILTER_OFF

  RxTx_Data[0] = 9                                   ' set initial data to be sent
  ECAN1DmaChannelInit(0, 1, @ECAN1RxTxRAMBuffer)     ' init dma channel 0 for
                                                     ' dma to ECAN peripheral transfer
  ECAN1DmaChannelInit(2, 0, @ECAN1RxTxRAMBuffer)     ' init dma channel 2 for
                                                     ' ECAN peripheral to dma transfer
  ECAN1Initialize(1, 3, 3, 3, 1, Can_Init_Flags)     ' initialize ECAN
  ECAN1SetBufferSize(ECAN1RAMBUFFERSIZE)      ' set number of rx+tx buffers in DMA RAM

  ECAN1SelectTxBuffers(0x000F)                       ' select transmit buffers
                                            ' 0x000F = buffers 0:3 are transmit buffers
  ECAN1SetOperationMode(_ECAN_MODE_CONFIG,0xFF)      ' set CONFIGURATION mode

ECAN1SetMask(_ECAN_MASK_0,  -1,  _ECAN_CONFIG_MATCH_MSG_TYPE  and  _ECAN_CONFIG_XTD_MSG)
' set all mask1 bits to ones
  ECAN1SetMask(_ECAN_MASK_1, -1, _ECAN_CONFIG_MATCH_MSG_TYPE and _ECAN_CONFIG_XTD_MSG)
' set all mask2 bits to ones
  ECAN1SetMask(_ECAN_MASK_2, -1, _ECAN_CONFIG_MATCH_MSG_TYPE and _ECAN_CONFIG_XTD_MSG)
' set all mask3 bits to ones
 ECAN1SetFilter(_ECAN_FILTER_10, ID_2nd, _ECAN_MASK_2, _ECAN_RX_BUFFER_7, _ECAN_CONFIG_
XTD_MSG)  ' set id of filter10 to 2nd node ID
                                                          ' assign mask2 to filter10
                                                          ' assign buffer7 to filter10

ECAN1SetOperationMode(_ECAN_MODE_NORMAL, 0xFF)      ' set NORMAL mode
```

```
ECAN1Write(ID_1st, RxTx_Data, 1, Can_Send_Flags)        ' send initial message

  while TRUE                                                ' endless loop
     Msg_Rcvd = ECAN1Read(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)  ' receive
message
    if ((Rx_ID = ID_2nd) and (Msg_Rcvd <> 0)) <> 0 then ' if message received check id
      PORTB = RxTx_Data[0]                          ' id correct, output data at PORTB
      Inc(RxTx_Data[0])
      Delay_ms(10)
      ECAN1Write(ID_1st, RxTx_Data, 1, Can_Send_Flags)   ' send incremented data back
    end if
  wend
end.
```

Code for the second ECAN node:

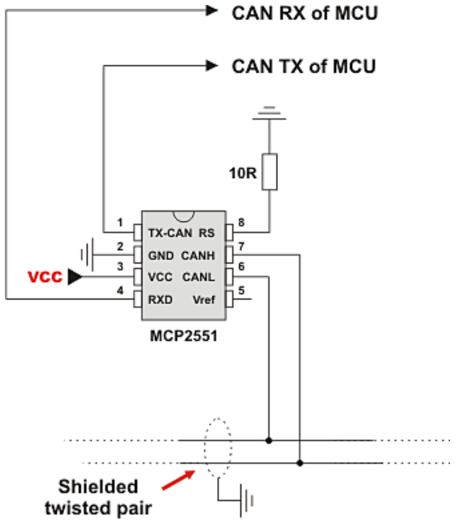
Copy Code To Clipboard

```
program ECAN_2nd

include ECAN_Defs

dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as word ' can flags
    Rx_Data_Len   as word                             ' received data length in bytes
    RxTx_Data     as byte[8]                          ' can rx/tx data buffer
    Msg_Rcvd      as word                             ' reception flag
    Rx_ID         as longint

const ID_1st as longint = 12111
const ID_2nd as longint = 3                                 ' node IDs


sub procedure C1Interrupt() org 0x005A                      ' ECAN event iterrupt

  IFS2.C1IF = 0                      ' clear ECAN interrupt flag
  if(C1INTF.TBIF <> 0) then          ' was it tx interrupt?
    C1INTF.TBIF = 0                  ' if yes clear tx interrupt flag
  end if

  if(C1INTF.RBIF <> 0) then          ' was it rx interrupt?
    C1INTF.RBIF = 0                  ' if yes clear rx interrupt flag
  end if
end sub

main:
  ' Set PLL : Fosc = ((Fin/PLLPRE)*PLLDIV)/PLLPOST ; (((10MHz/2)*32)/4) = 20MHz
  ' refer the pic33 family datasheet for more details
  CLKDIV = CLKDIV and 0xFFE0   ' CLKDIVbits.PLLPRE = 0
  PLLFBD = 0x1E                ' PLLFBDbits.PLLDIV = 0x1E
  CLKDIV = CLKDIV and 0xFF3F   ' CLKDIVbits.PLLPOST = 1
  CLKDIV = CLKDIV or 0x00C0
```

```
  AD1PCFGH = 0xFFFF                                 '
  AD1PCFGL = 0xFFFF                                 ' all ports digital I/O
  AD2PCFGL = 0xFFFF                                 '

  ' Clear Interrupt Flags

  IFS0 = 0
  IFS1 = 0
  IFS2 = 0
  IFS3 = 0
  IFS4 = 0

  ' Enable ECAN1 Interrupt

  IEC2.C1IE   = 1                                    ' enable ECAN1 interrupts
  C1INTE.TBIE = 1                                    ' enable ECAN1 tx interrupt
  C1INTE.RBIE = 1                                    ' enable ECAN1 rx interrupt

  PORTB    = 0                    ' clear PORTB
  TRISB    = 0                    ' set PORTB as output,
                                  ' for received message data displaying

  Can_Init_Flags = 0                                 '
  Can_Send_Flags = 0                                 ' clear flags
  Can_Rcv_Flags  = 0                                 '

  Can_Send_Flags = _ECAN_TX_PRIORITY_0 and           ' form value to be used
                   _ECAN_TX_XTD_FRAME and            ' with CANSendMessage
                   _ECAN_TX_NO_RTR_FRAME

  Can_Init_Flags = _ECAN_CONFIG_SAMPLE_THRICE and    ' form value to be used
                   _ECAN_CONFIG_PHSEG2_PRG_ON and    ' with CANInitialize
                   _ECAN_CONFIG_XTD_MSG and
                   _ECAN_CONFIG_MATCH_MSG_TYPE and
                   _ECAN_CONFIG_LINE_FILTER_OFF

  ECAN1DmaChannelInit(0, 1, @ECAN1RxTxRAMBuffer)     ' init dma channel 0 for
                                                     ' dma to ECAN peripheral transfer
  ECAN1DmaChannelInit(2, 0, @ECAN1RxTxRAMBuffer)     ' init dma channel 2 for
                                                     ' ECAN peripheral to dma transfer
  ECAN1Initialize(1, 3, 3, 3, 1, Can_Init_Flags)     ' initialize ECAN
  ECAN1SetBufferSize(ECAN1RAMBUFFERSIZE)    ' set number of rx+tx buffers in DMA RAM

  ECAN1SelectTxBuffers(0x000F)           ' select transmit buffers
                                         ' 0x000F = buffers 0:3 are transmit buffers
  ECAN1SetOperationMode(_ECAN_MODE_CONFIG,0xFF)      ' set CONFIGURATION mode

  ECAN1SetMask(_ECAN_MASK_0, -1, _ECAN_CONFIG_MATCH_MSG_TYPE and _ECAN_CONFIG_XTD_MSG)
' set all mask1 bits to ones
  ECAN1SetMask(_ECAN_MASK_1, -1, _ECAN_CONFIG_MATCH_MSG_TYPE and _ECAN_CONFIG_XTD_MSG)
' set all mask2 bits to ones
  ECAN1SetMask(_ECAN_MASK_2, -1, _ECAN_CONFIG_MATCH_MSG_TYPE and _ECAN_CONFIG_XTD_MSG)
' set all mask3 bits to ones
 ECAN1SetFilter(_ECAN_FILTER_10, ID_1st, _ECAN_MASK_2, _ECAN_RX_BUFFER_7, _ECAN_CONFIG_
XTD_MSG) ' set id of filter10 to 1st node ID
                                                        ' assign buffer7 to filter10
  ECAN1SetOperationMode(_ECAN_MODE_NORMAL,0xFF)         ' set NORMAL mode
```

```
while TRUE
      Msg_Rcvd = ECAN1Read(Rx_ID, RxTx_Data, Rx_Data_Len, Can_Rcv_Flags)    ' receive
message
    if ((Rx_ID = ID_1st) and (Msg_Rcvd <> 0) <> 0) then ' if message received check id
      PORTB = RxTx_Data[0]                           ' id correct, output data at PORTB
      Inc(RxTx_Data[0])                              ' increment received data
      ECAN1Write(ID_2nd, RxTx_Data, 1, Can_Send_Flags)    ' send incremented data back
    end if
  wend
end.
```

HW Connection



Example of interfacing ECAN transceiver with MCU and bus

EEPROM Library

EEPROM data memory is available with a number of dsPIC30 family and some PIC24 family MCU's. The mikroBasic PRO for dsPIC30/33 and PIC24 includes a library for comfortable work with MCU's internal EEPROM.

Important: Only 24F04KA201 and 24F16KA102 of PIC24 family of MCUs have EEPROM memory.

Library Routines

- EEPROM_Erase
- EEPROM_Erase_Block
- EEPROM_Read
- EEPROM_Write
- EEPROM_Write_Block

EEPROM_Erase

Prototype	`sub procedure EEPROM_Erase(dim address as longint)`
Description	Erases a single (16-bit) location from EEPROM memory.
Parameters	- `address`: address of the EEPROM memory location to be erased.
Returns	Nothing.
Requires	Nothing.

Example

```
dim eeAddr as longint
...
eeAddr = 0x7FFC80
EEPROM_Erase(eeAddr)
```

Notes	CPU is not halted for the Data Erase cycle. The user can poll WR bit, use NVMIF or Timer IRQ to detect the end of erase sequence.

EEPROM_Erase_Block

Prototype	`sub procedure EEPROM_Erase_Block(dim address as longint)`
Description	Erases one EEPROM row from EEPROM memory; For dsPIC30 family it is 16 words long, for 24F04KA201 and 24F16KA102 family it is 8 words long.
Parameters	- `address`: starting address of the EEPROM memory block to be erased.
Returns	Nothing.
Requires	Nothing.

Example

```
dim eeAddr as longint
...
eeAddr = 0x7FFC20
EEPROM_Erase_Block(eeAddr)
```

Notes	CPU is not halted for the Data Erase cycle. The user can poll WR bit, use NVMIF or Timer IRQ to detect the end of erase sequence.

EEPROM_Read

Prototype	`sub function EEPROM_Read(dim address as longint) as word`
Description	Reads data from specified `address`.
Parameters	- `address`: address of the EEPROM memory location to be read.
Returns	Word from the specified address.
Requires	It is the user's responsibility to obtain proper address parity (in this case, even).

Example

```
dim eeAddr as longint
      temp as word
...
eeAddr = 0x7FFC20
temp = EEPROM_Read(eeAddr)
```

Notes	None.

EEPROM_Write

Prototype	`sub procedure EEPROM_Write(dim address as longint, dim data_ as word)`
Description	Writes data to specified `address`.
Parameters	- `address`: address of the EEPROM memory location to be written. - `data`: data to be written.
Returns	Nothing.
Requires	Nothing.

Example

```
dim wrAddr as longint
    eeData as word
  ...
  eeData = 0xAAAA
  wrAddr = 0x7FFC30
  EEPROM_Write(wrAddr, eeData)
```

Notes	Specified memory location will be erased before writing starts.

EEPROM_Write_Block

Prototype	`sub procedure EEPROM_Write_Block(dim address as longint, dim byref data_ as word[100])`
Description	Writes one EEPROM row (16 words block) of data.
Parameters	- `address`: starting address of the EEPROM memory block to be written. - `data`: data block to be written.
Returns	Nothing.
Requires	It is the user's responsibility to maintain proper address alignment. In this case, address has to be a multiply of 32, which is the size (in bytes) of one row of MCU's EEPROM memory.

Example

```
dim  wrAddr as longint
     data as string[16]
...
wrAddr = 0x7FFC20
data = "mikroElektronika"
EEPROM_Write_Block(wrAddr, data)
```

Notes	- Specified memory block will be erased before writing starts. - This routine is not applicable to the 24F04KA201 and 24F16KA102 family of MCUs, due to the architecture specifics.

Library Example

This project demonstrates usage of EEPROM library functions for dsPIC30F4013. Each EEPROM (16-bit) location can be written to individually, or in 16-word blocks, which is somewhat faster than the former. If Writing in blocks, EEPROM data start address must be a multiply of 16. Please read Help for more details on the library functions!

Copy Code To Clipboard

```
program Eeprom
dim eeData, i as word
    eeAddr as longword
    dArr as word[16]
```

```
  main:

  ADPCFG = 0xFFFF                     ' Disable analog inputs

  TRISB = 0                           ' PORTB as output
  LATB = 0xFFFF
  eeAddr = 0x7FFC00                   ' Start address of EEPROM
  eeData = 0                          ' Data to be written

  while (eeData <= 0x00FF)
    Eeprom_Write(eeAddr, eeData)      ' Write data into EEPROM
    Inc(eeData)
    while (WR_bit)                    ' Wait for write to finish,
      nop
    wend

    LATB = Eeprom_Read(eeAddr)        ' then, read the just-written data.
    eeAddr = eeAddr + 2               ' Next address of EEPROM memory location

    Delay_ms(100)
  wend

  Delay_ms(1000)                      ' Wait 1 second.

  eeData = 0xAAAA
  for i = 0 to 1                      ' Initializing array of 16 integers with data
    dArr[i] = eeData
    eeData  = not eeData
  next i

  Eeprom_Write_Block(0x7FFC20, dArr)  ' Write entire row of EEPROM data
  while(WR_bit)                       ' Wait for write to finish
    nop
  wend


  eeAddr = 0x7FFC20                   ' Address of EEPROM where reading should start
  for i = 0 to 15                     ' Read the data back
    LATB = Eeprom_Read(eeAddr)        ' and show it on PORTB
    eeAddr = eeAddr + 2               ' Next address of EEPROM memory location
    Delay_ms(500)
  next i
end.
```

Epson S1D13700 Graphic Lcd Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for working with Glcds based on Epson S1D13700 controller.

The S1D13700 Glcd is capable of displaying both text and graphics on an LCD panel. The S1D13700 Glcd allows layered text and graphics, scrolling of the display in any direction, and partitioning of the display into multiple screens. It includes 32K bytes of embedded SRAM display memory which is used to store text, character codes, and bit-mapped graphics.

The S1D13700 Glcd handles display controller functions including:

- Transferring data from the controlling microprocessor to the buffer memory
- Reading memory data, converting data to display pixels
- Generating timing signals for the LCD panel

The S1D13700 Glcd is designed with an internal character generator which supports 160, 5x7 pixel characters in internal mask ROM (CGROM) and 64, 8x8 pixel characters incharacter generator RAM (CGRAM).
When the CGROM is not used, up to 256, 8x16 pixel characters are supported in CGRAM.

External dependencies of the Epson S1D13700 Graphic Lcd Library

The following variables must be defined in all projects using S1D13700 Graphic Lcd library:	**Description:**	**Example:**
`dim S1D13700_DATA as byte sfr external`	System data bus.	`dim S1D13700_DATA at PORTD`
`dim S1D13700_WR as sbit sfr external`	Write signal.	`dim S1D13700_WR as sbit at LATC2_bit`
`dim S1D13700_RD as sbit sfr external`	Read signal.	`dim S1D13700_RD as sbit at LATC1_bit`
`dim S1D13700_A0 as sbit sfr external`	System Address pin.	`dim S1D13700_A0 as sbit at LATC0_bit`
`dim S1D13700_RES as sbit sfr external`	Reset signal.	`dim S1D13700_RES as sbit at LATC4_bit`
`dim S1D13700_CS as sbit sfr external`	Chip select.	`dim S1D13700_CS as sbit at LATC4_bit`
`dim S1D13700_DATA_Direction as byte sfr external`	Direction of the system data bus pins.	`dim S1D13700_DATA_Direction sbit at TRISD`
`dim S1D13700_WR_Direction as sbit sfr external`	Direction of the Write pin.	`dim S1D13700_WR_Direction as sbit at TRISC2_bit`
`dim S1D13700_RD_Direction as sbit sfr external`	Direction of the Read pin.	`dim S1D13700_RD_Direction as sbit at TRISC1_bit`
`dim S1D13700_A0_Direction as sbit sfr external`	Direction of the System Address pin.	`dim S1D13700_A0_Direction as sbit at TRISC2_bit`
`dim S1D13700_RES_Direction as sbit sfr external`	Direction of the Reset pin.	`dim S1D13700_RES_Direction as sbit at TRISC0_bit`
`dim S1D13700_CS_Direction as sbit sfr external`	Direction of the Chip select pin.	`dim S1D13700_CS_Direction as sbit at TRISC4_bit`

Library Routines

- S1D13700_Init
- S1D13700_Write_Command
- S1D13700_Write_Parameter
- S1D13700_Read_Parameter
- S1D13700_Fill
- S1D13700_GrFill
- S1D13700_TxtFill
- S1D13700_Display_GrLayer
- S1D13700_Display_TxtLayer
- S1D13700_Set_Cursor
- S1D13700_Display_Cursor
- S1D13700_Write_Char
- S1D13700_Write_Text
- S1D13700_Dot
- S1D13700_Line
- S1D13700_H_Line
- S1D13700_V_Line
- S1D13700_Rectangle
- S1D13700_Box
- S1D13700_Rectangle_Round_Edges
- S1D13700_Rectangle_Round_Edges_Fill
- S1D13700_Circle
- S1D13700_Circle_Fill
- S1D13700_Image
- S1D13700_PartialImage

S1D13700_Init

Prototype	`sub procedure S1D13700_Init(dim width as word, dim height as word)`
Returns	Nothing.
Description	Initializes S1D13700 Graphic Lcd controller. Parameters: - `width`: width of the Glcd panel. - `height`: height of the Glcd panel.
Requires	Global variables: - `S1D13700_Data_Port`: Data Bus Port. - `S1D13700_WR`: Write signal pin. - `S1D13700_RD`: Read signal pin. - `S1D13700_A0`: Command/Data signal pin. - `S1D13700_RES`: Reset signal pin. - `S1D13700_CS`: Chip Select signal pin. - `S1D13700_Data_Port_Direction`: Data Bus Port Direction. - `S1D13700_WR_Direction`: Direction of Write signal pin. - `S1D13700_RD_Direction`: Direction of Read signal pin. - `S1D13700_A0_Direction`: Direction of Command/Data signal pin. - `S1D13700_RES_Direction`: Direction of Reset signal pin. - `S1D13700_CS_Direction`: Direction of Chip Select signal pin. must be defined before using this function.

Example

```
' S1D13700 module connections
dim S1D13700_Data_Port as byte at PORTD
dim S1D13700_WR as sbit at LATC2_bit
dim S1D13700_RD as sbit at LATC1_bit
dim S1D13700_A0 as sbit at LATC0_bit
dim S1D13700_RES as sbit at LATC4_bit
dim S1D13700_CS as sbit at LATC5_bit

dim S1D13700_Data_Port_Direction as byte at TRISD
dim S1D13700_WR_Direction as sbit at TRISC2_bit
dim S1D13700_RD_Direction as sbit at TRISC1_bit
dim S1D13700_A0_Direction as sbit at TRISC0_bit
dim S1D13700_RES_Direction as sbit at TRISC4_bit
dim S1D13700_CS_Direction as sbit at TRISC5_bit
' End of S1D13700 module connections
...
' init display for 320 pixel width, 240 pixel height
S1D13700_Init(320, 240)
```

S1D13700_Write_Command

Prototype	`sub procedure S1D13700_Write_Command(dim command as byte)`
Returns	Nothing.
Description	Writes a command to S1D13700 controller. Parameters: - `command`: command to be issued:
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`' Turn the display on` `S1D13700_Write_Command(S1D13700_DISP_ON)`

Value	Description
`S1D13700_SYSTEM_SET`	General system settings.
`S1D13700_POWER_SAVE`	Enter into power saving mode.
`S1D13700_DISP_ON`	Turn the display on.
`S1D13700_DISP_OFF`	Turn the display off.
`S1D13700_SCROLL`	Setup text and graphics address regions.
`S1D13700_CS_RIGHT`	Cursor moves right after write to display memory.
`S1D13700_CS_LEFT`	Cursor moves left after write to display memory.
`S1D13700_CS_UP`	Cursor moves up after write to display memory.
`S1D13700_CS_DOWN`	Cursor moves down after write to display memory.
`S1D13700_OVLAY`	Configure how layers overlay.
`S1D13700_CGRAM_ADR`	Configure character generator RAM address.
`S1D13700_HDOT_SCR`	Set horizontal scroll rate.
`S1D13700_CSRW`	Set the cursor address.
`S1D13700_CSRR`	Read the cursor address.
`S1D13700_GRAYSCALE`	Selects the gray scale depth, in bits-per-pixel (bpp).
`S1D13700_MEMWRITE`	Write to display memory.
`S1D13700_MEMREAD`	Read from display memory.

S1D13700_Write_Parameter

Prototype	`sub procedure S1D13700_Write_Parameter(dim parameter as byte)`
Returns	Nothing.
Description	Writes a parameter to S1D13700 controller. Parameters: - `parameter`: parameter to be written.
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine. Previously, a command must be sent through S1D13700_Write_Command routine.

Example	`S1D13700_Write_Command(S1D13700_CSRW) ' set cursor address`
	`S1D13700_Write_Parameter(Lo(start)) ' send lower byte of cursor address`
	`S1D13700_Write_Parameter(Hi(start)) ' send higher byte cursor address`

S1D13700_Read_Parameter

Prototype	`sub function S1D13700_Read_Parameter() as byte`
Returns	Nothing.
Description	Reads a parameter from GLCD port.
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`parameter = S1D13700_Read_Parameter()`

S1D13700_Fill

Prototype	`sub procedure S1D13700_Fill(dim d as byte, dim start as word, dim len as word)`
Returns	Nothing.
Description	Fills Glcd memory block with given byte. Parameters: - `d`: byte to be written. - `start`: starting address of the memory block. - `len`: length of the memory block in bytes.
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`' from the starting address of 0x3000, fill the memory block size of 0x7FFF with 0x20` `S1D13700_Fill(0x20, 0x3000, 0x7FFF)`

S1D13700_GrFill

Prototype	`sub procedure S1D13700_GrFill(dim d as byte)`
Returns	Nothing.
Description	Fill graphic layer with appropriate value (0 to clear). Parameters: - `d`: value to fill graphic layer with.
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`' clear current graphic panel` `S1D13700_GrFill(0)`

S1D13700_TxtFill

Prototype	`sub procedure S1D13700_TxtFill(dim d as byte)`
Returns	Nothing.
Description	Fill current text panel with appropriate value (0 to clear). Parameters: - `d`: this value will be used to fill text panel.
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`' clear current text panel` `S1D13700_TxtFill(0)`

S1D13700_Display_GrLayer

Prototype	`<sub procedure S1D13700_Display_GrLayer(dim mode as byte)`
Returns	Nothing.
Description	Display selected graphic layer. Parameters: - `mode`: graphic layer mode. Valid values:
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`' Turn on graphic layer` `S1D13700_Display_GrLayer(S1D13700_LAYER_ON)`

Value	Description
`S1D13700_LAYER_OFF`	Turn off graphic layer.
`S1D13700_LAYER_ON`	Turn on graphic layer.
`S1D13700_LAYER_FLASH_2Hz`	Turn on graphic layer and flash it at the rate of 2 Hz.
`S1D13700_LAYER_FLASH_16Hz`	Turn on graphic layer and flash it at the rate of 16 Hz.

S1D13700_Display_TxtLayer

Prototype	`sub procedure S1D13700_Display_TxtLayer(dim mode as byte)`
Returns	Nothing.
Description	Display selected text layer. Parameters: - `mode`: text layer mode. Valid values: `S1D13700_LAYER_OFF` – Turn off graphic layer. `S1D13700_LAYER_ON` – Turn on graphic layer. `S1D13700_LAYER_FLASH_2Hz` – Turn on graphic layer and flash it at the rate of 2 Hz. `S1D13700_LAYER_FLASH_16Hz` – Turn on graphic layer and flash it at the rate of 16 Hz.
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	

```
' Display on text layer
S1D13700_Display_TxtLayer(S1D13700_LAYER_ON)
```

Valid values for `mode`:

Value	Description
`S1D13700_LAYER_OFF`	Turn off graphic layer.
`S1D13700_LAYER_ON`	Turn on graphic layer.
`S1D13700_LAYER_FLASH_2Hz`	Turn on graphic layer and flash it at the rate of 2 Hz.
`S1D13700_LAYER_FLASH_16Hz`	Turn on graphic layer and flash it at the rate of 16 Hz.

S1D13700_Set_Cursor

Prototype	`sub procedure S1D13700_Set_Cursor(dim width as byte, dim height as byte, dim mode as byte)`
Returns	Nothing.
Description	Sets cursor properties. Parameters: - `width`: in pixels-1 (must be less than or equal to the horizontal char size). - `height`: in lines-1 (must be less than or equal to the vertical char size). - `mode`: cursor mode. Valid values:
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	

```
' set cursor with the following properties : width 5px, height 10px, cursor shape - block
S1D13700_Set_Cursor(5, 10, S1D13700_CURSOR_BLOCK)
```

Valid values for `mode`:

Value	Description
`S1D13700_CURSOR_UNDERSCORE`	Set cursor shape - underscore.
`S1D13700_CURSOR_BLOCK`	Set cursor shape - block.

S1D13700_Display_Cursor

<table>
<tr><td>Prototype</td><td><code>sub procedure S1D13700_Display_Cursor(dim mode as byte)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Displays cursor.
Parameters:
- <code>mode</code>: mode parameter. Valid values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>S1D13700_CURSOR_OFF</code></td><td>Turn off graphic layer.</td></tr>
<tr><td><code>S1D13700_CURSOR_ON</code></td><td>Turn on graphic layer.</td></tr>
<tr><td><code>S1D13700_CURSOR_FLASH_2Hz</code></td><td>Turn on graphic layer and flash it at the rate of 2 Hz.</td></tr>
<tr><td><code>S1D13700_CURSOR_FLASH_16Hz</code></td><td>Turn on graphic layer and flash it at the rate of 16 Hz.</td></tr>
</table>
</td></tr>
<tr><td>Requires</td><td>Glcd module needs to be initialized. See the S1D13700_Init routine.</td></tr>
<tr><td>Example</td><td><code>' set cursor on</code>
<code>S1D13700_Display_Cursor(S1D13700_CURSOR_ON)</code></td></tr>
</table>

S1D13700_Write_Char

<table>
<tr><td>Prototype</td><td><code>sub procedure S1D13700_Write_Char(dim c as char, dim x as word, dim y as word, dim mode as byte)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Writes a char in the current text layer of Glcd at coordinates (x, y).
Parameters:
- <code>c</code>: char to be written.
- <code>x</code>: char position on x-axis (column).
- <code>y</code>: char position on y-axis (row).
- <code>mode</code>: mode parameter. Valid values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>S1D13700_OVERLAY_OR</code></td><td>In the OR-Mode, text and graphics can be displayed and the data is logically “OR-ed”.
This is the most common way of combining text and graphics, for example labels on buttons.</td></tr>
<tr><td><code>S1D13700_OVERLAY_XOR</code></td><td>In this mode, the text and graphics data are combined via the logical “exclusive OR”.</td></tr>
<tr><td><code>S1D13700_OVERLAY_AND</code></td><td>The text and graphic data shown on display are combined via the logical “AND function”.</td></tr>
</table>
</td></tr>
<tr><td>Requires</td><td>Glcd module needs to be initialized. See the S1D13700_Init routine.</td></tr>
<tr><td>Example</td><td><code>S1D13700_Write_Char(“A”,22,23,S1D13700_OVERLAY_OR)</code></td></tr>
</table>

S1D13700_Write_Text

<table>
<tr><td>Prototype</td><td><code>sub procedure S1D13700_Write_Text(dim byref str as string, dim x, y as word, dim mode as byte)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Writes text in the current text panel of Glcd at coordinates (x, y).

Parameters:

- <code>str</code>: text to be written.
- <code>x</code>: text position on x-axis (column).
- <code>y</code>: text position on y-axis (row).
- <code>mode</code>: mode parameter. Valid values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>S1D13700_OVERLAY_OR</code></td><td>In the OR-Mode, text and graphics can be displayed and the data is logically “OR-ed”.
This is the most common way of combining text and graphics, for example labels on buttons.</td></tr>
<tr><td><code>S1D13700_OVERLAY_XOR</code></td><td>In this mode, the text and graphics data are combined via the logical “exclusive OR”.</td></tr>
<tr><td><code>S1D13700_OVERLAY_AND</code></td><td>The text and graphic data shown on display are combined via the logical “AND function”.</td></tr>
</table></td></tr>
<tr><td>Requires</td><td>Glcd module needs to be initialized. See the S1D13700_Init routine.</td></tr>
<tr><td>Example</td><td><code>S1D13700_Write_Text('EPSON LIBRARY DEMO, WELCOME !', 0, 0, S1D13700_OVERLAY_OR)</code></td></tr>
</table>

S1D13700_Dot

<table>
<tr><td>Prototype</td><td><code>sub procedure S1D13700_Dot(dim x as word, dim y as word, dim color as byte)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Draws a dot in the current graphic panel of Glcd at coordinates (x, y).

Parameters:

- <code>x</code>: dot position on x-axis.
- <code>y</code>: dot position on y-axis.
- <code>color</code>: color parameter. Valid values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>S1D13700_BLACK</code></td><td>Black color.</td></tr>
<tr><td><code>S1D13700_WHITE</code></td><td>White color.</td></tr>
</table></td></tr>
<tr><td>Requires</td><td>Glcd module needs to be initialized. See the S1D13700_Init routine.</td></tr>
<tr><td>Example</td><td><code>S1D13700_Dot(50, 50, S1D13700_WHITE)</code></td></tr>
</table>

S1D13700_Line

Prototype	`sub procedure S1D13700_Line(dim x0, y0, x1, y1 as word, dim pcolor as byte)`
Returns	Nothing.
Description	Draws a line from (x0, y0) to (x1, y1). Parameters: - `x0`: x coordinate of the line start. - `y0`: y coordinate of the line end. - `x1`: x coordinate of the line start. - `y1`: y coordinate of the line end. - `pcolor`: color parameter. Valid values:
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`S1D13700_Line(0, 0, 239, 127, S1D13700_WHITE)`

Value	Description
`S1D13700_BLACK`	Black color.
`S1D13700_WHITE`	White color.

S1D13700_H_Line

Prototype	`sub procedure S1D13700_H_Line(dim x_start, x_end, y_pos as word, dim color as byte)`
Returns	Nothing.
Description	Draws a horizontal line. Parameters: - `x_start`: `x` coordinate of the line start. - `x_end`: `x` coordinate of the line end. - `y_pos`: line position on the y axis. - `pcolor`: color parameter. Valid values :
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`S1D13700_Line(0, 0, 239, 127, S1D13700_WHITE)`

Value	Description
`S1D13700_BLACK`	Black color.
`S1D13700_WHITE`	White color.

S1D13700_V_Line

Prototype	`sub procedure S1D13700_V_Line(dim y_start, y_end, x_pos as word, dim color as byte)`
Returns	Nothing.
Description	Draws a horizontal line. Parameters: - `y_start`: y coordinate of the line start. - `y_end`: y coordinate of the line end. - `x_pos`: line position on the x axis. - `pcolor`: color parameter. Valid values:
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`S1D13700_Line(0, 0, 239, 127, S1D13700_WHITE)`

Value	Description
`S1D13700_BLACK`	Black color.
`S1D13700_WHITE`	White color.

S1D13700_Rectangle

Prototype	`sub procedure S1D13700_Rectangle(dim x0, y0, x1, y1 as word, dim pcolor as byte)`
Returns	Nothing.
Description	Draws a rectangle on Glcd. Parameters: - `x0`: x coordinate of the upper left rectangle corner. - `y0`: y coordinate of the upper left rectangle corner. - `x1`: x coordinate of the lower right rectangle corner. - `y1`: y coordinate of the lower right rectangle corner. - `pcolor`: color parameter. Valid values:
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`S1D13700_rectangle(20, 20, 219, 107, S1D13700_WHITE)`

Value	Description
`S1D13700_BLACK`	Black color.
`S1D13700_WHITE`	White color.

S1D13700_Box

<table>
<tr><td>Prototype</td><td><code>sub procedure S1D13700_Box(dim x0, y0, x1, y1 as word, dim pcolor as byte)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Draws a rectangle on Glcd.

Parameters:

- <code>x0</code>: x coordinate of the upper left rectangle corner.
- <code>y0</code>: y coordinate of the upper left rectangle corner.
- <code>x1</code>: x coordinate of the lower right rectangle corner.
- <code>y1</code>: y coordinate of the lower right rectangle corner.
- <code>pcolor</code>: color parameter. Valid values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>S1D13700_BLACK</code></td><td>Black color.</td></tr>
<tr><td><code>S1D13700_WHITE</code></td><td>White color.</td></tr>
</table></td></tr>
<tr><td>Requires</td><td>Glcd module needs to be initialized. See the S1D13700_Init routine.</td></tr>
<tr><td>Example</td><td><code>S1D13700_Box(0, 119, 239, 127, S1D13700_WHITE)</code></td></tr>
</table>

S1D13700_Rectangle_Round_Edges

<table>
<tr><td>Prototype</td><td><code>sub procedure S1D13700_Rectangle_Round_Edges(dim x_upper_left as word, dim y_upper_left as word, dim x_bottom_right as word, dim y_bottom_right as word, dim round_radius as word, dim color as byte)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Draws a rounded edge rectangle on Glcd.

Parameters:

- <code>x_upper_left</code>: x coordinate of the upper left rectangle corner.
- <code>y_upper_left</code>: y coordinate of the upper left rectangle corner.
- <code>x_bottom_right</code>: x coordinate of the lower right rectangle corner.
- <code>y_bottom_right</code>: y coordinate of the lower right rectangle corner.
- <code>round_radius</code>: radius of the rounded edge.
- <code>pcolor</code>: color parameter. Valid values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>S1D13700_BLACK</code></td><td>Black color.</td></tr>
<tr><td><code>S1D13700_WHITE</code></td><td>White color.</td></tr>
</table></td></tr>
<tr><td>Requires</td><td>Glcd module needs to be initialized. See the S1D13700_Init routine.</td></tr>
<tr><td>Example</td><td><code>S1D13700_Rectangle_Round_Edges(20, 20, 219, 107, 12, S1D13700_WHITE)</code></td></tr>
</table>

S1D13700_Rectangle_Round_Edges_Fill

<table>
<tr><td>Prototype</td><td><code>sub procedure S1D13700_Rectangle_Round_Edges_Fill(dim x_upper_left as word, dim y_upper_left as word, dim x_bottom_right as word, dim y_bottom_right as word, dim round_radius as word, dim color as byte)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Draws a filled rounded edge rectangle on Glcd.

Parameters:

- <code>x_upper_left</code>: x coordinate of the upper left rectangle corner.
- <code>y_upper_left</code>: y coordinate of the upper left rectangle corner.
- <code>x_bottom_right</code>: x coordinate of the lower right rectangle corner.
- <code>y_bottom_right</code>: y coordinate of the lower right rectangle corner.
- <code>round_radius</code>: radius of the rounded edge.
- <code>pcolor</code>: color parameter. Valid values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>S1D13700_BLACK</code></td><td>Black color.</td></tr>
<tr><td><code>S1D13700_WHITE</code></td><td>White color.</td></tr>
</table>
</td></tr>
<tr><td>Requires</td><td>Glcd module needs to be initialized. See the S1D13700_Init routine.</td></tr>
<tr><td>Example</td><td><code>S1D13700_Rectangle_Round_Edges_Fill(20, 20, 219, 107, 12, S1D13700_WHITE)</code></td></tr>
</table>

S1D13700_Circle

<table>
<tr><td>Prototype</td><td><code>sub procedure S1D13700_Circle(dim x_center as word, dim y_center as word, dim radius as word, dim color as byte)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Draws a circle on Glcd.

Parameters:

- <code>x_center</code>: x coordinate of the circle center.
- <code>y_center</code>: y coordinate of the circle center.
- <code>radius</code>: radius size.
- <code>color</code>: color parameter. Valid values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>S1D13700_BLACK</code></td><td>Black color.</td></tr>
<tr><td><code>S1D13700_WHITE</code></td><td>White color.</td></tr>
</table>
</td></tr>
<tr><td>Requires</td><td>Glcd module needs to be initialized. See the S1D13700_Init routine.</td></tr>
<tr><td>Example</td><td><code>S1D13700_Circle(120, 64, 110, S1D13700_WHITE)</code></td></tr>
</table>

S1D13700_Circle_Fill

Prototype	`sub procedure S1D13700_Circle_Fill(dim x_center as word, dim y_center as word, dim radius as word, dim color as byte)`
Returns	Nothing.
Description	Draws a filled circle on Glcd. Parameters: - `x_center`: x coordinate of the circle center. - `y_center`: y coordinate of the circle center. - `radius`: radius size. - `color`: color parameter. Valid values:
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`S1D13700_Circle_Fill(120, 64, 110, S1D13700_WHITE)`

Value	Description
`S1D13700_BLACK`	Black color.
`S1D13700_WHITE`	White color.

S1D13700_Image

Prototype	`sub procedure S1D13700_Image(dim image as ^const byte)`
Returns	Nothing.
Description	Displays bitmap on Glcd. Parameters: - `image`: image to be displayed. Bitmap array is located in code memory. **Note**: Image dimension must match the display dimension.
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`S1D13700_Image(@image)`

S1D13700_PartialImage

Prototype	`sub procedure S1D13700_PartialImage(dim x_left, y_top, width, height, picture_width, picture_height as word, dim image as ^const byte)`
Returns	Nothing.
Description	Displays a partial area of the image on a desired location. Parameters: - `x_left`: x coordinate of the desired location (upper left coordinate). - `y_top`: y coordinate of the desired location (upper left coordinate). - `width`: desired image width. - `height`: desired image height. - `picture_width`: width of the original image. - `picture_height`: height of the original image. - `image`: image to be displayed. Bitmap array is located in code memory. **Note**: Image dimension must match the display dimension.
Requires	Glcd module needs to be initialized. See the S1D13700_Init routine.
Example	`' Draws a 10x15 part of the image starting from the upper left corner on the coordinate (10,12). Original image size is 16x32.` `S1D13700_PartialImage(10, 12, 10, 15, 16, 32, @image)`

Flash Memory Library

This library provides routines for accessing microcontroller's (internal) Flash memory.

On the dsPIC30/33 and PIC24, Flash memory is mapped to address space 3:2, which means that every 3 consecutive bytes of Flash have 2 consecutive address locations available. That is why mikroE's library allows data to be written to flash in two ways: "regular" and "compact". In the "regular" mode, which is used for word(16-bit) variables, the 3rd (unaddressable) flash memory byte remains unused. In the "compact" mode, which can be used for 1 byte-sized variables/ arrays, all flash bytes are being used.

All dsPIC30/33 and PIC24 MCUs use the RTSP module to perform Read/Erase/Write operations on Flash memory. This, together with the internal structure of the Flash, imposes certain rules to be followed when working with Flash memory:

dsPIC30:

- Erasing can be done only in 32-instructions (64 addresses, 96 bytes) memory blocks. This means that the block start address should be a multiply of 64 (i.e. have 6 lower bits set to zero).
- Data is read and written in 4-instructions (8 addresses, 12 bytes) blocks.This means that the block start address should be a multiply of 8 (i.e. have 3 lower bits set to zero).
- On the dsPIC30s, 2 address locations are assigned on every 3 bytes of (flash) program memory. Due to this specific and non-one-to-one address mapping, the mikroBasic PRO for dsPIC30/33 and PIC24 offers two sets of Flash handling functions: "regular" and "compact".
  Using the "regular" set, the user can write one byte of data to a single address, which means that each byte of written data has its own address, but on every 2 written bytes one byte of Flash memory remains empty.
  Using the "compact" set, every byte of Flash memory, including those non-addressable, is filled with data; this method can only be used for data organized in bytes.
  The "compact" functions have `_Compact` as name suffix.
- For run-time FLASH read/write, the dsPIC30's RTSP module is being used. It organizes data into rows and panels. Each row contains write latches that can hold 4 instructions (12 bytes). The number of panels varies from one dsPIC30 MCU model to another. Because of that, the flash write sequence has been split into several operations (`_Write_Init(), _Write_LoadLatch4(), _Write_DoWrite()`), in order to be usable on all dsPICs.

PIC24 and dsPIC33:

- Erasing can be done only in 512-instructions (1024 addresses, 1536 bytes) memory blocks, which means that the block start address should be a multiply of 1024 (i.e. have 10 lower bits set to zero).
- Data is read and written in 64-instructions (128 addresses, 192 bytes) blocks.This means that the block start address should be a multiply of 128 (i.e. have 7 lower bits set to zero).
- On the dsPIC33 and PIC24s, 2 address locations are assigned on every 3 bytes of (flash) program memory. Due to this specific and non-one-to-one address mapping, the mikroBasic PRO for dsPIC30/33 and PIC24 offers two sets of Flash handling functions: "regular" and "compact".
  Using the "regular" set, the user can write one byte of data to a single address, which means that each byte of written data has its own address, but on every 2 written bytes one byte of Flash memory remains empty.
  Using the "compact" set, every byte of Flash memory, including those non-addressable, is filled with data; this method can only be used for data organized in bytes.
  The "compact" functions have `_Compact` as name suffix.

24F04KA201 and 24F16KA102 Family Specifics:

- These MCU's have their Flash memory organized into memory blocks of 32 instructions (96 bytes), unlike other PIC24 devices.
- Erasing can be done only in 32-instructions (64 addresses, 96 bytes) memory blocks, which means that the block start address should be a multiply of 64 (i.e. have 6 lower bits set to zero).
- Data is read and written in 32-instructions (64 addresses, 96 bytes) blocks. This means that the block start address should be a multiply of 64 (i.e. have 6 lower bits set to zero).
- Unlike other PIC24 devices, writing or erasing one block of data (32 instructions), is followed by erasing the memory block of the same size (32 instructions).

Library Routines

dsPIC30 Functions

- FLASH_Erase32

- FLASH_Write_Block
- FLASH_Write_Compact
- FLASH_Write_Init
- FLASH_Write_Loadlatch4
- FLASH_Write_Loadlatch4_Compact
- FLASH_Write_DoWrite

- FLASH_Read4
- FLASH_Read4_Compact

PIC24 and dsPIC33 Functions

- FLASH_Erase
- FLASH_Write
- FLASH_Write_Compact
- FLASH_Read
- FLASH_Read_Compact

dsPIC30 Functions

FLASH_Erase32

Prototype	`sub procedure FLASH_Erase32(dim flash_address as longint)`
Description	Erases one block (32 instructions, 64 addresses, 96 bytes)from the program FLASH memory.
Parameters	- `address`: starting address of the FLASH memory block
Returns	Nothing.
Requires	Nothing.
Example	`' erase the 32-instruction block, starting from address 0x006000` `FLASH_Erase32(0x006000)`
Notes	The user should take care about the address alignment (see the explanation at the beginning of this page).

FLASH_Write_Block

Prototype	`sub procedure FLASH_Write_Block(dim flash_address as longint, dim data_address as word)`
Description	Fills one writeable block of Flash memory (4 instructions, 8 addresses, 12 bytes) in the “regular” mode. Addresses and data are being mapped 1-on-1. This also means that 3rd byte of each program location remains unused.
Parameters	- `flash_address`: starting address of the FLASH memory block - `data_address`: data to be written
Returns	Nothing.
Requires	The block to be written to must be erased first, either from the user code (through the RTSP), or during the programming of MCU. Please note that block size that is to be erased is different from the one that can be written with this function!

Example

```
dim flash_address as longint
    cArr          as string[4]
    ptr_data      as word
...
flash_address = 0x006000
cArr = "ABCD"
ptr_data = @cArr
FLASH_Write_Block(flash_address, ptr_data)
```

Notes	The user should take care about the address alignment (see the explanation at the beginning of this page).

FLASH_Write_Compact

Prototype	`sub procedure FLASH_Write_Compact(dim flash_address as longint, dim data_address as word, dim bytes as word)`
Description	Fills a portion of Flash memory using the dsPIC30 RTSP module, in the “compact” manner. In this way, several blocks of RTSP’s latch can be written in one pass. One latch block contains 4 instructions (8 addresses, 12 bytes). Up to 8 latch blocks can be written in one round, resulting in a total of 8*12 = 96 bytes. This method uses all available bytes of the program FLASH memory, including those that are not mapped to address space (every 3rd byte).
Parameters	- `flash_address`: starting address of the FLASH memory block - `data_address`: data to be written - `bytes`: number of bytes to be written. The amount of bytes to be written must be a multiply of 12, since this is the size of the RTSP’s write latch(es).
Returns	Nothing.
Requires	The block to be written to must be erased first, either from the user code FLASH_Erase32, or during the programming of MCU. Please note that block size that is to be erased is different from the one that can be written with this function!

Example

```
dim flash_address as longint
    cArr          as string[36]
    ptr_data      as word
...
flash_address = 0x006000
cArr = “mikroElektronika12mikroElektronika34”
ptr_data = @cArr
FLASH_Write_Compact(flash_address, ptr_data, 36)
```

Notes	The user should take care about the address alignment (see the explanation at the beginning of this page).

FLASH_Write_Init

Prototype	`sub procedure FLASH_Write_Init(dim flash_address as longint, dim data_address as word)`
Description	Initializes RTSP for write-to-FLASH operation.
Parameters	- `flash_address`: starting address of the FLASH memory block - `data_address`: data to be written
Returns	Nothing.
Requires	The block to be written to must be erased first, either from the user code FLASH_Erase32, or during the programming of MCU. Please note that block size that is to be erased is different from the one that can be written with this function!

Example

```
const iArr as word[8] = (“m”, “i”, “k”, “r”, “o”, “E”, “l”, “e”)
dim ptr_data as word
  ...
  ptr_data = @iArr
  FLASH_Write_Init(0x006100, ptr_data)
  FLASH_Write_Loadlatch4()
  FLASH_Write_Loadlatch4()
  FLASH_Write_DoWrite()
```

Notes	The user should take care about the address alignment (see the explanation at the beginning of this page).

FLASH_Write_Loadlatch4

Prototype	`sub procedure FLASH_Write_Loadlatch4()`
Description	Loads the current RTSP write latch with data (4 instructions, 8 addresses, 12 bytes). The data is filled in the “regular” mode.
Parameters	None.
Returns	Nothing.
Requires	The block to be written to must be erased first, either from the user code FLASH_Erase32, or during the programming of MCU. Please note that block size that is to be erased is different from the one that can be written with this function! This function is used as a part of the Flash write sequence, therefore the FLASH_Write_Init function must be called before this one. This function can be called several times before commiting the actual write-to-Flash operation FLASH_Write_DoWrite. This depends on the organization of the RTSP module for the certain dsPIC30. Please consult the Datasheet for particular dsPIC30 on this subject.

Example

```
const  iArr  as word[8] = (“m”, “i”, “k”, “r”, “o”, “E”, “l”, “e”)
dim ptr_data as word
  ...
  ptr_data = @iArr
  FLASH_Write_Init(0x006100, ptr_data)
  FLASH_Write_Loadlatch4()
  FLASH_Write_Loadlatch4()
  FLASH_Write_DoWrite()
```

Notes	None.

FLASH_Write_Loadlatch4_Compact

Prototype	`void FLASH_Write_Loadlatch4_Compact();`
Description	Loads the current RTSP write latch with data (4 instructions, 8 addresses, 12 bytes). The data is filled in the “compact” mode.
Parameters	None.
Returns	Nothing.
Requires	The block to be written to must be erased first, either from the user code FLASH_Erase32, or during the programming of MCU. Please note that block size that is to be erased is different from the one that can be written with this function! This function is used as a part of the Flash write sequence, therefore the FLASH_Write_Init function must be called before this one. This function can be called several times before committing actual write-to-Flash operation FLASH_Write_DoWrite. This depends on the organization of the RTSP module for the certain dsPIC30. Please consult the Datasheet for particular dsPIC30 on this subject.

Example

```
const  iArr  as word[8] = (“m”, “i”, “k”, “r”, “o”, “E”, “l”, “e”)
dim ptr_data as word
  ...
  ptr_data = @iArr
  FLASH_Write_Init(0x006100, ptr_data)
  FLASH_Write_Loadlatch4_Compact()
  FLASH_Write_Loadlatch4_Compact()
  FLASH_Write_DoWrite()
```

Notes	None.

FLASH_Write_DoWrite

Prototype	`sub procedure FLASH_Write_DoWrite()`
Description	Commits the FLASH write operation.
Parameters	None.
Returns	Nothing.
Requires	The block to be written to must be erased first, either from the user code FLASH_Erase32, or during the programming of MCU. Please note that block size that is to be erased is different from the one that can be written with this function! This function is used as a part of the Flash write sequence, therefore FLASH_Write_Init and certain number of FLASH_Write_Loadlatch4 or FLASH_Write_Loadlatch4_Compact function calls must be made before this one. This function is to be called once, at the and of the FLASH write sequence.

Example

```
const  iArr  as word[8] = (“m”, “i”, “k”, “r”, “o”, “E”, “l”, “e”)
dim ptr_data as word
   ...
  ptr_data = @iArr
  FLASH_Write_Init(0x006100, ptr_data)
  FLASH_Write_Loadlatch4()
  FLASH_Write_Loadlatch4()
  FLASH_Write_DoWrite()
```

Notes	None.

FLASH_Read4

Prototype	`sub procedure FLASH_Read4(dim flash_address as longint, dim write_to as word)`
Description	Reads one latch row (4 instructions, 8 addresses) in the “regular” mode.
Parameters	- `address`: starting address of the FLASH memory block to be read - `write_to`: starting address of RAM buffer for storing read data
Returns	Starting address of RAM buffer for storing read data.
Requires	Nothing.

Example

```
dim flash_address as longint
    cArr          as word[4]
    ptr_data      as word
...
flash_address = 0x006000
ptr_data = @cArr
FLASH_Read4(flash_address, ptr_data)
```

Notes	The user should take care of the address alignment (see the explanation at the beginning of this page).

FLASH_Read4_Compact

Prototype	`sub procedure FLASH_Read4_Compact(dim flash_address as longint, dim write_to as word)`
Description	Reads one latch row (4 instructions, 8 addresses) in the “compact” mode.
Parameters	- `address`: starting address of the FLASH memory block to be read - `write_to`: starting address of RAM buffer for storing read data
Returns	Starting address of RAM buffer for storing read data.
Requires	Nothing.

Example

```
dim flash_address as longint
    cArr          as word[8]
    ptr_data      as word
...
flash_address = 0x006000
ptr_data = @cArr
FLASH_Read4_Compact(flash_address, ptr_data)
```

Notes	The user should take care of the address alignment (see the explanation at the beginning of this page).

PIC24 and dsPIC33 Functions

FLASH_Erase

Prototype	`sub procedure FLASH_Erase(dim address as longint)`
Description	Erases one block (512 instructions, 1024 addresses, 1536 bytes) from the program FLASH memory.
Parameters	- `address`: starting address of the FLASH memory block
Returns	Nothing.
Requires	Nothing.

Example

```
'--- erase the flash memory block, starting from address 0x006400
dim flash_address as longint
...
flash_address = 0x006400
FLASH_Erase(flash_address)
```

Notes	The user should take care about the address alignment (see the explanation at the beginning of this page).

FLASH_Write

Prototype	`sub procedure FLASH_Write(dim address as longint, dim byref data_ as word[64])`
Description	Fills one writeable block of Flash memory (64 instructions, 128 addresses, 192 bytes) in the “regular” mode. Addresses and data are being mapped 1-on-1. This also means that 3rd byte of each program location remains unused.
Parameters	- `address`: starting address of the FLASH memory block - `data_`: data to be written
Returns	Nothing.
Requires	The block to be written to must be erased first, either from the user code (through the RTSP), or during the programming of MCU. Please note that block size that is to be erased is different from the one that can be written with this function!

Example	`dim data_ as word[64] = {“m”, “i”, “k”, “r”, “o”, “E”, “l”, “e”, “k”, “t”, “r”, “o”, “n”, “i”, “k”, “a”}`
	`...`
	
	`FLASH_Write(0x006500, data_)`

Notes	The user should take care about the address alignment (see the explanation at the beginning of this page).

FLASH_Write_Compact

Prototype	`sub procedure FLASH_Write_Compact(dim address as longint, dim byref data_ as byte[192])`
Description	Fills a portion of Flash memory (64 instructions, 128 addresses, 192 bytes) using the dsPIC33 and PIC24s RTSP (Run Time Self Programming) module, in the “compact” manner. This method uses all available bytes of the program FLASH memory, including those that are not mapped to address space (every 3rd byte).
Parameters	- `address`: starting address of the FLASH memory block - `data_`: data to be written
Returns	Nothing.
Requires	The block to be written to must be erased first, either from the user code (FLASH_Erase), or during the programming of MCU. Please note that block size that is to be erased is different from the one that can be written with this function!

Example	`dim data_ as string[192]`
	`...`
	`data_ = “supercalifragillisticexpialidotiousABCDEFGHIJKLMNOPRSTUVWXYZ1234”`
	
	`FLASH_Write_Compact(0x006400, data_)`

Notes	The user should take care of the address alignment (see the explanation at the beginning of this page).

FLASH_Read

Prototype	`sub procedure FLASH_Read(dim address as longint, dim byref write_to as word[100], dim NoWords as word)`
Description	Reads required number of words from the flash memory in the “regular” mode.
Parameters	- `address`: starting address of the FLASH memory block to be read - `write_to`: starting address of RAM buffer for storing read data - `NoWords`: number of words to be read
Returns	Address of RAM buffer for storing read data.
Requires	

Example

```
dim Buffer as word[10]
    start_address as longint
...
FLASH_Write(0x006500, data)
start_address = 0x6500
FLASH_Read(start_address, Buffer, 10)
```

Notes	The user should take care of the address alignment (see the explanation at the beginning of this page).

FLASH_Read_Compact

Prototype	`sub procedure FLASH_Read_Compact(dim address as longint, dim byref write_to as byte[100], dim NoBytes as word)`
Description	Reads required number of bytes from the flash memory in the “compact” mode.
Parameters	- `address`: starting address of the FLASH memory block to be read - `write_to`: starting address of RAM buffer for storing read data - `NoBytes`: number of bytes to be read
Returns	Address of RAM buffer for storing read data.
Requires	

Example

```
dim Buffer as byte[10]
    start_address as longint
...
FLASH_Write(0x006500, data)
start_address = 0x6500
FLASH_Read(start_address, Buffer, 10)
```

Notes	The user should take care of the address alignment (see the explanation at the beginning of this page).

Library Example

In this example written for dsPIC30F4013, various read/write tecniques to/from the on-chip FLASH memory are shown. Flash memory is mapped to address space 3:2, meaning every 3 consecutive bytes of Flash have 2 consecutive address locations available.
That is why mikroE’s library allows data to be written to Flash in two ways: ‘regular’ and ‘compact’. In ‘regular’ mode, which is used for variables that are size of 2 bytes and more, the 3rd (un-addressable) byte remains unused.
In ‘compact’ mode, which can be used for 1 byte-sized variables/arrays, all bytes of flash are being used.

Copy Code To Clipboard

```
program Flash_Test
dim WriteWordArr  as word[8]
    WriteByteArr  as byte[32]
    ReadByteArr as byte[40]
    RealongwordArr as word[20]

    pw as ^word
    pb as ^byte
    i as word
    temp_byte as byte
main:
   ' Initialize arrays
  WriteWordArr[0] = "*"  WriteWordArr[1] = "m"  WriteWordArr[2] = "i"  WriteWordArr[3] = "k"
  WriteWordArr[4] = "r"  WriteWordArr[5] = "o"  WriteWordArr[6] = "E"  WriteWordArr[7] = "*"

  WriteByteArr[0]  = "m"  WriteByteArr[1]  = "i"  WriteByteArr[2]  = "k"  WriteByteArr[3] = "r"
  WriteByteArr[4]  = "o"  WriteByteArr[5]  = "E"  WriteByteArr[6]  = "l"  WriteByteArr[7] = "e"
  WriteByteArr[8]  = "k"  WriteByteArr[9]  = "t"  WriteByteArr[10] = "r"  WriteByteArr[11] = "o"
  WriteByteArr[12] = "n"  WriteByteArr[13] = "i"  WriteByteArr[14] = "k"  WriteByteArr[15] = "a"
  WriteByteArr[16] = " "  WriteByteArr[17] = "F"  WriteByteArr[18] = "l"  WriteByteArr[19] = "a"
  WriteByteArr[20] = "s"  WriteByteArr[21] = "h"  WriteByteArr[22] = " "  WriteByteArr[23] = "e"
  WriteByteArr[24] = "x"  WriteByteArr[25] = "a"  WriteByteArr[26] = "m"  WriteByteArr[27] = "p"
  WriteByteArr[28] = "l"  WriteByteArr[29] = "e"  WriteByteArr[30] = "."  WriteByteArr[31] = 0


  pb = @WriteByteArr
   '--- erase the block first
  FLASH_Erase32(0x006000)

  pb = @WriteByteArr[0]
  FLASH_Write_Compact(0x006000, pb, 36)
  (*
  This is what FLASH_Write_Compact() does "beneath the hood"
  *
  FLASH_Write_Init(0x006000, pv1)
  FLASH_Write_Loadlatch4_Compact()
  FLASH_Write_Loadlatch4_Compact()
  FLASH_Write_Loadlatch4_Compact()
  FLASH_Write_DoWrite()
  *)

 '--- read compact format
  pb = @ReadByteArr
  FLASH_Read4_Compact(0x006000, pb)
  pb = pb + 12
  FLASH_Read4_Compact(0x006008, pb)
  pb = pb + 12
  FLASH_Read4_Compact(0x006010, pb)
  pb = pb + 12
  pb^ = 0    'termination

  UART1_Init(9600)
  UART1_Write(10)
  UART1_Write(13)
```

```
  UART1_Write_Text(“Start”)
  UART1_Write(10)
  UART1_Write(13)
  i = 0
  while(ReadByteArr[i])
    temp_byte = ReadByteArr[i]
    UART1_Write(temp_byte)
    Inc(i)
  wend

  '--- now for some non-compact flash-write
  pw = @WriteWordArr
  '--- erase the block first
  FLASH_Erase32(0x006100)
  FLASH_Write_Init(0x006100, pw)
  FLASH_Write_Loadlatch4()
  FLASH_Write_Loadlatch4()
  FLASH_Write_DoWrite()

  '--- read non-compact format
  pw = @RealongwordArr[0]
  FLASH_Read4(0x006100, pw)
  pw = pw + 4
  FLASH_Read4(0x006108, pw)
  pw = pw + 4
  pw^ = 0    'termination

  '--- show what has been written
  UART1_Write(10)
  UART1_Write(13)
  i = 0
  while(RealongwordArr[i]<>0)
    temp_byte = RealongwordArr[i]
    UART1_Write(temp_byte)
    i = i + 1
  wend

end.
```

Graphic Lcd Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for operating Graphic Lcd 128x64 (with commonly used Samsung KS108/KS107 controller).

For creating a custom set of Glcd images use Glcd Bitmap Editor Tool.

Library Dependency Tree

External dependencies of Graphic Lcd Library

The following variables must be defined in all projects using Graphic Lcd Library:	Description:	Example:
`dim GLCD_D0 as sbit sfr external`	Data 0 line.	`dim GLCD_D0 as sbit at RB0_bit`
`dim GLCD_D1 as sbit sfr external`	Data 1 line.	`dim GLCD_D1 as sbit at RB1_bit`
`dim GLCD_D2 as sbit sfr external`	Data 2 line.	`dim GLCD_D2 as sbit at RB2_bit`
`dim GLCD_D3 as sbit sfr external`	Data 3 line.	`dim GLCD_D3 as sbit at RB3_bit`
`dim GLCD_D4 as sbit sfr external`	Data 4 line.	`dim GLCD_D4 as sbit at RD0_bit`
`dim GLCD_D5 as sbit sfr external`	Data 5 line.	`dim GLCD_D5 as sbit at RD1_bit`
`dim GLCD_D6 as sbit sfr external`	Data 6 line.	`dim GLCD_D6 as sbit at RD2_bit`
`dim GLCD_D7 as sbit sfr external`	Data 7 line.	`dim GLCD_D7 as sbit at RD3_bit`
`dim GLCD_CS1 as sbit sfr external`	Chip Select 1 line.	`dim GLCD_CS1 as sbit at LATB4_bit`
`dim GLCD_CS2 as sbit sfr external`	Chip Select 2 line.	`dim GLCD_CS2 as sbit at LATB5_bit`
`dim GLCD_RS as sbit sfr external`	Register select line.	`dim GLCD_RS as sbit at LATF0_bit`
`dim GLCD_RW as sbit sfr external`	Read/Write line.	`dim GLCD_RW as sbit at LATF1_bit`
`dim GLCD_EN as sbit sfr external`	Enable line.	`dim GLCD_EN as sbit at LATF4_bit`
`dim GLCD_RST as sbit sfr external`	Reset line.	`dim GLCD_RST as sbit at RB5_bit`
`dim GLCD_D0_Direction as sbit sfr external`	Direction of the Data 0 pin.	`dim GLCD_D0_Direction as sbit at TRISB0_bit`
`dim GLCD_D1_Direction as sbit sfr external`	Direction of the Data 1 pin.	`dim GLCD_D1_Direction as sbit at TRISB1_bit`
`dim GLCD_D2_Direction as sbit sfr external`	Direction of the Data 2 pin.	`dim GLCD_D2_Direction as sbit at TRISF2_bit`
`dim GLCD_D3_Direction as sbit sfr external`	Direction of the Data 3 pin.	`dim GLCD_D3_Direction as sbit at TRISF3_bit`
`dim GLCD_D4_Direction as sbit sfr external`	Direction of the Data 4 pin.	`dim GLCD_D4_Direction as sbit at TRISD0_bit`
`dim GLCD_D5_Direction as sbit sfr external`	Direction of the Data 5 pin.	`dim GLCD_D5_Direction as sbit at TRISD1_bit`
`dim GLCD_D6_Direction as sbit sfr external`	Direction of the Data 6 pin.	`dim GLCD_D6_Direction as sbit at TRISD2_bit`
`dim GLCD_D7_Direction as sbit sfr external`	Direction of the Data 7 pin.	`dim GLCD_D7_Direction as sbit at TRISD3_bit`
`dim GLCD_CS1_Direction as sbit sfr external`	Direction of the Chip Select 1 pin.	`dim GLCD_CS1_Direction as sbit at TRISB4_bit`
`dim GLCD_CS2_Direction as sbit sfr external`	Direction of the Chip Select 2 pin.	`dim GLCD_CS2_Direction as sbit at TRISB5_bit`
`dim GLCD_RS_Direction as sbit sfr external`	Direction of the Register select pin.	`dim GLCD_RS_Direction as sbit at TRISF0_bit`
`dim GLCD_RW_Direction as sbit sfr external`	Direction of the Read/Write pin.	`dim GLCD_RW_Direction as sbit at TRISF1_bit`
`dim GLCD_EN_Direction as sbit sfr external`	Direction of the Enable pin.	`dim GLCD_EN_Direction as sbit at TRISF4_bit`
`dim GLCD_RST_Direction as sbit sfr external`	Direction of the Reset pin.	`dim GLCD_RST_Direction as sbit at TRISF5_bit`

Library Routines

Basic routines:

- Glcd_Init
- Glcd_Set_Side
- Glcd_Set_X
- Glcd_Set_Page
- Glcd_Read_Data
- Glcd_Write_Data

Advanced routines:

- Glcd_Fill
- Glcd_Dot
- Glcd_Line
- Glcd_V_Line
- Glcd_H_Line
- Glcd_Rectangle
- Glcd_Rectangle_Round_Edges
- Glcd_Rectangle_Round_Edges_Fill
- Glcd_Box
- Glcd_Circle
- Glcd_Circle_Fill
- Glcd_Set_Font
- Glcd_Write_Char
- Glcd_Write_Text
- Glcd_Image
- Glcd_PartialImage

Glcd_Init

Prototype	`sub procedure Glcd_Init()`
Description	Initializes the Glcd module. Each of the control lines are both port and pin configurable, while data lines must be on a single port (pins <0:7>).
Parameters	None.
Returns	Nothing.
Requires	Global variables: - `GLCD_D0` : Data pin 0 - `GLCD_D1` : Data pin 1 - `GLCD_D2` : Data pin 2 - `GLCD_D3` : Data pin 3 - `GLCD_D4` : Data pin 4 - `GLCD_D5` : Data pin 5 - `GLCD_D6` : Data pin 6 - `GLCD_D7` : Data pin 7 - `GLCD_CS1` : Chip select 1 signal pin - `GLCD_CS2` : Chip select 2 signal pin - `GLCD_RS` : Register select signal pin - `GLCD_RW` : Read/Write Signal pin

Requires	- `GLCD_EN` : Enable signal pin - `GLCD_RST` : Reset signal pin - `GLCD_D0_Direction` : Direction of the Data pin 0 - `GLCD_D1_Direction` : Direction of the Data pin 1 - `GLCD_D2_Direction` : Direction of the Data pin 2 - `GLCD_D3_Direction` : Direction of the Data pin 3 - `GLCD_D4_Direction` : Direction of the Data pin 4 - `GLCD_D5_Direction` : Direction of the Data pin 5 - `GLCD_D6_Direction` : Direction of the Data pin 6 - `GLCD_D7_Direction` : Direction of the Data pin 7 - `GLCD_CS1_Direction` : Direction of the Chip select 1 pin - `GLCD_CS2_Direction` : Direction of the Chip select 2 pin - `GLCD_RS_Direction` : Direction of the Register select signal pin - `GLCD_RW_Direction` : Direction of the Read/Write signal pin - `GLCD_EN_Direction` : Direction of the Enable signal pin - `GLCD_RST_Direction` : Direction of the Reset signal pin must be defined before using this function.

Example

```
' Glcd module connections
dim GLCD_D7 as sbit at RD3_bit
    GLCD_D6 as sbit at RD2_bit
    GLCD_D5 as sbit at RD1_bit
    GLCD_D4 as sbit at RD0_bit
    GLCD_D3 as sbit at RB3_bit
    GLCD_D2 as sbit at RB2_bit
    GLCD_D1 as sbit at RB1_bit
    GLCD_D0 as sbit at RB0_bit
    GLCD_D7_Direction as sbit at TRISD3_bit
    GLCD_D6_Direction as sbit at TRISD2_bit
    GLCD_D5_Direction as sbit at TRISD1_bit
    GLCD_D4_Direction as sbit at TRISD0_bit
    GLCD_D3_Direction as sbit at TRISB3_bit
    GLCD_D2_Direction as sbit at TRISB2_bit
    GLCD_D1_Direction as sbit at TRISB1_bit
    GLCD_D0_Direction as sbit at TRISB0_bit

dim GLCD_CS1 as sbit at LATB4_bit
    GLCD_CS2 as sbit at LATB5_bit
    GLCD_RS  as sbit at LATF0_bit
    GLCD_RW  as sbit at LATF1_bit
    GLCD_EN  as sbit at LATF4_bit
    GLCD_RST as sbit at LATF5_bit

dim GLCD_CS1_Direction as sbit at TRISB4_bit
    GLCD_CS2_Direction as sbit at TRISB5_bit
    GLCD_RS_Direction  as sbit at TRISF0_bit
    GLCD_RW_Direction  as sbit at TRISF1_bit
    GLCD_EN_Direction  as sbit at TRISF4_bit
    GLCD_RST_Direction as sbit at TRISF5_bit
' End Glcd module connections

...

Glcd_Init()
```

Notes	None.

Glcd_Set_Side

Prototype	`sub procedure Glcd_Set_Side(dim x_pos as byte)`
Description	Selects Glcd side. Refer to the Glcd datasheet for detailed explanation.
Parameters	- `x_pos`: Specifies position on x-axis of the Glcd. Valid values: 0..127. Values from 0 to 63 specify the left side, values from 64 to 127 specify the right side of the Glcd.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.

Example	The following two lines are equivalent, and both of them select the left side of Glcd:
	`Glcd_Select_Side(0)`
	`Glcd_Select_Side(10)`

Notes	For side, x axis and page layout explanation see schematic at the bottom of this page.

Glcd_Set_X

Prototype	`sub procedure Glcd_Set_X(dim x_pos as byte)`
Description	Sets x-axis position to `x_pos` dots from the left border of Glcd within the selected side.
Parameters	- `x_pos`: position on x-axis. Valid values: 0..63
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`Glcd_Set_X(25)`
Notes	For side, x axis and page layout explanation see schematic at the bottom of this page.

Glcd_Set_Page

Prototype	`sub procedure Glcd_Set_Page(dim page as byte)`
Description	Selects page of the Glcd.
Parameters	- `page`: page number. Valid values: 0..7
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`Glcd_Set_Page(5)`
Notes	For side, x axis and page layout explanation see schematic at the bottom of this page.

Glcd_Read_Data

Prototype	`sub function Glcd_Read_Data() as byte`
Description	Reads data from from the current location of Glcd memory and moves to the next location.
Parameters	None.
Returns	One byte from Glcd memory, formatted as a word (16-bit).
Requires	Glcd needs to be initialized, see Glcd_Init routine. Glcd side, x-axis position and page should be set first. See functions Glcd_Set_Side, Glcd_Set_X, and Glcd_Set_Page.

Example	`dim data_ as byte`
	`...`
	`Glcd_Read_Data()`
	`data_ = Glcd_Read_Data()`

Notes	This routine needs to be called twice; After the first call, data is placed in the buffer register. After the second call, data is passed from the buffer register to data lines.

Glcd_Write_Data

Prototype	`sub procedure Glcd_Write_Data(dim data_ as byte)`
Returns	Nothing.
Description	Writes one byte to the current location in Glcd memory and moves to the next location. Parameters: - `data_`: data to be written
Requires	Glcd needs to be initialized, see Glcd_Init routine. Glcd side, x-axis position and page should be set first. See functions Glcd_Set_Side, Glcd_Set_X, and Glcd_Set_Page.

Example	`dim data_ as byte`
	`...`
	`Glcd_Write_Data(data_)`

Glcd_Fill

Prototype	`sub procedure Glcd_Fill(dim pattern as byte)`
Description	Fills Glcd memory with the byte pattern. To clear the Glcd screen, use `Glcd_Fill(0)`. To fill the screen completely, use `Glcd_Fill(0xFF)`.
Parameters	- `pattern`: byte to fill Glcd memory with.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`Glcd_Fill(0)   ' Clear screen`
Notes	None.

Glcd_Dot

Prototype	`sub procedure Glcd_Dot(dim x_pos, y_pos, color as byte)`
Description	Draws a dot on Glcd at coordinates (`x_pos, y_pos`).
Parameters	- `x_pos`: x position. Valid values: 0..127 - `y_pos`: y position. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines a dot state: 0 clears dot, 1 puts a dot, and 2 inverts dot state.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Invert the dot in the upper left corner` `Glcd_Dot(0, 0, 2)`
Notes	For x and y axis layout explanation see schematic at the bottom of this page.

Glcd_Line

Prototype	`sub procedure Glcd_Line(dim x_start, y_start, x_end, y_end as integer, dim color as byte)`
Description	Draws a line on Glcd.
Parameters	- `x_start`: x coordinate of the line start. Valid values: 0..127 - `y_start`: y coordinate of the line start. Valid values: 0..63 - `x_end`: x coordinate of the line end. Valid values: 0..127 - `y_end`: y coordinate of the line end. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the line color: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw a line between dots (0,0) and (20,30)` `Glcd_Line(0, 0, 20, 30, 1)`
Notes	None.

Glcd_V_Line

Prototype	`sub procedure Glcd_V_Line(dim y_start, y_end, x_pos, color as byte)`
Description	Draws a vertical line on Glcd.
Parameters	- `y_start`: y coordinate of the line start. Valid values: 0..63 - `y_end`: y coordinate of the line end. Valid values: 0..63 - `x_pos`: x coordinate of vertical line. Valid values: 0..127 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the line color: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw a vertical line between dots (10,5) and (10,25)` `Glcd_V_Line(5, 25, 10, 1)`
Notes	None.

Glcd_H_Line

Prototype	`sub procedure Glcd_H_Line(dim x_start, x_end, y_pos, color as byte)`
Description	Draws a horizontal line on Glcd.
Parameters	- `x_start`: x coordinate of the line start. Valid values: 0..127 - `x_end`: x coordinate of the line end. Valid values: 0..127 - `y_pos`: y coordinate of horizontal line. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the line color: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw a horizontal line between dots (10,20) and (50,20)` `Glcd_H_Line(10, 50, 20, 1)`
Notes	None.

Glcd_Rectangle

Prototype	`sub procedure Glcd_Rectangle(dim x_upper_left, y_upper_left, x_bottom_right, y_bottom_right, color as byte)`
Description	Draws a rectangle on Glcd.
Parameters	- `x_upper_left`: x coordinate of the upper left rectangle corner. Valid values: 0..127 - `y_upper_left`: y coordinate of the upper left rectangle corner. Valid values: 0..63 - `x_bottom_right`: x coordinate of the lower right rectangle corner. Valid values: 0..127 - `y_bottom_right`: y coordinate of the lower right rectangle corner. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the rectangle border: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw a rectangle between dots (5,5) and (40,40)` `Glcd_Rectangle(5, 5, 40, 40, 1)`
Notes	None.

Glcd_Rectangle_Round_Edges

Prototype	`sub procedure Glcd_Rectangle_Round_Edges(dim x_upper_left as byte, dim y_upper_left as byte, dim x_bottom_right as byte, dim y_bottom_right as byte, dim radius as byte, dim color as byte)`
Description	Draws a rounded edge rectangle on Glcd.
Parameters	- `x_upper_left`: x coordinate of the upper left rectangle corner. Valid values: 0..127 - `y_upper_left`: y coordinate of the upper left rectangle corner. Valid values: 0..63 - `x_bottom_right`: x coordinate of the lower right rectangle corner. Valid values: 0..127 - `y_bottom_right`: y coordinate of the lower right rectangle corner. Valid values: 0..63 - `round_radius`: radius of the rounded edge. - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the rectangle border: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw a rounded edge rectangle between dots (5,5) and (40,40) with the radius of 12` `Glcd_Rectangle_Round_Edges(5, 5, 40, 40, 12, 1)`
Notes	None.

Glcd_Rectangle_Round_Edges_Fill

Prototype	`sub procedure Glcd_Rectangle_Round_Edges_Fill(dim x_upper_left as byte, dim y_upper_left as byte, dim x_bottom_right as byte, dim y_bottom_right as byte, dim radius as byte, dim color as byte)`
Description	Draws a filled rounded edge rectangle on Glcd with color.
Parameters	- `x_upper_left`: x coordinate of the upper left rectangle corner. Valid values: 0..127 - `y_upper_left`: y coordinate of the upper left rectangle corner. Valid values: 0..63 - `x_bottom_right`: x coordinate of the lower right rectangle corner. Valid values: 0..127 - `y_bottom_right`: y coordinate of the lower right rectangle corner. Valid values: 0..63 - `round_radius`: radius of the rounded edge - `color`: color parameter. Valid values: 0..2 The parameter color determines the `color` of the rectangle border: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draws a filled rounded edge rectangle between dots (5,5) and (40,40) with the radius of 12` `Glcd_Rectangle_Round_Edges_Fill(5, 5, 40, 40, 12, 1)`
Notes	None.

Glcd_Box

Prototype	`sub procedure Glcd_Box(dim x_upper_left, y_upper_left, x_bottom_right, y_bottom_right, color as byte)`
Description	Draws a box on Glcd. Parameters:
Parameters	- `x_upper_left`: x coordinate of the upper left box corner. Valid values: 0..127 - `y_upper_left`: y coordinate of the upper left box corner. Valid values: 0..63 - `x_bottom_right`: x coordinate of the lower right box corner. Valid values: 0..127 - `y_bottom_right`: y coordinate of the lower right box corner. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the box fill: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw a box between dots (5,15) and (20,40)` `Glcd_Box(5, 15, 20, 40, 1)`
Notes	None.

Glcd_Circle

Prototype	`sub procedure Glcd_Circle(dim x_center, y_center, radius as integer, dim color as byte)`
Description	Draws a circle on Glcd.
Parameters	- `x_center`: x coordinate of the circle center. Valid values: 0..127 - `y_center`: y coordinate of the circle center. Valid values: 0..63 - `radius`: radius size - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the circle line: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw a circle with center in (50,50) and radius=10` `Glcd_Circle(50, 50, 10, 1)`
Notes	None.

Glcd_Circle_Fill

Prototype	`sub procedure Glcd_Circle_Fill(dim x_center as integer, dim y_center as integer, dim radius as integer, dim color as byte)`
Description	Draws a filled circle on Glcd.
Parameters	- `x_center`: x coordinate of the circle center. Valid values: 0..127 - `y_center`: y coordinate of the circle center. Valid values: 0..63 - `radius`: radius size - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the circle line: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw a circle with center in (50,50) and radius=10` `Glcd_Circle_Fill(50, 50, 10, 1)`
Notes	None.

Glcd_Set_Font

Prototype	`sub procedure Glcd_Set_Font(dim byref const activeFont as byte, dim aFontWidth, aFontHeight as byte, dim aFontOffs as byte)`
Description	Sets font that will be used with Glcd_Write_Char and Glcd_Write_Text routines.
Parameters	- `activeFont`: font to be set. Needs to be formatted as an array of char - `aFontWidth`: width of the font characters in dots. - `aFontHeight`: height of the font characters in dots. - `aFontOffs`: number that represents difference between the mikroBasic PRO for dsPIC30/33 and PIC24 character set and regular ASCII set (eg. if ‘A’ is 65 in ASCII character, and ‘A’ is 45 in the mikroBasic PRO for dsPIC30/33 and PIC24 character set, aFontOffs is 20). Demo fonts supplied with the library have an offset of 32, which means that they start with space. The user can use fonts given in the file “__Lib_GLCDFonts” file located in the Uses folder or create his own fonts. List of supported fonts: - `Font_Glcd_System3x5` - `Font_Glcd_System5x7` - `Font_Glcd_5x7` - `Font_Glcd_Character8x7` For the sake of the backward compatibility, these fonts are supported also: - `System3x5` (equivalent to `Font_Glcd_System3x5`) - `FontSystem5x7_v2` (equivalent to `Font_Glcd_System5x7`) - `font5x7` (equivalent to `Font_Glcd_5x7`) - `Character8x7` (equivalent to `Font_Glcd_Character8x7`)
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Use the custom 5x7 font “myfont” which starts with space (32):` `Glcd_Set_Font(@myfont, 5, 7, 32)`
Notes	None.

Glcd_Write_Char

Prototype	`sub procedure Glcd_Write_Char(dim character, x_pos, page_num, color as byte)`
Description	Prints character on the Glcd.
Parameters	- `character`: character to be written - `x_pos`: character starting position on x-axis. Valid values: 0..(127-FontWidth) - `page_num`: the number of the page on which character will be written. Valid values: 0..7 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the character: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine. Use Glcd_Set_Font to specify the font for display; if no font is specified, then default `Font_Glcd_System5x7` font supplied with the library will be used.
Example	`' Write character 'C' on the position 10 inside the page 2:` `Glcd_Write_Char('C', 10, 2, 1)`
Notes	For x axis and page layout explanation see schematic at the bottom of this page.

Glcd_Write_Text

Prototype	`sub procedure Glcd_Write_Text(dim byref text as string, dim x_pos, page_num, color as byte)`
Description	Prints text on Glcd.
Parameters	- `text`: text to be written - `x_pos`: text starting position on x-axis. - `page_num`: the number of the page on which text will be written. Valid values: 0..7 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the text: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine. Use Glcd_Set_Font to specify the font for display; if no font is specified, then default `Font_Glcd_System5x7` font supplied with the library will be used.
Example	`' Write text "Hello world!" on the position 10 inside the page 2:` `Glcd_Write_Text("Hello world!", 10, 2, 1)`
Notes	For x axis and page layout explanation see schematic at the bottom of this page.

Glcd_Image

Prototype	`sub procedure Glcd_Image(dim byref constimage as byte)`
Description	Displays bitmap on Glcd.
Parameters	- `image`: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic PRO for dsPIC30/33 and PIC24 pointer to const and pointer to RAM equivalency).
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draw image my_image on Glcd` `Glcd_Image(my_image)`
Notes	Use the mikroBasic PRO for dsPIC30/33 and PIC24 integrated Glcd Bitmap Editor, **Tools > Glcd Bitmap Editor**, to convert image to a constant array suitable for displaying on Glcd.

Glcd_PartialImage

Prototype	`sub procedure Glcd_PartialImage(dim x_left, y_top, width, height, picture_width, picture_height as word, const image as ^byte)`
Description	Displays a partial area of the image on a desired location.
Parameters	- `x_left`: x coordinate of the desired location (upper left coordinate). - `y_top`: y coordinate of the desired location (upper left coordinate). - `width`: desired image width. - `height`: desired image height. - `picture_width`: width of the original image. - `picture_height`: height of the original image. - `image`: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic PRO for PIC pointer to const and pointer to RAM equivalency).
Returns	Nothing.
Requires	Glcd needs to be initialized, see Glcd_Init routine.
Example	`' Draws a 10x15 part of the image starting from the upper left corner on the coordinate (10,12). Original image size is 16x32.` `Glcd_PartialImage(10, 12, 10, 15, 16, 32, @image)`
Notes	Use the mikroBasic PRO for dsPIC30/33 and PIC24 integrated Glcd Bitmap Editor, **Tools > Glcd Bitmap Editor**, to convert image to a constant array suitable for displaying on Glcd.

Library Example

The following drawing demo tests advanced routines of the Glcd library.

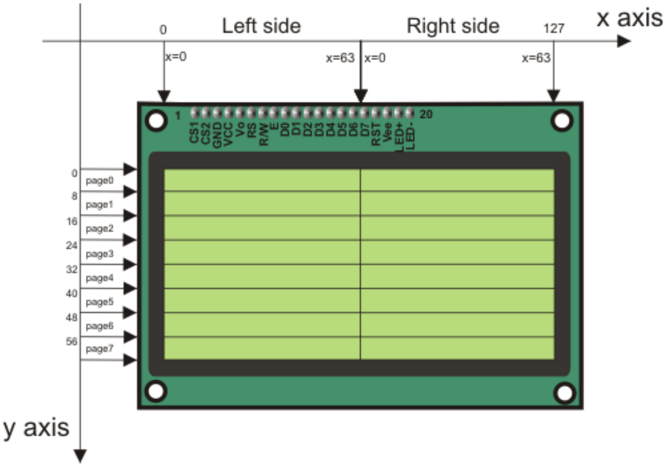
Copy Code To Clipboard

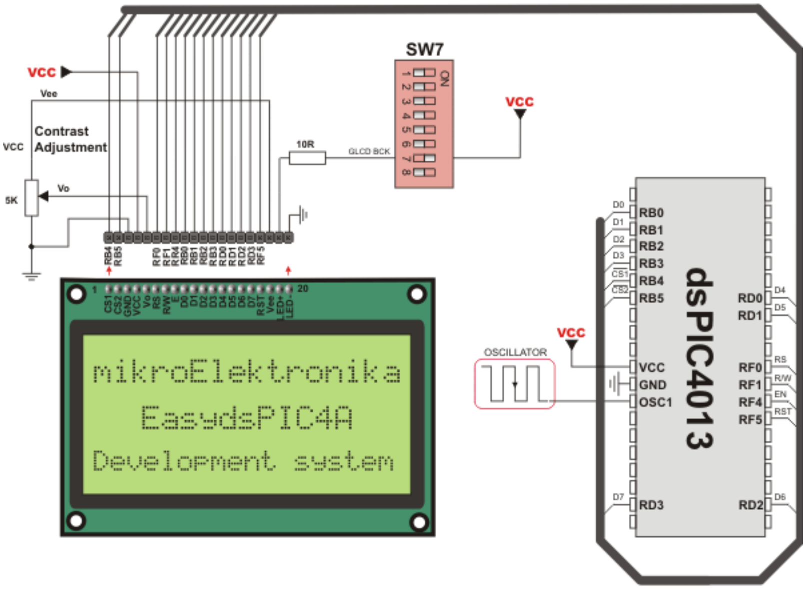
```
program GLCD_Test

' Glcd module connections
dim GLCD_D7 as sbit at RD3_bit
    GLCD_D6 as sbit at RD2_bit
    GLCD_D5 as sbit at RD1_bit
    GLCD_D4 as sbit at RD0_bit
    GLCD_D3 as sbit at RB3_bit
    GLCD_D2 as sbit at RB2_bit
    GLCD_D1 as sbit at RB1_bit
    GLCD_D0 as sbit at RB0_bit
    GLCD_D7_Direction as sbit at TRISD3_bit
    GLCD_D6_Direction as sbit at TRISD2_bit
    GLCD_D5_Direction as sbit at TRISD1_bit
    GLCD_D4_Direction as sbit at TRISD0_bit
    GLCD_D3_Direction as sbit at TRISB3_bit
    GLCD_D2_Direction as sbit at TRISB2_bit
    GLCD_D1_Direction as sbit at TRISB1_bit
    GLCD_D0_Direction as sbit at TRISB0_bit

dim GLCD_CS1 as sbit at LATB4_bit
    GLCD_CS2 as sbit at LATB5_bit
    GLCD_RS  as sbit at LATF0_bit
    GLCD_RW  as sbit at LATF1_bit
    GLCD_EN  as sbit at LATF4_bit
    GLCD_RST as sbit at LATF5_bit

dim GLCD_CS1_Direction as sbit at TRISB4_bit
    GLCD_CS2_Direction as sbit at TRISB5_bit
    GLCD_RS_Direction  as sbit at TRISF0_bit
    GLCD_RW_Direction  as sbit at TRISF1_bit
    GLCD_EN_Direction  as sbit at TRISF4_bit
    GLCD_RST_Direction as sbit at TRISF5_bit
' End Glcd module connections

dim counter as byte
    someText as char[18]

sub procedure Delay2S()                                ' 2 seconds delay function
  Delay_ms(2000)
end sub

main:
  #DEFINE COMPLETE_EXAMPLE        ' comment this line to make simpler/smaller example
  ADPCFG = 0xFFFF                 ' Configure AN pins as digital
```

```
Glcd_Init()                                       ‘ Initialize GLCD
Glcd_Fill(0x00)                                   ‘ Clear GLCD

while TRUE
  #IFDEF COMPLETE_EXAMPLE
  Glcd_Image(@truck_bmp)                          ‘ Draw image
  Delay2S() delay2S()
  #ENDIF
  Glcd_Fill(0x00)                                 ‘ Clear Glcd

  Glcd_Box(62,40,124,63,1)                        ‘ Draw box
  Glcd_Rectangle(5,5,84,35,1)                     ‘ Draw rectangle
  Glcd_Line(0, 0, 127, 63, 1)                     ‘ Draw line
  Delay2S()
  counter = 5
  while (counter <= 59)                       ‘ Draw horizontal and vertical lines
    Delay_ms(250)
    Glcd_V_Line(2, 54, counter, 1)
    Glcd_H_Line(2, 120, counter, 1)
    Counter = counter + 5
  wend

  Delay2S()

  Glcd_Fill(0x00)                                 ‘ Clear Glcd
  #IFDEF COMPLETE_EXAMPLE
  Glcd_Set_Font(@Character8x7, 8, 7, 32)          ‘ Choose font “Character8x7”
  Glcd_Write_Text(“mikroE”, 1, 7, 2)              ‘ Write string
  #ENDIF
  for counter = 1 to 10                           ‘ Draw circles
    Glcd_Circle(63,32, 3*counter, 1)
  next counter
  Delay2S()

  Glcd_Box(12,20, 70,57, 2)                       ‘ Draw box
  Delay2S()

  Glcd_Fill(0xFF)                                 ‘ Fill Glcd
  #IFDEF COMPLETE_EXAMPLE
  Glcd_Set_Font(@Character8x7, 8, 7, 32)          ‘ Change font
  someText = “8x7 Font”
  Glcd_Write_Text(someText, 5, 0, 2)              ‘ Write string
  delay2S()

  Glcd_Set_Font(@System3x5, 3, 5, 32)             ‘ Change font
  someText = “3X5 CAPITALS ONLY”
  Glcd_Write_Text(someText, 60, 2, 2)             ‘ Write string
  delay2S()

  Glcd_Set_Font(@font5x7, 5, 7, 32)               ‘ Change font
  someText = “5x7 Font”
  Glcd_Write_Text(someText, 5, 4, 2)              ‘ Write string
```

```
      delay2S()

    Glcd_Set_Font(@FontSystem5x7_v2, 5, 7, 32)      ' Change font
    someText = “5x7 Font (v2)”
    Glcd_Write_Text(someText, 51, 6, 2)             ' Write string
    delay2S()
    #ENDIF
  wend
end.
```

HW Connection





Glcd HW connection

I²C Library

The I²C full master I²C module is available with a number of the dsPIC30/33 and PIC24 MCU models. The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library which supports the master I²C mode.

Important:

- I²C library routines require you to specify the module you want to use. To select the desired I²C module, simply change the letter **x** in the routine prototype for a number from **1** to **3**.
- Number of I²C modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

Library Routines

- I2Cx_Init
- I2Cx_Start
- I2Cx_Restart
- I2Cx_Is_Idle
- I2Cx_Read
- I2Cx_Write
- I2Cx_Stop

I2Cx_Init

Prototype	`sub procedure I2Cx_Init(dim scl as longint)`
Description	Configures and initializes the desired I²C module with default settings. This function enables the I²C module by setting the I2CEN bit. The rest of the bits in I²C control register remains unchanged. Default initialization (after reset) of I²C module is: - continue operation in IDLE mode - IPMI mode disabled - 7-bit slave address - slew rate control enabled - general call address disabled - software or receive clock stretching disabled
Parameters	- `scl`: requested serial clock rate.
Returns	Nothing.
Requires	MCU with the I²C module.
Example	`' Initialize the I2C1 module with clock_rate=100000` `I2C1_Init(100000)`
Notes	Refer to the MCU's datasheet for correct values of the scl in respect with *Fosc*. I²C library routines require you to specify the module you want to use. To select the desired I²C module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of I²C modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

I2Cx_Start

Prototype	`sub procedure I2Cx_Start()`
Description	Determines if the I^{2}C bus is free and issues START signal.
Parameters	None.
Returns	Nothing.
Requires	MCU with at least one I^{2}C module. Used I^{2}C module must be initialized before using this function. See I2Cx_Init routine.
Example	`' Issue START signal` `I2C1_Start()`
Notes	I^{2}C library routines require you to specify the module you want to use. To select the desired I^{2}C module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of I^{2}C modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

I2Cx_Restart

Prototype	`sub procedure I2Cx_Restart()`
Description	Issues repeated START signal.
Parameters	None.
Returns	Nothing.
Requires	MCU with at least one I^{2}C module. Used I^{2}C module must be initialized before using this function. See I2Cx_Init routine.
Example	`' Issue RESTART signal` `I2C1_Restart()`
Notes	I^{2}C library routines require you to specify the module you want to use. To select the desired I^{2}C module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of I^{2}C modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

I2Cx_Is_Idle

Prototype	`sub function I2Cx_Is_Idle() as word`
Description	Waits for the I^{2}C bus to become free. This is a blocking function.
Parameters	None.
Returns	- `0` if I^{2}C bus is free. - `1` if I^{2}C bus is not free.
Requires	MCU with at least one I^{2}C module. Used I^{2}C module must be initialized before using this function. See I2Cx_Init routine.

Example

```
dim data_ as byte
...
if I2C1_Is_Idle() then
  I2C1_Write(data_)
end if
...
```

Notes	I^{2}C library routines require you to specify the module you want to use. To select the desired I^{2}C module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of I^{2}C modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

I2Cx_Read

Prototype	`sub function I2Cx_Read(dim ack as word) as byte`
Description	Reads a byte from the I^{2}C bus.
Parameters	- `ack`: acknowledge signal parameter. If the `ack = 0`, acknowledge signal will be sent after reading, otherwise the *not acknowledge* signal will be sent.
Returns	Received data.
Requires	MCU with at least one I^{2}C module. Used I^{2}C module must be initialized before using this function. See I2Cx_Init routine. Also, START signal needs to be issued in order to use this function. See I2Cx_Start.

Example

```
dim take as byte
...
' Read data and send the not_acknowledge signal
take = I2C1_Read(1)
```

Notes	I^{2}C library routines require you to specify the module you want to use. To select the desired I^{2}C module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of I^{2}C modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

I2Cx_Write

Prototype	`sub function I2Cx_Write(dim data_ as byte) as word`
Description	Sends data byte via the I^{2}C bus.
Parameters	- `data_`: data to be sent
Returns	- `0` if there were no errors. - `1` if write collision was detected on the I^{2}C bus.
Requires	MCU with at least one I^{2}C module. Used I^{2}C module must be initialized before using this function. See I2Cx_Init routine. Also, START signal needs to be issued in order to use this function. See I2Cx_Start.

Example

```
dim data_ as byte
    error as word
...
error = I2C1_Write(data_)
error = I2C1_Write(0xA3)
```

Notes	I^{2}C library routines require you to specify the module you want to use. To select the desired I^{2}C module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of I^{2}C modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

I2Cx_Stop

Prototype	`sub procedure I2Cx_Stop()`
Description	Issues STOP signal.
Parameters	None.
Returns	Nothing.
Requires	MCU with at least one I^{2}C module. Used I^{2}C module must be initialized before using this function. See I2Cx_Init routine.

Example

```
' Issue STOP signal
I2C1_Stop()
```

Notes	I^{2}C library routines require you to specify the module you want to use. To select the desired I^{2}C module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of I^{2}C modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

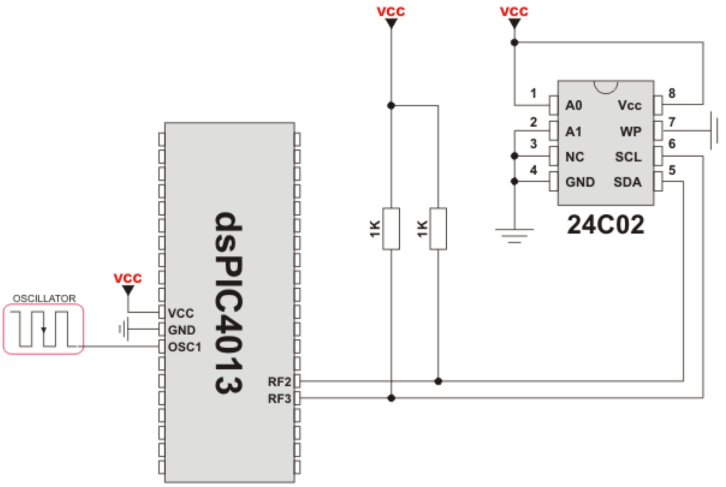
Library Example

This code demonstrates working with the I^{2}C library. Program sends data to EEPROM (data is written at the address 2). After that, program reads data from the same EEPROM address and displays it on PORTB for visual check. See the figure below how to interface the 24C02 to dsPIC30/33 and PIC24.

Copy Code To Clipboard

```
program I2C_Simple

main:
  ADPCFG = 0xFFFF         ' initialize AN pins as digital

  LATB = 0
  TRISB = 0               ' Configure PORTB as output

  I2C1_Init(100000)       ' initialize I2C communication
  I2C1_Start()            ' issue I2C start signal
  I2C1_Write(0xA2)        ' send byte via I2C  (device address + W)
  I2C1_Write(2)           ' send byte (address of EEPROM location)
  I2C1_Write(0xAA)        ' send data (data to be written)
  I2C1_Stop()             ' issue I2C stop signal

  Delay_100ms()

  I2C1_Start()            ' issue I2C start signal
  I2C1_Write(0xA2)        ' send byte via I2C  (device address + W)
  I2C1_Write(2)           ' send byte (data address)
  I2C1_Restart()          ' issue I2C signal repeated start
  I2C1_Write(0xA3)        ' send byte (device address + R)
  PORTB = I2C1_Read(1)    ' Read the data (NO acknowledge)
  I2C1_Stop()             ' issue I2C stop signal
end.
```

HW Connection



Interfacing 24c02 to dsPIC30/33 and PIC24 via I^{2}C

Keypad Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for working with 4x4 keypad. The library routines can also be used with 4x1, 4x2, or 4x3 keypad. For connections explanation see schematic at the bottom of this page.

External dependencies of Keypad Library

The following variable must be defined in all projects using Keypad Library:	**Description:**	**Example:**
`dim keypadPort as word sfr external`	Keypad Port.	`dim keypadPort as word at PORTB`
`dim keypadPort_Direction as word sfr external`	Keypad Port.	`dim keypadPort_Direction as word at TRISB`

Library Routines

- Keypad_Init
- Keypad_Key_Press
- Keypad_Key_Click

Keypad_Init

Prototype	`sub procedure Keypad_Init()`
Description	Initializes given port for working with keypad.
Parameters	None.
Returns	Nothing.
Requires	Global variable: - `keypadPort` - Keypad port must be defined before using this function.
Example	see below
Notes	The Keypad library uses lower byte (bits <7..0>) of `keypadPort`.

Example

```
' Keypad module connections
dim keypadPort as word at PORTB
dim keypadPort_Direction as word at TRISB
' End Keypad module connections
...
Keypad_Init()
```

Keypad_Key_Press

Prototype	`sub function Keypad_Key_Press() as word`
Description	Reads the key from keypad when key gets pressed.
Parameters	None.
Returns	The code of a pressed key (1..16). If no key is pressed, returns 0.
Requires	Port needs to be initialized for working with the Keypad library, see Keypad_Init.

Example	`dim kp as word`
	`...`
	`kp = Keypad_Key_Press()`

Notes	None

Keypad_Key_Click

Prototype	`sub function Keypad_Key_Click() as word`
Description	Call to `Keypad_Key_Click` is a blocking call: the function waits until some key is pressed and released. When released, the function returns 1 to 16, depending on the key. If more than one key is pressed simultaneously the function will wait until all pressed keys are released. After that the function will return the code of the first pressed key.
Parameters	None.
Returns	The code of a clicked key (1..16). If no key is clicked, returns 0.
Requires	Port needs to be initialized for working with the Keypad library, see Keypad_Init.
Example	`kp = Keypad_Key_Click()`
Notes	None

Library Example

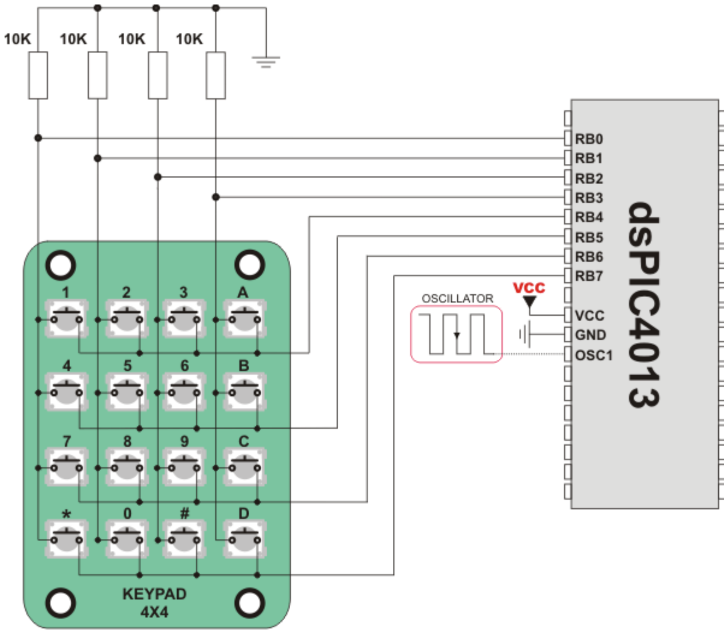
The following code can be used for testing the keypad. It is written for keypad_4x3 or _4x4. The code returned by the keypad functions (1..16) is transformed into ASCII codes [0..9,A..F], and then sent via UART1.

Copy Code To Clipboard

```
program Keypad_Test
dim kp, oldstate as byte
    txt as char[6]

‘ Keypad module connections
dim keypadPort as word at PORTB
dim keypadPort_Direction as word at TRISB
‘ End Keypad module connections

main:
  ADPCFG = 0xFFFF
  oldstate = 0
  UART1_Init(9600)
  Delay_ms(100)
  Keypad_Init()                        ‘ Initialize Keypad

  UART1_Write_Text(“Press any key on your kaypad...”)
  UART1_Write(10)
  UART1_Write(13)

  while TRUE
    kp = 0                             ‘ Reset key code variable

    ‘ Wait for key to be pressed and released
    while ( kp = 0 )
      kp = Keypad_Key_Click()          ‘ Store key code in kp variable
    wend
    ‘ Prepare value for output, transform key to it’s ASCII value
    select case kp
‘    case 10
‘      kp = 42  ‘ “*”       ‘ Uncomment this block for keypad4x3
‘    case 11
‘      kp = 48  ‘ “0”
‘    case 12
‘      kp = 35  ‘ “#”
‘    case else
‘      kp = kp + 48

    case 1
      kp = 49  ‘ 1          ‘ Uncomment this block for keypad4x4
    case 2
      kp = 50  ‘ 2
    case 3
      kp = 51  ‘ 3
    case 4
      kp = 65  ‘ A
    case 5
      kp = 52  ‘ 4
    case 6
      kp = 53  ‘ 5
```

```
    case 7
      kp = 54  ' 6
    case 8
      kp = 66  ' B
    case 9
      kp = 55  ' 7
    case 10
      kp = 56  ' 8
    case 11
      kp = 57  ' 9
    case 12
      kp = 67  ' C
    case 13
      kp = 42  ' *
    case 14
      kp = 48  ' 0
    case 15
      kp = 35  ' #
    case 16
      kp = 68  ' D

    end select

    UART1_Write_Text(“Key pressed: ”)
    UART1_Write(kp)                       ' Send value of pressed button to UART
    UART1_Write(10)
    UART1_Write(13)
  wend
end.
```

HW Connection



4x4 Keypad connection scheme

Lcd Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for communication with Lcds (with HD44780 compliant controllers) through the 4-bit interface. An example of Lcd connections is given on the schematic at the bottom of this page.

For creating a set of custom Lcd characters use Lcd Custom Character Tool.

Library Dependency Tree

External dependencies of Lcd Library

The following variables must be defined in all projects using Lcd Library :	Description:	Example:
`dim LCD_RS as sbit sfr external`	Register Select line.	`dim LCD_RS as sbit at LATD0_bit`
`dim LCD_EN as sbit sfr external`	Enable line.	`dim LCD_EN as sbit at LATD1_bit`
`dim LCD_D4 as sbit sfr external`	Data 4 line.	`dim LCD_D4 as sbit at LATB0_bit`
`dim LCD_D5 as sbit sfr external`	Data 5 line.	`dim LCD_D5 as sbit at LATB1_bit`
`dim LCD_D6 as sbit sfr external`	Data 6 line.	`dim LCD_D6 as sbit at LATB2_bit`
`dim LCD_D7 as sbit sfr external`	Data 7 line.	`dim LCD_D7 as sbit at LATB3_bit`
`dim LCD_RS_Direction as sbit sfr external`	Register Select direction pin.	`dim LCD_RS_Direction as sbit at TRISD0_bit`
`dim LCD_EN_Direction as sbit sfr external`	Enable direction pin.	`dim LCD_EN_Direction as sbit at TRISD1_bit`
`dim LCD_D4_Direction as sbit sfr external`	Data 4 direction pin.	`dim LCD_D4_Direction as sbit at TRISB0_bit`
`dim LCD_D5_Direction as sbit sfr external`	Data 5 direction pin.	`dim LCD_D5_Direction as sbit at TRISB1_bit`
`dim LCD_D6_Direction as sbit sfr external`	Data 6 direction pin.	`dim LCD_D6_Direction as sbit at TRISB2_bit`
`dim LCD_D7_Direction as sbit sfr external`	Data 7 direction pin.	`dim LCD_D7_Direction as sbit at TRISB3_bit`

Library Routines

- Lcd_Init
- Lcd_Out
- Lcd_Out_Cp
- Lcd_Chr
- Lcd_Chr_Cp
- Lcd_Cmd

Lcd_Init

Prototype	`sub procedure Lcd_Init()`
Description	Initializes Lcd module.
Parameters	None.
Returns	Nothing.
Requires	Global variables: - `LCD_RS`: Register Select (data/instruction) signal pin - `LCD_EN`: Enable signal pin - `LCD_D4`: Data bit 4 - `LCD_D5`: Data bit 5 - `LCD_D6`: Data bit 6 - `LCD_D7`: Data bit 7 - `LCD_RS_Direction`: Direction of the Register Select pin - `LCD_EN_Direction`: Direction of the Enable signal pin - `LCD_D4_Direction`: Direction of the Data 4 pin - `LCD_D5_Direction`: Direction of the Data 5 pin - `LCD_D6_Direction`: Direction of the Data 6 pin - `LCD_D7_Direction`: Direction of the Data 7 pin must be defined before using this function.
Example	
Notes	None

Example

```
' LCD module connections
dim LCD_RS as sbit at LATD0_bit
dim LCD_EN as sbit at LATD1_bit
dim LCD_D4 as sbit at LATB0_bit
dim LCD_D5 as sbit at LATB1_bit
dim LCD_D6 as sbit at LATB2_bit
dim LCD_D7 as sbit at LATB3_bit

dim LCD_RS_Direction as sbit at TRISD0_bit
dim LCD_EN_Direction as sbit at TRISD1_bit
dim LCD_D4_Direction as sbit at TRISB0_bit
dim LCD_D5_Direction as sbit at TRISB1_bit
dim LCD_D6_Direction as sbit at TRISB2_bit
dim LCD_D7_Direction as sbit at TRISB3_bit
' End LCD module connections
...
Lcd_Init()
```

Lcd_Out

Prototype	`sub procedure Lcd_Out(dim row, column as word, dim byref text as string)`
Description	Prints text on Lcd starting from specified position. Both string variables and literals can be passed as a text.
Parameters	- `row`: starting position row number - `column`: starting position column number - `text`: text to be written
Returns	Nothing.
Requires	The Lcd module needs to be initialized. See Lcd_Init routine.
Example	`' Write text “Hello!” on Lcd starting from row 1, column 3:` `Lcd_Out(1, 3, “Hello!”)`
Notes	None

Lcd_Out_Cp

Prototype	`sub procedure Lcd_Out_Cp(dim byref text as string)`
Returns	Nothing.
Description	Prints text on Lcd at current cursor position. Both string variables and literals can be passed as a text.
Parameters	- `text`: text to be written
Requires	The Lcd module needs to be initialized. See Lcd_Init routine.
Example	`' Write text “Here!” at current cursor position:` `Lcd_Out_Cp(“Here!”)`
Notes	None

Lcd_Chr

Prototype	`sub procedure Lcd_Chr(dim row, column as word, dim out_char as byte)`
Description	Prints character on Lcd at specified position. Both variables and literals can be passed as a character.
Parameters	- `row`: writing position row number - `column`: writing position column number - `out_char`: character to be written
Returns	Nothing.
Requires	The Lcd module needs to be initialized. See Lcd_Init routine.
Example	`' Write character “i” at row 2, column 3:` `Lcd_Chr(2, 3, “i”)`
Notes	None

Lcd_Chr_Cp

Prototype	`sub procedure Lcd_Chr_Cp(dim out_char as byte)`
Description	Prints character on Lcd at current cursor position. Both variables and literals can be passed as a character.
Parameters	- `out_char`: character to be written
Returns	Nothing.
Requires	The Lcd module needs to be initialized. See Lcd_Init routine.
Example	`' Write character “e” at current cursor position:` `Lcd_Chr_Cp(“e”)`
Notes	None

Lcd_Cmd

Prototype	`sub procedure Lcd_Cmd(dim out_char as byte)`
Description	Sends command to Lcd.
Parameters	- `out_char`: command to be sent
Returns	Nothing.
Requires	The Lcd module needs to be initialized. See Lcd_Init table.
Example	`' Clear Lcd display:` `Lcd_Cmd(_LCD_CLEAR)`
Notes	Predefined constants can be passed to the function, see Available Lcd Commands.

Available Lcd Commands

Lcd Command	**Purpose**
`_LCD_FIRST_ROW`	Move cursor to the 1st row
`_LCD_SECOND_ROW`	Move cursor to the 2nd row
`_LCD_THIRD_ROW`	Move cursor to the 3rd row
`_LCD_FOURTH_ROW`	Move cursor to the 4th row
`_LCD_CLEAR`	Clear display
`_LCD_RETURN_HOME`	Return cursor to home position, returns a shifted display to its original position. Display data RAM is unaffected.
`_LCD_CURSOR_OFF`	Turn off cursor
`_LCD_UNDERLINE_ON`	Underline cursor on
`_LCD_BLINK_CURSOR_ON`	Blink cursor on
`_LCD_MOVE_CURSOR_LEFT`	Move cursor left without changing display data RAM
`_LCD_MOVE_CURSOR_RIGHT`	Move cursor right without changing display data RAM
`_LCD_TURN_ON`	Turn Lcd display on
`_LCD_TURN_OFF`	Turn Lcd display off
`_LCD_SHIFT_LEFT`	Shift display left without changing display data RAM
`_LCD_SHIFT_RIGHT`	Shift display right without changing display data RAM

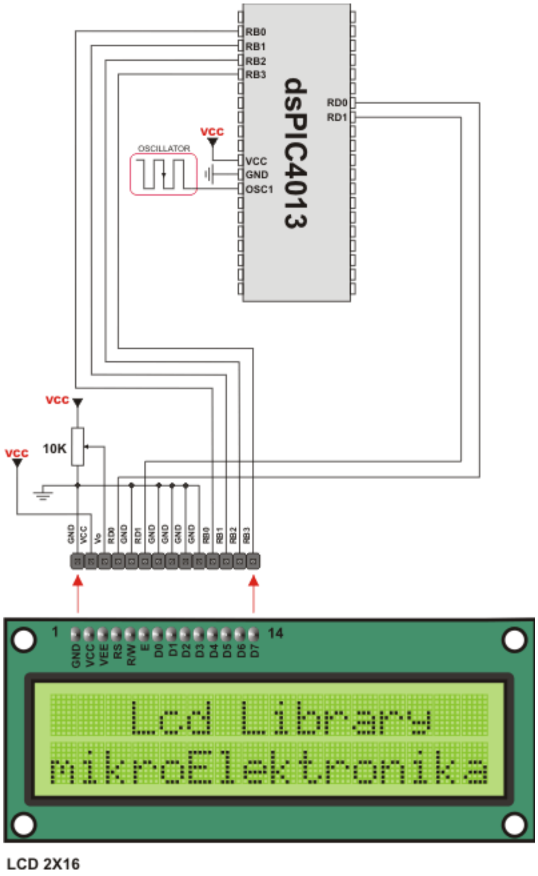
Library Example

The following code demonstrates usage of the Lcd Library routines:

Copy Code To Clipboard

```
program  Lcd_Test

' LCD module connections
dim LCD_RS as sbit at LATD0_bit
dim LCD_EN as sbit at LATD1_bit
dim LCD_D4 as sbit at LATB0_bit
dim LCD_D5 as sbit at LATB1_bit
dim LCD_D6 as sbit at LATB2_bit
dim LCD_D7 as sbit at LATB3_bit

dim LCD_RS_Direction as sbit at TRISD0_bit
dim LCD_EN_Direction as sbit at TRISD1_bit
dim LCD_D4_Direction as sbit at TRISB0_bit
dim LCD_D5_Direction as sbit at TRISB1_bit
dim LCD_D6_Direction as sbit at TRISB2_bit
dim LCD_D7_Direction as sbit at TRISB3_bit
' End LCD module connections

dim txt1 as char[16]
    txt2 as char[9]
    txt3 as char[8]
    txt4 as char[7]
    i as byte                        ' Loop variable

sub procedure Move_Delay()           ' Function used for text moving
  Delay_ms(500)                      ' You can change the moving speed here
end sub

main:
  ADPCFG = 0xFFFF                    ' Configure AN pins as digital I/O

  txt1 = "mikroElektronika"
  txt2 = "EasydsPIC4A"
  txt3 = "Lcd4bit"
  txt4 = "example"

Lcd_Init()                           ' Initialize LCD
  Lcd_Cmd(_LCD_CLEAR)                ' Clear display
  Lcd_Cmd(_LCD_CURSOR_OFF)           ' Cursor off
  LCD_Out(1,6,txt3)                  ' Write text in first row
  LCD_Out(2,6,txt4)                  ' Write text in second row
  Delay_ms(2000)
  Lcd_Cmd(_LCD_CLEAR)                ' Clear display

  LCD_Out(1,1,txt1)                  ' Write text in first row
  Lcd_Out(2,3,txt2)                  ' Write text in second row
```

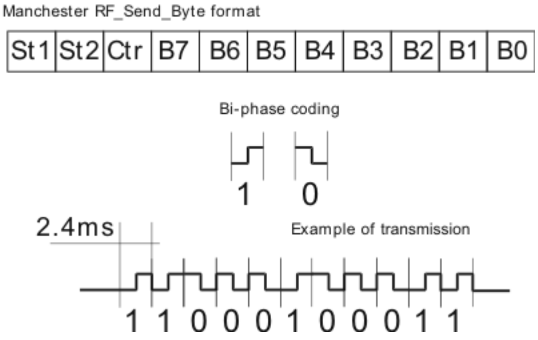
```
  Delay_ms(500)

  ' Moving text
  for i = 0 to 4                    ' Move text to the right 4 times
    Lcd_Cmd(_LCD_SHIFT_RIGHT)
    Move_Delay()
  next i

  while TRUE                        ' Endless loop
    for i = 0 to 8                  ' Move text to the left 7 times
      Lcd_Cmd(_LCD_SHIFT_LEFT)
      Move_Delay()
    next i

    for i = 0 to 8                  ' Move text to the right 7 times
      Lcd_Cmd(_LCD_SHIFT_RIGHT)
      Move_Delay()
    next i
  wend
end.
```



Lcd HW connection

Manchester Code Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for handling Manchester coded signals. The Manchester code is a code in which data and clock signals are combined to form a single self-synchronizing data stream; each encoded bit contains a transition at the midpoint of a bit period, the direction of transition determines whether the bit is 0 or 1; the second half is the true bit value and the first half is the complement of the true bit value (as shown in the figure below).



Important:

- The Manchester receive routines are blocking calls (`Man_Receive_Init` and `Man_Synchro`). This means that MCU will wait until the task has been performed (e.g. byte is received, synchronization achieved, etc).
- Manchester code library implements time-based activities, so interrupts need to be disabled when using it.

External dependencies of Manchester Code Library

The following variables must be defined in all projects using Manchester Code Library:	Description:	Example:
`dim MANRXPIN as sbit sfr external`	Receive line.	`dim MANRXPIN as sbit at RF0_bit`
`dim MANTXPIN as sbit sfr external`	Transmit line.	`dim MANTXPIN as sbit at LATF1_bit`
`dim MANRXPIN_Direction as sbit sfr external`	Direction of the Receive pin.	`dim MANRXPIN_Direction as sbit at TRISF0_bit`
`dim MANTXPIN_Direction as sbit sfr external`	Direction of the Transmit pin.	`dim MANTXPIN_Direction as sbit at TRISF1_bit`

Library Routines

- Man_Receive_Init
- Man_Receive
- Man_Send_Init
- Man_Send
- Man_Synchro
- Man_Break

The following routines are for the internal use by compiler only:

- Manchester_0
- Manchester_1
- Manchester_Out

Man_Receive_Init

Prototype	`sub function Man_Receive_Init() as word`
Description	The function configures Receiver pin. After that, the function performs synchronization procedure in order to retrieve baud rate out of the incoming signal.
Parameters	None.
Returns	- `0` - if initialization and synchronization were successful. - `1` - upon unsuccessful synchronization. - `255` - upon user abort.
Requires	Global variables: - `MANRXPIN` : Receive line - `MANRXPIN_Direction` : Direction of the receive pin must be defined before using this function.
Example	see below
Notes	In case of multiple persistent errors on reception, the user should call this routine once again or Man_Synchro routine to enable synchronization.

Example

```
' Initialize Receiver
dim MANRXPIN as sbit at RF0_bit
dim MANRXPIN_Direction as sbit at TRISF0_bit
...
Man_Receive_Init()
```

Man_Receive

Prototype	`sub function Man_Receive(dim byref error as word) as byte`
Description	The function extracts one byte from incoming signal.
Parameters	- `error`: error flag. If signal format does not match the expected, the `error` flag will be set to non-zero.
Returns	A byte read from the incoming signal.
Requires	To use this function, the user must prepare the MCU for receiving. See Man_Receive_Init routines.

Example

```
dim data_, error as word
...
error = 0
data_ = 0
data_ = Man_Receive(error)
if (error <> 0) then
    ' error handling
end if
```

Notes	None.

Man_Send_Init

Prototype	`sub procedure Man_Send_Init()`
Description	The function configures Transmitter pin.
Parameters	None.
Returns	Nothing.
Requires	Global variables: - `MANTXPIN` : Transmit line - `MANTXPIN_Direction` : Direction of the transmit pin must be defined before using this function.

Example

```
' Initialize Transmitter:
dim MANTXPIN as sbit at LATF1_bit
dim MANTXPIN_Direction as sbit at TRISF1_bit
...
Man_Send_Init()
```

Notes	None.

Man_Send

Prototype	`sub procedure Man_Send(dim data as byte)`
Description	Sends one byte.
Parameters	- `tr_data`: data to be sent
Returns	Nothing.
Requires	To use this function, the user must prepare the MCU for sending. See Man_Send_Init routine.

Example	`dim msg as byte`
	`...`
	`Man_Send(msg)`

Notes	Baud rate used is 500 bps.

Man_Synchro

Prototype	`sub function Man_Synchro() as word`
Description	Measures half of the manchester bit length with 10us resolution.
Parameters	None.
Returns	- `0` - if synchronization was not successful. - Half of the manchester bit length, given in multiples of 10us - upon successful synchronization.
Requires	To use this function, you must first prepare the MCU for receiving. See Man_Receive_Init.

Example	`dim man__half_bit_len as word`
	`...`
	`man__half_bit_len = Man_Synchro()`

Notes	None.

Man_Break

Prototype	`sub procedure Man_Break()`
Description	Man_Receive is blocking routine and it can block the program flow. Call this routine from interrupt to unblock the program execution. This mechanism is similar to WDT.
Parameters	None.
Returns	Nothing.
Requires	Nothing.

Example

```
dim data1, error, counter as byte

sub procedure Timer1Int() org IVT_ADDR_T1INTERRUPT
  counter = 0
  if (counter >= 20) then
    Man_Break()
    counter = 0                ' reset counter
  else
    Inc(counter)               ' increment counter
  end if

  T1IF_bit = 0                 ' Clear Timer1 overflow interrupt flag
end sub

main:

  ...

  if (Man_Receive_Init() = 0)
    ...
  end if

  ...

  ' try Man_Receive with blocking prevention mechanism
  IPC0 = IPC0 or 0x1000        ' Interrupt priority level = 1
  T1IE_bit = 1                 ' Enable Timer1 interrupts
  T1CON = 0x8030       ' Timer1 ON, internal clock FCY, prescaler 1:256

  data1 = Man_Receive(@error)
  T1IE_bit = 0         ' Disable Timer1 interrupts
end.
```

Notes	Interrupts should be disabled before using Manchester routines again (see note at the top of this page).

Library Example

The following code is code for the Manchester receiver, it shows how to use the Manchester Library for receiving data:

Copy Code To Clipboard

```
program Manchester_Receiver

' LCD module connections
dim LCD_RS as sbit at LATD0_bit
    LCD_EN as sbit at LATD1_bit
    LCD_D4 as sbit at LATB0_bit
    LCD_D5 as sbit at LATB1_bit
    LCD_D6 as sbit at LATB2_bit
    LCD_D7 as sbit at LATB3_bit

dim LCD_RS_Direction as sbit at TRISD0_bit
    LCD_EN_Direction as sbit at TRISD1_bit
    LCD_D4_Direction as sbit at TRISB0_bit
    LCD_D5_Direction as sbit at TRISB1_bit
    LCD_D6_Direction as sbit at TRISB2_bit
    LCD_D7_Direction as sbit at TRISB3_bit
' End LCD module connections


' Manchester module connections
dim MANRXPIN as sbit at RF0_bit
    MANRXPIN_Direction as sbit at TRISF0_bit
    MANTXPIN as sbit at LATF1_bit
    MANTXPIN_Direction as sbit at TRISF1_bit
' End Manchester module connections

dim error_flag, ErrorCount, counter, temp as byte

main:
  ErrorCount = 0
  counter = 0

  ADPCFG = 0xFFFF                        ' Configure AN pins as digital I/O
  Lcd_Init()                             ' Initialize LCD
  Lcd_Cmd(_LCD_CLEAR)                    ' Clear LCD display

  Man_Receive_Init()                     ' Initialize Receiver

  while TRUE                             ' Endless loop
    Lcd_Cmd(_LCD_FIRST_ROW)              ' Move cursor to the 1st row

    while TRUE                           ' Wait for the "start" byte
      temp = Man_Receive(error_flag)     ' Attempt byte receive
      if (temp = 0x0B) then              ' "Start" byte, see Transmitter example
        break                            ' We got the starting sequence
      end if
      if (error_flag <> 0) then          ' Exit so we do not loop forever
        break
       end if
    wend
```

```
      do
      temp = Man_Receive(error_flag)     ' Attempt byte receive
      if (error_flag <> 0) then          ' If error occured
        Lcd_Chr_CP(“?”)                  ' Write question mark on LCD
        Inc(ErrorCount)                  ' Update error counter
        if (ErrorCount > 20) then        ' In case of multiple errors
          temp = Man_Synchro()           ' Try to synchronize again
          ‘Man_Receive_Init()          ' Alternative, try to Initialize Receiver again
          ErrorCount = 0                 ' Reset error counter
        end if
      else                               ' No error occured
        if (temp <> 0x0E) then   ' If “End” byte was received(see Transmitter example)
                                         ' do not write anymore received byte on Lcd
          Lcd_Chr_CP(temp)               ' else write character on Lcd
          Inc(counter)                 ' Counts how many chars have been written on Lcd
          if counter = 25 then           ' If there were more then 25 characters
                                         ' synchronization is off
            Lcd_Cmd(_LCD_CLEAR)          ' Clear the Lcd of garbled communication
            temp = Man_Synchro()         ' Try to synchronize again
          end if
        else
          counter = 0                    ' reset counter
        end if
        Delay_ms(25)
      end if
    loop until ( temp = 0x0E )
  wend                                   ' If “End” byte was received exit do loop
end.
```

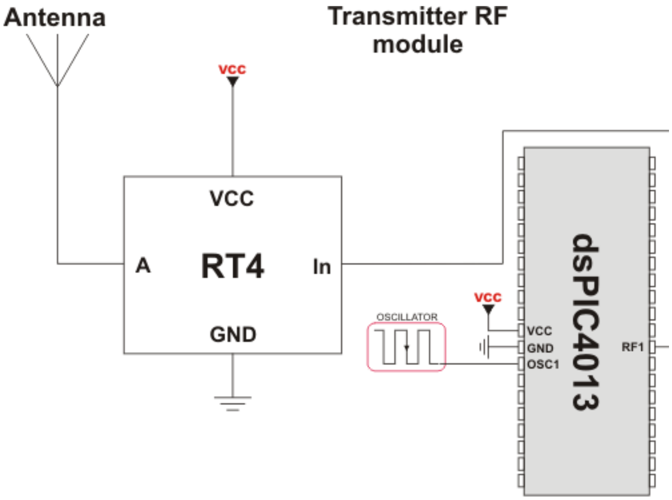
The following code is code for the Manchester transmitter, it shows how to use the Manchester Library for transmitting data:

Copy Code To Clipboard

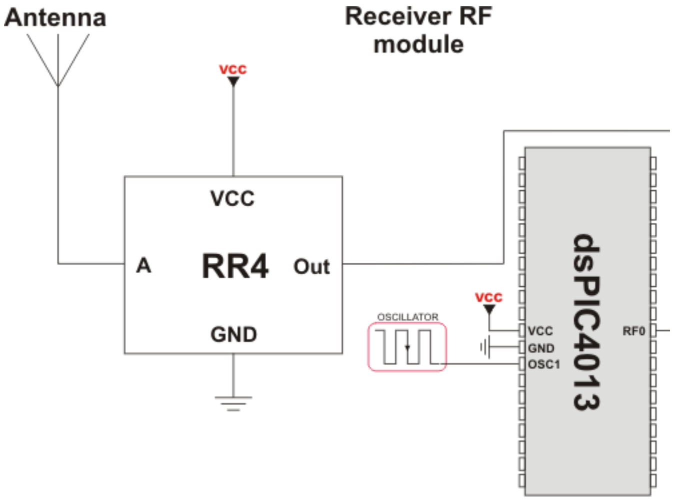
```
program Manchester_Transmitter

' Manchester module connections
dim MANRXPIN as sbit at RF0_bit
    MANRXPIN_Direction as sbit at TRISF0_bit
    MANTXPIN as sbit at LATF1_bit
    MANTXPIN_Direction as sbit at TRISF1_bit
' End Manchester module connections

dim index, character as byte
    s1 as char[17]

main:
  s1 = “mikroElektronika”
  ADPCFG = 0xFFFF               ' Configure AN pins as digital I/O

  Man_Send_Init()               ' Initialize transmitter

  while TRUE                    ' Endless loop
    Man_Send(0x0B)              ' Send “start” byte
    Delay_ms(100)               ' Wait for a while
```

```
    character = s1[0]           ' Take first char from string
    index = 0                   ' Initialize index variable
    while (character <> 0)      ' String ends with zero
      Man_Send(character)       ' Send character
      Delay_ms(90)              ' Wait for a while
      Inc(index)                ' Increment index variable
      character = s1[index]     ' Take next char from string
    wend
    Man_Send(0x0E)              ' Send "end" byte
    Delay_ms(1000)
  wend
end.
```

Connection Example



Simple Transmitter connection



Simple Receiver connection

Multi Media Card Library

The Multi Media Card (MMC) is a Flash memory card standard. MMC cards are currently available in sizes up to and including 32 GB and are used in cellular phones, digital audio players, digital cameras and PDA's.
mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for accessing data on Multi Media Card via SPI communication. This library also supports SD (Secure Digital) and high capacity SDHC (Secure Digital High Capacity) memory cards.

Secure Digital Card

Secure Digital (SD) is a Flash memory card standard, based on the older Multi Media Card (MMC) format.
SD cards are currently available in sizes of up to and including 2 GB, and are used in digital cameras, digital camcorders, handheld computers, media players, mobile phones, GPS receivers, video games and PDAs.

Secure Digital High Capacity Card

SDHC (Secure Digital High Capacity, SD 2.0) is an extension of the SD standard which increases card's storage capacity up to 32 GB by using sector addressing instead of byte addressing in the previous SD standard.
SDHC cards share the same physical and electrical form factor as older (SD 1.x) cards, allowing SDHC-devices to support both newer SDHC cards and older SD-cards. The current standard limits the maximum capacity of an SDHC card to 32 GB.

Important:

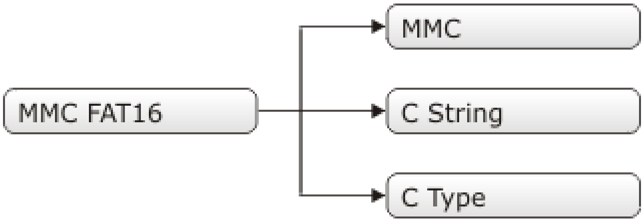
- Routines for file handling can be used only with FAT16 file system.
- Library functions create and read files from the root directory only.
- Library functions populate both FAT1 and FAT2 tables when writing to files, but the file data is being read from the FAT1 table only; i.e. there is no recovery if the FAT1 table gets corrupted.
- If MMC/SD card has Master Boot Record (MBR), the library will work with the first available primary (logical) partition that has non-zero size. If MMC/SD card has Volume Boot Record (i.e. there is only one logical partition and no MBRs), the library works with entire card as a single partition. For more information on MBR, physical and logical drives, primary/secondary partitions and partition tables, please consult other resources, e.g. Wikipedia and similar.
- Before write operation, make sure you don't overwrite boot or FAT sector as it could make your card on PC or digital camera unreadable. Drive mapping tools, such as Winhex, can be of a great assistance.
- Library uses SPI module for communication. The user must initialize the appropriate SPI module before using the MMC Library.
- For MCUs with multiple SPI modules it is possible to initialize all of them and then switch by using the `SPI_Set_Active()` function. See the SPI Library functions.

The SPI module has to be initialized through `SPIx_Init_Advanced` routine with the following parameters:

- SPI Master
- 8bit mode
- secondary prescaler 1
- primary prescaler 64
- Slave Select disabled
- data sampled in the middle of data output time
- clock idle high
- Serial output data changes on transition from active clock state to idle clock state

Tip: Once the MMC/SD card is initialized, SPI module can be reinitialized at higher a speed. See the Mmc_Init and Mmc_Fat_Init routines.

Library Dependency Tree



External dependencies of MMC Library

The following variable must be defined in all projects using MMC library:	Description:	Example:
`dim Mmc_Chip_Select as sbit sfr external`	Chip select pin.	`dim Mmc_Chip_Select as sbit at LATF0_bit`
`dim Mmc_Chip_Select_Direction as sbit sfr external`	Direction of the chip select pin.	`dim Mmc_Chip_Select_Direction as sbit at TRISF0_bit`

Library Routines

- Mmc_Init
- Mmc_Read_Sector
- Mmc_Write_Sector
- Mmc_Read_Cid
- Mmc_Read_Csd

Routines for file handling:

- Mmc_Fat_Init
- Mmc_Fat_QuickFormat
- Mmc_Fat_Assign
- Mmc_Fat_Reset
- Mmc_Fat_Read
- Mmc_Fat_Rewrite
- Mmc_Fat_Append
- Mmc_Fat_Delete
- Mmc_Fat_Write
- Mmc_Fat_Set_File_Date
- Mmc_Fat_Get_File_Date
- Mmc_Fat_Get_File_Date_Modified
- Mmc_Fat_Get_File_Size
- Mmc_Fat_Get_Swap_File

Mmc_Init

<table>
<tr><td>Prototype</td><td><code>sub function Mmc_Init() as word</code></td></tr>
<tr><td>Description</td><td>Initializes MMC through hardware SPI interface.

Mmc_Init needs to be called before using other functions of this library.</td></tr>
<tr><td>Parameters</td><td>None.</td></tr>
<tr><td>Returns</td><td>- <code>0</code> - if MMC/SD card was detected and successfully initialized
- <code>1</code> - otherwise</td></tr>
<tr><td>Requires</td><td>The appropriate hardware SPI module must be previously initialized.

Global variables:

- <code>Mmc_Chip_Select</code>: Chip Select line
- <code>Mmc_Chip_Select_Direction</code>: Direction of the Chip Select pin

must be defined before using this function.</td></tr>
<tr><td>Example</td><td><pre>
' MMC module connections
dim Mmc_Chip_Select as sbit at LATF0_bit   ' for writing to output pin always use latch (PIC18 family)
dim Mmc_Chip_Select_Direction as sbit at TRISF0_bit
' MMC module connections

dim error as byte
...
' Initialize the SPI module
SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_64, _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE)
' Loop until MMC is initialized
Mmc_Init()
' Reinitialize the SPI module at higher speed (change primary prescaler).
SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_4, _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE)
</pre></td></tr>
<tr><td>Notes</td><td>None.</td></tr>
</table>

Mmc_Read_Sector

Prototype	`sub function Mmc_Read_Sector(dim sector as longword, dim byref dbuff as byte[512]) as word`
Description	The function reads one sector (512 bytes) from MMC card.
Parameters	- `sector`: MMC/SD card sector to be read. - `dbuff`: buffer of minimum 512 bytes in length for data storage.
Returns	- `0` - if reading was successful - `1` - if an error occurred
Requires	MMC/SD card must be initialized. See Mmc_Init.

Example

```
' read sector 510 of the MMC/SD card
dim error as word
    sectorNo as longword
    dataBuffer as byte[512]
...
sectorNo = 510
error = Mmc_Read_Sector(sectorNo, dataBuffer)
```

Notes	None.

Mmc_Write_Sector

Prototype	`sub function Mmc_Write_Sector(dim sector as longword, dim byref data as byte[512]) as word`
Description	The function writes 512 bytes of data to one MMC card sector.
Parameters	- `sector`: MMC/SD card sector to be written to. - `dbuff`: data to be written (buffer of minimum 512 bytes in length).
Returns	- `0` - if writing was successful - `1` - if there was an error in sending write command - `2` - if there was an error in writing (data rejected)
Requires	MMC/SD card must be initialized. See Mmc_Init.

Example

```
' write to sector 510 of the MMC/SD card
dim error as word
    sectorNo as longword
    dataBuffer as byte[512]
...
sectorNo = 510
error = Mmc_Write_Sector(sectorNo, dataBuffer)
```

Notes	None.

Mmc_Read_Cid

Prototype	`sub function Mmc_Read_Cid(dim byref data_cid as byte[16]) as byte`
Description	The function reads 16-byte CID register.
Parameters	- `data_cid`: buffer of minimum 16 bytes in length for storing CID register content.
Returns	- `0` - if CID register was read successfully - `1` - if there was an error while reading
Requires	MMC/SD card must be initialized. See Mmc_Init.

Example

```
dim error as word
    dataBuffer as byte[512]
...
error = Mmc_Read_Cid(dataBuffer)
```

Notes	None.

Mmc_Read_Csd

Prototype	`sub function Mmc_Read_Csd(dim byref data_csd as byte[16]) as word`
Description	The function reads 16-byte CSD register.
Parameters	- `data_csd`: buffer of minimum 16 bytes in length for storing CSD register content.
Returns	- `0` - if CSD register was read successfully - `1` - if there was an error while reading
Requires	MMC/SD card must be initialized. See Mmc_Init.

Example

```
dim error as word
    dataBuffer as byte[512]
...
error = Mmc_Read_Csd(dataBuffer)
```

Notes	None.

Mmc_Fat_Init

Prototype	`sub function Mmc_Fat_Init() as word`
Description	Initializes MMC/SD card, reads MMC/SD FAT16 boot sector and extracts necessary data needed by the library.
Parameters	None.
Returns	- `0` - if MMC/SD card was detected and successfully initialized - `1` - if FAT16 boot sector was not found - `255` - if MMC/SD card was not detected
Requires	Global variables: - `Mmc_Chip_Select`: Chip Select line - `Mmc_Chip_Select_Direction`: Direction of the Chip Select pin must be defined before using this function. The appropriate hardware SPI module must be previously initialized. See the SPIx_Init, SPIx_Init_Advanced routines.

Example	`' MMC module connections`
	`dim Mmc_Chip_Select as sbit sfr at LATF0_bit`
	`dim Mmc_Chip_Select_Direction as sbit sfr at TRISF0_bit`
	`' MMC module connections`
	`...`
	`' Initialize the SPI module`
	`SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_64, _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE)`
	`' Initialize MMC/SD card and MMC_FAT16 library globals`
	`Mmc_Fat_Init()`
	`' Reinitialize the SPI module at higher speed (change primary prescaler).`
	`SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_4, _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE)`

Notes	MMC/SD card has to be formatted to FAT16 file system.

Mmc_Fat_QuickFormat

Prototype	`sub function Mmc_Fat_QuickFormat(dim byref mmc_fat_label as string[11]) as word`
Description	Formats to FAT16 and initializes MMC/SD card.
Parameters	- `mmc_fat_label`: volume label (11 characters in length). If less than 11 characters are provided, the label will be padded with spaces. If null string is passed volume will not be labeled
Returns	- `0` - if MMC/SD card was detected, successfully formated and initialized - `1` - if FAT16 format was unseccessful - `255` - if MMC/SD card was not detected
Requires	The appropriate hardware SPI module must be previously initialized.

Example	`// Initialize the SPI module`
	`SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_64, _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE);`
	`// Format and initialize MMC/SD card and MMC_FAT16 library globals`
	`Mmc_Fat_QuickFormat(“mikroE”);`
	`// Reinitialize the SPI module at higher speed (change primary prescaler).`
	`SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_4,_SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE);`

Notes	This routine can be used instead or in conjunction with Mmc_Fat_Init routine. If MMC/SD card already contains a valid boot sector, it will remain unchanged (except volume label field) and only FAT and ROOT tables will be erased. Also, the new volume label will be set.

Mmc_Fat_Assign

<table>
<tr><td>Prototype</td><td>sub function Mmc_Fat_Assign(dim byref filename as char[11], dim file_cre_attr as byte) as word</td></tr>
<tr><td>Description</td><td>Assigns file for file operations (read, write, delete...). All subsequent file operations will be applied on an assigned file.</td></tr>
<tr><td>Parameters</td><td>- filename: name of the file that should be assigned for file operations. File name should be in DOS 8.3 (file_name.extension) format. The file name and extension will be automatically padded with spaces by the library if they have less than length required (i.e. “mikro.tx” -> “mikro .tx “), so the user does no have to take care of that. The file name and extension are case insensitive. The library will convert them to proper case automatically, so the user does not have to take care of that.
Also, in order to keep backward compatibility with the first version of this library, file names can be entered as UPPERCASE string of 11 bytes in length with no dot character between file name and extension (i.e. “MIKROELETXT” -> MIKROELE.TXT). In this case last 3 characters of the string are considered to be file extension.

- file_cre_attr: file creation and attributes flags. Each bit corresponds to the appropriate file attribute:
<table>
<tr><th>Bit</th><th>Mask</th><th>Description</th></tr>
<tr><td>0</td><td>0x01</td><td>Read Only</td></tr>
<tr><td>1</td><td>0x02</td><td>Hidden</td></tr>
<tr><td>2</td><td>0x04</td><td>System</td></tr>
<tr><td>3</td><td>0x08</td><td>Volume Label</td></tr>
<tr><td>4</td><td>0x10</td><td>Subdirectory</td></tr>
<tr><td>5</td><td>0x20</td><td>Archive</td></tr>
<tr><td>6</td><td>0x40</td><td>Device (internal use only, never found on disk)</td></tr>
<tr><td>7</td><td>0x80</td><td>File creation flag. If file does not exist and this flag is set, a new file with specified name will be created.</td></tr>
</table>
</td></tr>
<tr><td>Returns</td><td>- 1 - if file already exists or file does not exist but a new file is created.
- 0 - if file does not exist and no new file is created.</td></tr>
<tr><td>Requires</td><td>MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init.</td></tr>
<tr><td>Example</td><td><i>' create file with archive attribut if it does not already exist</i>
Mmc_Fat_Assign(“MIKRO007.TXT”,0xA0)</td></tr>
<tr><td>Notes</td><td>Long File Names (LFN) are not supported.</td></tr>
</table>

Mmc_Fat_Reset

Prototype	`sub procedure Mmc_Fat_Reset(dim byref size as longword)`
Description	Procedure resets the file pointer (moves it to the start of the file) of the assigned file, so that the file can be read.
Parameters	- `size`: buffer to store file size to. After file has been opened for reading, its size is returned through this parameter.
Returns	Nothing.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.

Example	`dim size as longword`
	`...`
	`Mmc_Fat_Reset(size)`

Notes	None.

Mmc_Fat_Read

Prototype	`sub procedure Mmc_Fat_Read(dim byref bdata as byte)`
Description	Reads a byte from the currently assigned file opened for reading. Upon function execution file pointers will be set to the next character in the file.
Parameters	- `bdata`: buffer to store read byte to. Upon this function execution read byte is returned through this parameter.
Returns	Nothing.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign. The file must be opened for reading. See Mmc_Fat_Reset.

Example	`dim character as byte`
	`...`
	`Mmc_Fat_Read(character)`

Notes	None.

Mmc_Fat_Rewrite

Prototype	`sub procedure Mmc_Fat_Rewrite()`
Description	Opens the currently assigned file for writing. If the file is not empty its content will be erased.
Parameters	None.
Returns	Nothing.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.
Example	`' open file for writing` `Mmc_Fat_Rewrite()`
Notes	None.

Mmc_Fat_Append

Prototype	`sub procedure Mmc_Fat_Append()`
Description	Opens the currently assigned file for appending. Upon this function execution file pointers will be positioned after the last byte in the file, so any subsequent file write operation will start from there.
Parameters	None.
Returns	Nothing.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.
Example	`' open file for appending` `Mmc_Fat_Append()`
Notes	None.

Mmc_Fat_Delete

Prototype	`sub procedure Mmc_Fat_Delete()`
Description	Deletes currently assigned file from MMC/SD card.
Parameters	None.
Returns	Nothing.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.
Example	`' delete current file` `Mmc_Fat_Delete()`
Notes	None.

Mmc_Fat_Write

Prototype	`sub procedure Mmc_Fat_Write(dim byref fdata as byte[512], dim data_len as word)`
Description	Writes requested number of bytes to the currently assigned file opened for writing.
Parameters	- `fdata`: data to be written. - `data_len`: number of bytes to be written.
Returns	Nothing.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign. The file must be opened for writing. See Mmc_Fat_Rewrite or Mmc_Fat_Append.

Example	`dim file_contents as byte[42]`
	`...`
	`Mmc_Fat_Write(file_contents, 42)  ' write data to the assigned file`

Notes	None.

Mmc_Fat_Set_File_Date

Prototype	`sub procedure Mmc_Fat_Set_File_Date(dim year as word, dim month as byte, dim day as byte, dim hours as byte, dim mins as byte, dim seconds as byte)`
Description	Sets the date/time stamp. Any subsequent file write operation will write this stamp to the currently assigned file's time/date attributes.
Parameters	- `year`: year attribute. Valid values: 1980-2107 - `month`: month attribute. Valid values: 1-12 - `day`: day attribute. Valid values: 1-31 - `hours`: hours attribute. Valid values: 0-23 - `mins`: minutes attribute. Valid values: 0-59 - `seconds`: seconds attribute. Valid values: 0-59
Returns	Nothing.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign. The file must be opened for writing. See Mmc_Fat_Rewrite or Mmc_Fat_Append.
Example	`Mmc_Fat_Set_File_Date(2005,9,30,17,41,0)`
Notes	None.

Mmc_Fat_Get_File_Date

Prototype	`sub procedure Mmc_Fat_Get_File_Date(dim byref year as word, dim byref month as byte, dim byref day as byte, dim byref hours as byte, dim byref mins as byte)`
Description	Reads time/date attributes of the currently assigned file.
Parameters	- `year`: buffer to store year attribute to. Upon function execution year attribute is returned through this parameter. - `month`: buffer to store month attribute to. Upon function execution month attribute is returned through this parameter. - `day`: buffer to store day attribute to. Upon function execution day attribute is returned through this parameter. - `hours`: buffer to store hours attribute to. Upon function execution hours attribute is returned through this parameter. - `mins`: buffer to store minutes attribute to. Upon function execution minutes attribute is returned through this parameter.
Returns	Nothing.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.

Example

```
dim year as word
    month, day, hours, mins as byte
...
Mmc_Fat_Get_File_Date(year, month, day, hours, mins)
```

Notes	None.

Mmc_Fat_Get_File_Date_Modified

Prototype	`sub procedure Mmc_Fat_Get_File_Date_Modified(dim byref year as word, dim byref month as byte, dim byref day as byte, dim byref hours as byte, dim byref mins as byte)`
Description	Retrieves the last modification date/time for the currently selected file. Seconds are not being retrieved since they are written in 2-sec increments.
Parameters	- `year`: buffer to store year attribute to. Upon function execution year attribute is returned through this parameter. - `month`: buffer to store month attribute to. Upon function execution month attribute is returned through this parameter. - `day`: buffer to store day attribute to. Upon function execution day attribute is returned through this parameter. - `hours`: buffer to store hours attribute to. Upon function execution hours attribute is returned through this parameter. - `mins`: buffer to store minutes attribute to. Upon function execution minutes attribute is returned through this parameter.
Returns	Nothing.
Requires	The file must be assigned, see Mmc_Fat_Assign.

Example

```
dim year as word
    month, day, hours, mins as byte
...
Mmc_Fat_Get_File_Date_Modified(year, month, day, hours, mins)
```

Mmc_Fat_Get_File_Size

Prototype	`sub function Mmc_Fat_Get_File_Size() as longword`
Description	This function reads size of the currently assigned file in bytes.
Parameters	None.
Returns	This function returns size of active file (in bytes).
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init. The file must be previously assigned. See Mmc_Fat_Assign.

Example

```
dim my_file_size as longword
...
my_file_size = Mmc_Fat_Get_File_Size()
```

Notes	None

Mmc_Fat_Get_Swap_File

Prototype	`sub function Mmc_Fat_Get_Swap_File(dim sectors_cnt as longword, dim byref filename as string[11], dim file_attr as byte) as longword`
Description	This function is used to create a swap file of predefined name and size on the MMC/SD media. If a file with specified name already exists on the media, search for consecutive sectors will ignore sectors occupied by this file. Therefore, it is recommended to erase such file if it already exists before calling this function. If it is not erased and there is still enough space for a new swap file, this function will delete it after allocating new memory space for a new swap file. The purpose of the swap file is to make reading and writing to MMC/SD media as fast as possible, by using the Mmc_Read_Sector() and Mmc_Write_Sector() functions directly, without potentially damaging the FAT system. The swap file can be considered as a "window" on the media where the user can freely write/read data. It's main purpose in this library is to be used for fast data acquisition; when the time-critical acquisition has finished, the data can be re-written into a "normal" file, and formatted in the most suitable way.
Parameters	- `sectors_cnt`: number of consecutive sectors that user wants the swap file to have. - `filename`: name of the file that should be assigned for file operations. File name should be in DOS 8.3 (file_name.extension) format. The file name and extension will be automatically padded with spaces by the library if they have less than length required (i.e. "mikro.tx" -> "mikro .tx "), so the user does no have to take care of that. The file name and extension are case insensitive. The library will convert them to proper case automatically, so the user does not have to take care of that. Also, in order to keep backward compatibility with the first version of this library, file names can be entered as UPPERCASE string of 11 bytes in length with no dot character between file name and extension (i.e. "MIKROELETXT" -> MIKROELE.TXT). In this case last 3 characters of the string are considered to be file extension. - `file_attr`: file creation and attributes flags. Each bit corresponds to the appropriate file attribute: (see table below)
Returns	- Number of the start sector for the newly created swap file, if there was enough free space on the MMC/SD card to create file of required size. - `0` - otherwise.
Requires	MMC/SD card and MMC library must be initialized for file operations. See Mmc_Fat_Init.

Bit	Mask	Description
0	0x01	Read Only
1	0x02	Hidden
2	0x04	System
3	0x08	Volume Label
4	0x10	Subdirectory
5	0x20	Archive
6	0x40	Device (internal use only, never found on disk)
7	0x80	Not used

Example	see code below
Notes	Long File Names (LFN) are not supported.

Example

```
'-------------- Try to create a swap file with archive atribute, whose size will be at least 1000 sectors.
'               If it succeeds, it sends No. of start sector over UART
dim size as longword
...
size = Mmc_Fat_Get_Swap_File(1000, "mikroE.txt", 0x20)
if (size <> 0) then
  UART1_Write(0xAA)
  UART1_Write(Lo(size))
  UART1_Write(Hi(size))
  UART1_Write(Higher(size))
  UART1_Write(Highest(size))
  UART1_Write(0xAA)
end if
```

Library Example

This project consists of several blocks that demonstrate various aspects of usage of the Mmc_Fat16 library. These are:

- Creation of new file and writing down to it;
- Opening existing file and re-writing it (writing from start-of-file);
- Opening existing file and appending data to it (writing from end-of-file);
- Opening a file and reading data from it (sending it to UART terminal);
- Creating and modifying several files at once;
- Reading file contents;
- Deleting file(s);
- Creating the swap file (see Help for details);

Copy Code To Clipboard

```
program MMC_FAT_Test

dim
  Mmc_Chip_Select as sbit at LATF0_bit    ' for writing to output pin always use latch (PIC18 family)
  Mmc_Chip_Select_Direction as sbit at TRISF0_bit

const LINE_LEN = 43

dim
  err_txt as string[20]
  file_contents as string[LINE_LEN]

  filename as string[14]   ' File names

  character as byte
  loop1, loop2 as byte
  size as longint

  buffer as byte[512]
```

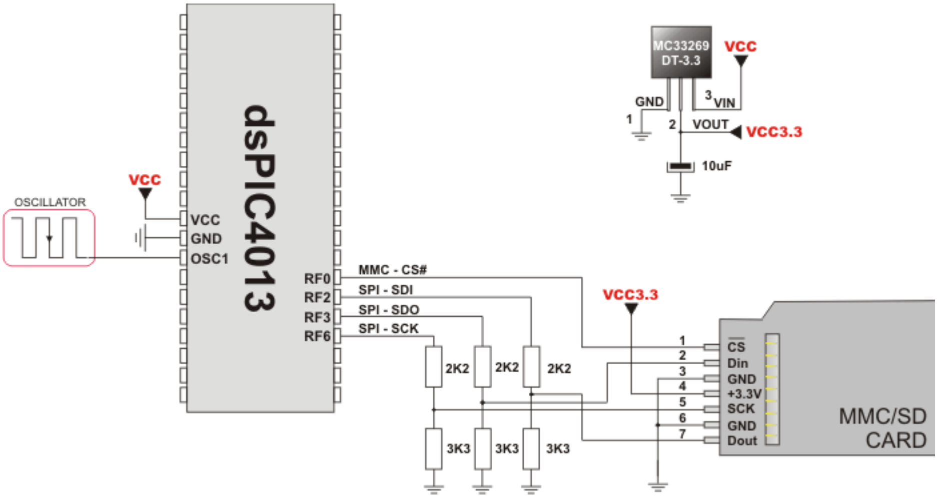
```
' UART write text and new line (carriage return + line feed)
sub procedure UART_Write_Line( dim byref uart_text as string )
  UART1_Write_Text(uart_text)
  UART1_Write(13)
  UART1_Write(10)
end sub

'-------------- Creates new file and writes some data to it
sub procedure M_Create_New_File()
  filename[7] = "A"                               ' Set filename for single-file tests
  Mmc_Fat_Set_File_Date(2005,6,21,10,35,0)        ' Set file date & time info
  Mmc_Fat_Assign(filename, 0xA0)                  ' Will not find file and then create file
  Mmc_Fat_Rewrite                                 ' To clear file and start with new data
  for loop1=1 to 99                               '  We want 5 files on the MMC card
    UART1_Write(".")
    file_contents[0] = loop1 div 10 + 48
    file_contents[1] = loop1 mod 10 + 48
    Mmc_Fat_Write(file_contents, LINE_LEN-1)      ' write data to the assigned file
  next loop1
end sub

'-------------- Creates many new files and writes data to them
sub procedure M_Create_Multiple_Files()
  for loop2 = "B" to "Z"
    UART1_Write(loop2)                           ' signal the progress
    filename[7] = loop2                          ' set filename
    Mmc_Fat_Set_File_Date(2005,6,21,10,35,0)     ' Set file date & time info
    Mmc_Fat_Assign(filename, 0xA0)               ' find existing file or create a new one
    Mmc_Fat_Rewrite                              ' To clear file and start with new data
    for loop1 = 1 to 44
      file_contents[0] = byte(loop1 div 10 + 48)
      file_contents[1] = byte(loop1 mod 10 + 48)
      Mmc_Fat_Write(file_contents, LINE_LEN-1)  ' write data to the assigned file
    next loop1
  next loop2
end sub

'-------------- Opens an existing file and rewrites it
sub procedure M_Open_File_Rewrite()
  filename[7] = "C"                    ' Set filename for single-file tests
  Mmc_Fat_Assign(filename, 0)
  Mmc_Fat_Rewrite
  for loop1 = 1 to 55
    file_contents[0] = byte(loop1 div 10 + 48)
    file_contents[1] = byte(loop1 mod 10 + 48)
    Mmc_Fat_Write(file_contents, 42)  ' write data to the assigned file
  next loop1
end sub

'-------------- Opens an existing file and appends data to it
'               (and alters the date/time stamp)
sub procedure M_Open_File_Append()
  filename[7] = "B"
  Mmc_Fat_Assign(filename, 0)
  Mmc_Fat_Set_File_Date(2009, 1, 23, 17, 22, 0)
  Mmc_Fat_Append()                                  ' Prepare file for append
  file_contents = " for mikroElektronika 2007"      ' Prepare file for append
```

```
  file_contents[26] = 10                           ' LF
  Mmc_Fat_Write(file_contents, 27)                 ' Write data to assigned file
end sub

'-------------- Opens an existing file, reads data from it and puts it to USART
sub procedure M_Open_File_Read()
  filename[7] = "B"
  Mmc_Fat_Assign(filename, 0)
  Mmc_Fat_Reset(size)                  ' To read file, procedure returns size of file
  while size > 0
    Mmc_Fat_Read(character)
    UART1_Write(character)             ' Write data to UART
    Dec(size)
  wend
end sub

'-------------- Deletes a file. If file doesn't exist, it will first be created
'               and then deleted.
sub procedure M_Delete_File()
  filename[7] = "F"
  Mmc_Fat_Assign(filename, 0)
  Mmc_Fat_Delete
end sub

'-------------- Tests whether file exists, and if so sends its creation date
'               and file size via USART
sub procedure M_Test_File_Exist
dim
  fsize as longint
  year as word
  month_, day, hour_, minute_ as byte
  outstr as char[12]

  filename[7] = "B"
  if Mmc_Fat_Assign(filename, 0) <> 0 then
    '--- file has been found - get its date
    Mmc_Fat_Get_File_Date(year,month_,day,hour_,minute_)
    UART1_Write_Text(" created: ")
    WordToStr(year, outstr)
    UART1_Write_Text(outstr)
    ByteToStr(month_, outstr)
    UART1_Write_Text(outstr)
    WordToStr(day, outstr)
    UART1_Write_Text(outstr)
    WordToStr(hour_, outstr)
    UART1_Write_Text(outstr)
    WordToStr(minute_, outstr)
    UART1_Write_Text(outstr)

   '--- file has been found - get its modified date
    Mmc_Fat_Get_File_Date_Modified(year, month_, day, hour_, minute_)
    UART1_Write_Text(" modified: ")
    WordToStr(year, outstr)
    UART1_Write_Text(outstr)
    ByteToStr(month_, outstr)
    UART1_Write_Text(outstr)
    WordToStr(day, outstr)
```

```
    UART1_Write_Text(outstr)
    WordToStr(hour_, outstr)
    UART1_Write_Text(outstr)
    WordToStr(minute_, outstr)
    UART1_Write_Text(outstr)

    '--- get file size
    fsize = Mmc_Fat_Get_File_Size
    LongIntToStr(fsize, outstr)
    UART_Write_Line(outstr)
  else
    '--- file was not found - signal it
    UART1_Write(0x55)
    Delay_ms(1000)
    UART1_Write(0x55)
  end if
end sub

'-------------- Tries to create a swap file, whose size will be at least 100
'                sectors (see Help for details)
sub procedure M_Create_Swap_File()
dim i as word

  for i=0 to 511
    Buffer[i] = i
  next i

  size = Mmc_Fat_Get_Swap_File(5000, "mikroE.txt", 0x20)    ' see help on this function
for details

  if (size <> 0) then
    LongIntToStr(size, err_txt)
    UART_Write_Line(err_txt)

    for i=0 to 4999
      Mmc_Write_Sector(size, Buffer)
      Inc(size)
      UART1_Write(".")
    next i
  end if
end sub

'-------------- Main. Uncomment the function(s) to test the desired operation(s)
main:
  err_txt = "FAT16 not found"
  file_contents = "XX MMC/SD FAT16 library by Anton Rieckert#"
  file_contents[41] = 10                        ' newline
  filename = "MIKRO00xTXT"

  #DEFINE COMPLETE_EXAMPLE      ' comment this line to make simpler/smaller example
  PORTD = 0
  TRISD = 0
  PORTF = 0
  TRISF = 0
  ADPCFG = 0xFFFF               ' initialize AN pins as digital
```

```
'--- set up USART for the file read
    SPI1_Init_Advanced(_SPI_MASTER,  _SPI_8_BIT,  _SPI_PRESCALE_SEC_1,  _SPI_PRESCALE_
PRI_64,
                    _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_
ACTIVE_2_IDLE)

  UART1_Init(19200)                       ' Initialize UART module at 9600 bps
  Delay_ms(100)                           ' Wait for UART module to stabilize

  U1MODE.ALTIO = 1   ' Switch Rx and Tx pins on their alternate locations.
                     ' This is used to free the pins for other module, namely the SPI.

  UART_Write_Line("dsPIC-Started")        ' dsPIC present report
  ' use fat16 quick format instead of init routine if a formatting is needed
  if Mmc_Fat_Init() = 0 then
    ' reinitialize spi at higher speed
    SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_4,
    _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE)
    '--- Test start
    UART_Write_Line("Test Start.")
    M_Create_New_File()
    #IFDEF COMPLETE_EXAMPLE
    M_Create_Multiple_Files()
    M_Open_File_Rewrite()
    M_Open_File_Append()
    M_Open_File_Read()
    M_Delete_File()
    M_Test_File_Exist()
    M_Create_Swap_File()
    #ENDIF
    UART_Write_Line("Test End.")
  else
    UART_Write_Line(err_txt)  ' Note: Cf_Fat_Init tries to initialize a card more than once.
      '      If card is not present, initialization may last longer (depending on clock speed)
  end if
end.
```

HW Connection



Pin diagram of MMC memory card

OneWire Library

The OneWire library provides routines for communication via the Dallas OneWire protocol, e.g. with DS18x20 digital thermometer. OneWire is a Master/Slave protocol, and all communication cabling required is a single wire. OneWire enabled devices should have open collector drivers (with single pull-up resistor) on the shared data line.

Slave devices on the OneWire bus can even get their power supply from data line. For detailed schematic see device datasheet.

Some basic characteristics of this protocol are:

- single master system,
- low cost,
- low transfer rates (up to 16 kbps),
- fairly long distances (up to 300 meters),
- small data transfer packages.

Each OneWire device also has a unique 64-bit registration number (8-bit device type, 48-bit serial number and 8-bit CRC), so multiple slaves can co-exist on the same bus.

Important:

- Oscillator frequency *Fosc* needs to be at least 4MHz in order to use the routines with Dallas digital thermometers.
- This library implements time-based activities, so interrupts need to be disabled when using OneWire library.

Library Routines

- Ow_Reset
- Ow_Read
- Ow_Write

Ow_Reset

Prototype	`sub function Ow_Reset(dim byref port as word, dim pin as word) as word`
Description	Issues OneWire reset signal for DS18x20.
Parameters	- `port`: OneWire bus port - `pin`: OneWire bus pin
Returns	- `0` if the device is present - `1` if the device is not present
Requires	Devices compliant with the Dallas OneWire protocol.
Example	`' Issue Reset signal on One-Wire Bus connected to pin RF6` `Ow_Reset(PORTF, 6)`
Notes	None.

Ow_Read

Prototype	`sub function Ow_Read(dim byref port as word, dim pin as word) as byte`
Description	Reads one byte of data via the OneWire bus.
Parameters	- `port`: OneWire bus port - `pin`: OneWire bus pin
Returns	Data read from an external device over the OneWire bus.
Requires	Devices compliant with the Dallas OneWire protocol.
Example	see code below
Notes	None.

```
' Read a byte from the One-Wire Bus connected to pin RF6
dim read_data as byte
...
read_data = Ow_Read(PORTF, 6)
```

Ow_Write

Prototype	`sub procedure Ow_Write(dim byref port as word, dim pin, data_ as word)`
Description	Writes one byte of data via the OneWire bus.
Parameters	- `port`: OneWire bus port - `pin`: OneWire bus pin - `data_`: data to be written
Returns	Nothing.
Requires	Devices compliant with the Dallas OneWire protocol.
Example	`' Send a byte to the One-Wire Bus connected to pin RF6` `Ow_Write(PORTF, 6, 0xCC)`
Notes	None.

Library Example

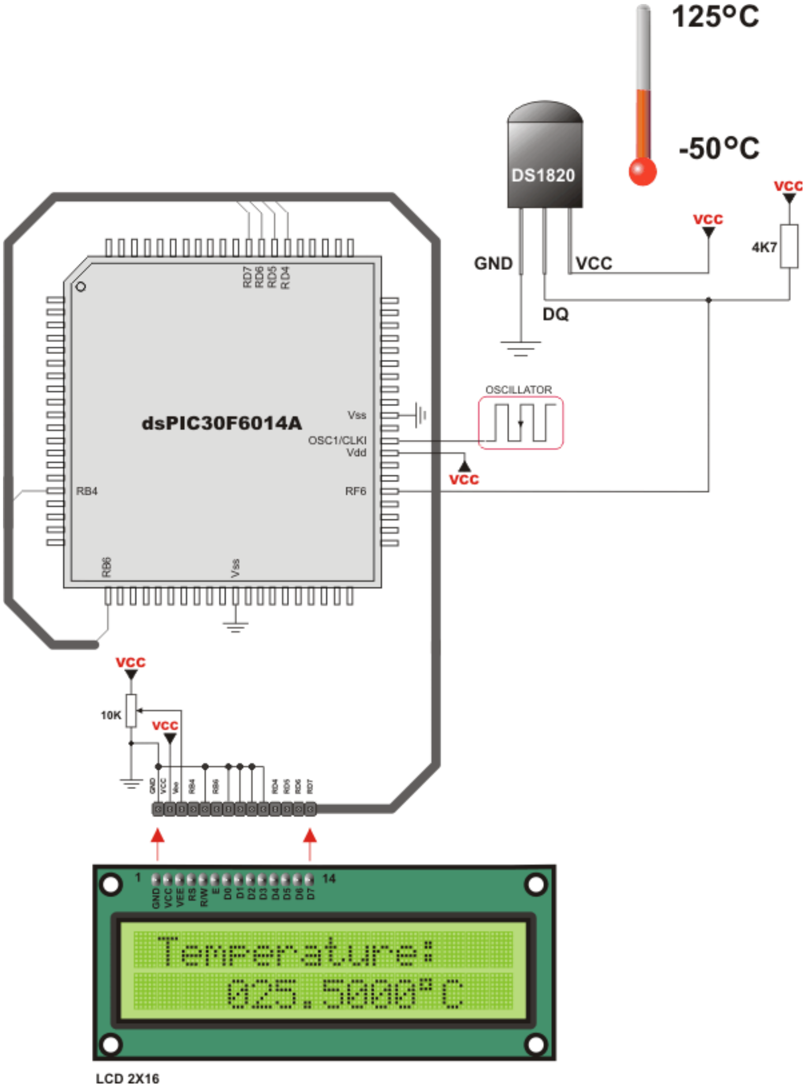
This example reads the temperature using DS18x20 connected to pin RF6. After reset, MCU obtains temperature from the sensor and prints it on the Lcd. Be sure to set Fosc appropriately in your project, to pull-up RF6 line and to turn off the PORTF leds.

Copy Code To Clipboard

```
program OneWire

' LCD module connections
dim LCD_RS as sbit at LATB4_bit
dim LCD_EN as sbit at LATB6_bit
dim LCD_D4 as sbit at LATD4_bit
dim LCD_D5 as sbit at LATD5_bit
dim LCD_D6 as sbit at LATD6_bit
dim LCD_D7 as sbit at LATD7_bit

dim LCD_RS_Direction as sbit at TRISB4_bit
dim LCD_EN_Direction as sbit at TRISB6_bit
dim LCD_D4_Direction as sbit at TRISD4_bit
dim LCD_D5_Direction as sbit at TRISD5_bit
dim LCD_D6_Direction as sbit at TRISD6_bit
dim LCD_D7_Direction as sbit at TRISD7_bit
' End LCD module connections

'  Set TEMP_RESOLUTION to the corresponding resolution of used DS18x20 sensor:
'  18S20: 9  (default settingcan be 9,10,11,or 12)
'  18B20: 12
const TEMP_RESOLUTION as byte = 9

dim text as char[9]
    temp as word

sub procedure Display_Temperature( dim temp2write as word )
const RES_SHIFT = TEMP_RESOLUTION - 8

dim temp_whole as byte
    temp_fraction as word

text = "000.0000"
  ' Check if temperature is negative
  if (temp2write and 0x8000) then
      text[0] = "-"
      temp2write = not temp2write + 1
  end if

  ' Extract temp_whole
  temp_whole = word(temp2write >> RES_SHIFT)

  ' Convert temp_whole to characters
  if ( temp_whole div 100 ) then
    text[0] = temp_whole div 100  + 48
  else
    text[0] = "0"
  end if
```

```
  text[1] = (temp_whole div 10)mod 10 + 48               ' Extract tens digit
  text[2] =  temp_whole mod 10        + 48               ' Extract ones digit

  ' Extract temp_fraction and convert it to unsigned int
  temp_fraction  =  word(temp2write << (4-RES_SHIFT))
  temp_fraction  = temp_fraction and 0x000F
  temp_fraction  = temp_fraction * 625

  ' Convert temp_fraction to characters
  text[4] = word(temp_fraction div 1000)     + 48        ' Extract thousands digit
  text[5] = word((temp_fraction div 100)mod 10 + 48)     ' Extract hundreds digit
  text[6] = word((temp_fraction div 10)mod 10  + 48)     ' Extract tens digit
  text[7] = word(temp_fraction mod 10)       + 48        ' Extract ones digit

  ' Print temperature on Lcd
  Lcd_Out(2, 5, text)
end sub

main:
  ADPCFG = 0                                     ' Configure AN pins as digital I/O

  text = "000.0000"
  Lcd_Init()                                     ' Initialize LCD
  Lcd_Cmd(_LCD_CLEAR)                            ' Clear LCD
  Lcd_Cmd(_LCD_CURSOR_OFF)                       ' Turn cursor off
  Lcd_Out(1, 1, " Temperature:   ")

  Lcd_Chr(2,13,178)      ' Print degree character, "C" for Centigrades
                         ' Different LCD displays have different char code for degree
  Lcd_Chr(2,14,"C")      ' If you see greek alpha letter try typing 178 instead of 223

  '--- Main loop
  while TRUE
    '--- Perform temperature reading
    Ow_Reset(PORTF, 6)                           ' Onewire reset signal
    Ow_Write(PORTF, 6, 0xCC)                     ' Issue command SKIP_ROM
    Ow_Write(PORTF, 6, 0x44)                     ' Issue command CONVERT_T
    Delay_us(120)

    Ow_Reset(PORTF, 6)
    Ow_Write(PORTF, 6, 0xCC)                     ' Issue command SKIP_ROM
    Ow_Write(PORTF, 6, 0xBE)                     ' Issue command READ_SCRATCHPAD

    temp =  Ow_Read(PORTF, 6)
    temp = (Ow_Read(PORTF, 6) << 8) + temp

    '--- Format and display result on Lcd

    Display_Temperature(temp)

    Delay_ms(520)
  wend
end.
```

HW Connection



Example of DS1820 connection

Peripheral Pin Select Library

The Peripheral Pin Select library enables user to have more than one digital peripheral multiplexed on a single pin. Users may independently map the input and/or output of any one of many digital peripherals to any one of these I/O pins.
The peripherals managed by the Peripheral Pin Select library are all digital only peripherals.

A key difference between pin select and non pin select peripherals is that pin select peripherals are not associated with a default I/O pin. The peripheral must always be assigned to a specific I/O pin before it can be used.
In contrast, non pin select peripherals are always available on a default pin, assuming that the peripheral is active and not conflicting with another peripheral.

When a pin selectable peripheral is active on a given I/O pin, it takes priority over all other digital I/O and digital communication peripherals associated with the pin.

Important: Before using any of the digital peripherals or its library routines, user must set the desired pins as input/output and assign the desired peripheral to these pins.

Library Routines

- Unlock_IOLOCK
- Lock_IOLOCK
- PPS_Mapping

Unlock_IOLOCK

Prototype	`sub procedure Unlock_IOLOCK()`
Description	Unlocks I/O pins for Peripheral Pin Mapping.
Parameters	None.
Returns	Nothing.
Requires	Nothing.
Example	`Unlock_IOLOCK()`
Notes	None.

Lock_IOLOCK

Prototype	`sub procedure Lock_IOLOCK()`
Description	Locks I/O pins for Peripheral Pin Mapping.
Parameters	None.
Returns	Nothing.
Requires	Nothing.
Example	`Lock_IOLOCK()`

PPS_Mapping

Prototype	`sub function PPS_Mapping(dim rp_num, input_output, funct_name as byte)`
Description	Sets desired internal MCU module to be mapped on the requested pins.
Parameters	- `rp_num`: Remappable pin number. Consult the appropriate datasheet for adequate values. - `direction`: Sets requested pin to be used as an input or output. See Direction Parameters for adequate values. - `funct_name`: Selects internal MCU module function for usage. See Input Functions or Output Functions for adequate values.
Returns	- `0` - if non-existing peripheral pin is selected. - `1` - if desired function is not implemented for the chosen MCU. - `2` - if any of the other RPOUT registers is configured to output the SCK1OUT function while SCK1CM is set (only for P24FJ256GA110 Family). - `255` - if peripheral pin mapping was successful.
Requires	Nothing.
Example	`PPS_Mapping(15, _INPUT, _RX2_DT2)     ' Sets pin 15 to be Input, and maps RX2/DT2 Input to it` `PPS_Mapping(5, _OUTPUT, _TX2_CK2)     ' Sets pin 5 to be Output, and maps EUSART2 Asynchronous Transmit/Synchronous Clock Output to it`
Notes	None.

Direction Parameters

Direction Parameter	Description
_INPUT	Sets selected pin as input
_OUTPUT	Sets selected pin as output

Input Functions

Function Name	Description
_CIRX	ECAN1 Receive
_COFSI	DCI Frame Sync Input
_CSCKI	DCI Serial Clock Input
_CSDI	DCI Serial Data Input
_FLTA1	PWM1 Fault
_FLTA2	PWM2 Fault
_FLTA3	PWM3 Fault
_FLTA4	PWM4 Fault
_FLTA5	PWM5 Fault
_FLTA6	PWM6 Fault
_FLTA7	PWM7 Fault
_FLTA8	PWM8 Fault
_IC1	Input Capture 1
_IC2	Input Capture 2
_IC3	Input Capture 3
_IC4	Input Capture 4
_IC5	Input Capture 5
_IC6	Input Capture 6
_IC7	Input Capture 7
_IC8	Input Capture 8
_IC9	Input Capture 9
_INDX1	QEI1 Index
_INDX2	QEI2 Index
_INT1	External Interrupt 1
_INT2	External Interrupt 2
_INT3	External Interrupt 3
_INT4	External Interrupt 4

_QEA1	QEI1 Phase A
_QEA2	QEI2 Phase A
_QEB1	QEI1 Phase B
_QEB2	QEI2 Phase B
_SCK1IN	SPI1 Clock Input
_SCK2IN	SPI2 Clock Input
_SCK3IN	SPI3 Clock Input
_SDI1	SPI1 Data Input
_SDI2	SPI2 Data Input
_SDI3	SPI3 Data Input
_SS1IN	SPI1 Slave Select Input
_SS2IN	SPI2 Slave Select Input
_SS3IN	SPI3 Slave Select Input

_T1CK	Timer1 External Clock
_T2CK	Timer2 External Clock
_T3CK	Timer3 External Clock
_T4CK	Timer4 External Clock
_T5CK	Timer5 External Clock
_U1CTS	UART1 Clear To Send
_U2CTS	UART2 Clear To Send
_U3CTS	UART3 Clear To Send
_U4CTS	UART4 Clear To Send
_U1RX	UART1 Receive
_U2RX	UART2 Receive
_U3RX	UART3 Receive
_U4RX	UART4 Receive

Output Functions

Function Name	**Description**
_NULL	The NULL function is assigned to all RPn outputs at device Reset and disables the RPn output function.
_ACMP1	RPn tied to Analog Comparator Output 1
_ACMP2	RPn tied to Analog Comparator Output 2
_ACMP3	RPn tied to Analog Comparator Output 3
_ACMP4	RPn tied to Analog Comparator Output 4
_C1OUT	Comparator 1 Output
_C2OUT	Comparator 2 Output
_C3OUT	Comparator 3 Output
_COFSOS	DCI Frame Sync Output
_CSCKO	DCI Serial Clock Output
_CSDO	DCI Serial Data Output
_CTPLS	CTMU Output Pulse
_C1TX	ECAN1 Transmit
_OC1	Output Compare 1
_OC2	Output Compare 2
_OC3	Output Compare 3
_OC4	Output Compare 4
_OC5	Output Compare 5
_OC6	Output Compare 6
_OC7	Output Compare 7
_OC8	Output Compare 8

_OC9	Output Compare 9
_OCFA	Output Compare Fault A
_OCFB	Output Compare Fault B
_PWM4H	RPn tied to PWM output pins associated with PWM Generator 4
_PWM4L	RPn tied to PWM output pins associated with PWM Generator 4
_REFCLKO	REFCLK output signal
_SCK1OUT	SPI1 Clock Output
_SCK2OUT	SPI2 Clock Output
_SCK3OUT	SPI3 Clock Output
_SDO1	SPI1 Data Output
_SDO2	SPI2 Data Output
_SDO3	SPI3 Data Output
_SS1OUT	SPI1 Slave Select Output
_SS2OUT	SPI2 Slave Select Output
_SS3OUT	SPI3 Slave Select Output
_SYNCI1	External Synchronization signal to PWM Master Time Base
_SYNCI2	External Synchronization signal to PWM Master Time Base
_SYNCO1	RPn tied to external device synchronization signal via PWM master time base
_U1RTS	UART1 Request To Send
_U2RTS	UART2 Request To Send
_U3RTS	UART3 Request To Send
_U4RTS	UART4 Request To Send
_U1TX	UART1 Transmit
_U2TX	UART2 Transmit
_U3TX	UART3 Transmit
_U4TX	UART4 Transmit
_UPDN	QEI direction (UPDN) status
_UPDN1	QEI1 direction (UPDN) status
_UPDN2	QEI2 direction (UPDN) status

Port Expander Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for communication with the Microchip's Port Expander MCP23S17 via SPI interface. Connections of the dsPIC30/33 and PIC24 MCU and MCP23S17 is given on the schematic at the bottom of this page.

Important:

- The library uses the SPI module for communication. User must initialize the appropriate SPI module before using the Port Expander Library.
- For MCUs with multiple SPI modules it is possible to initialize all of them and then switch by using the `SPI_Set_Active()` function. See the SPI Library functions.
- Library does not use Port Expander interrupts.

Library Dependency Tree

External dependencies of Port Expander Library

The following variables must be defined in all projects using Port Expander Library:	Description:	Example:
`dim SPExpanderRST as sbit sfr external`	Reset line.	`dim SPExpanderRST as sbit at LATF0_bit`
`dim SPExpanderCS as sbit sfr external`	Chip Select line.	`dim SPExpanderCS as sbit at LATF1_bit`
`dim SPExpanderRST_Direction as sbit sfr external`	Direction of the Reset pin.	`dim SPExpanderRST_Direction as sbit at TRISF0_bit`
`dim SPExpanderCS_Direction as sbit sfr external`	Direction of the Chip Select pin.	`dim SPExpanderCS_Directions as sbit at TRISF1_bit`

Library Routines

- Expander_Init
- Expander_Init_Advanced
- Expander_Read_Byte
- Expander_Write_Byte
- Expander_Read_PortA
- Expander_Read_PortB
- Expander_Read_PortAB
- Expander_Write_PortA
- Expander_Write_PortB
- Expander_Write_PortAB
- Expander_Set_DirectionPortA
- Expander_Set_DirectionPortB
- Expander_Set_DirectionPortAB
- Expander_Set_PullUpsPortA
- Expander_Set_PullUpsPortB
- Expander_Set_PullUpsPortAB

Expander_Init

Prototype	`sub procedure Expander_Init(dim ModuleAddress as byte)`
Description	Initializes Port Expander using SPI communication. Port Expander module settings: - hardware addressing enabled - automatic address pointer incrementing disabled (byte mode) - BANK_0 register adressing - slew rate enabled
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page
Returns	Nothing.
Requires	Global variables: - `SPExpanderCS`: Chip Select line - `SPExpanderRST`: Reset line - `SPExpanderCS_Direction`: Direction of the Chip Select pin - `SPExpanderRST_Direction`: Direction of the Reset pin must be defined before using this function. SPI module needs to be initialized. See SPIx_Init and SPIx_Init_Advanced routines.

Example

```
' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
    SPExpanderCS  as sbit at LATF1_bit
    SPExpanderRST_Direction as sbit at TRISF0_bit
    SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections

...
SPI1_Init()        ' initialize SPI module
Expander_Init(0)   ' initialize Port Expander
```

Notes	None.

Expander_Init_Advanced

Prototype	`sub procedure Expander_Init_Advanced(dim byref rstPort, rstPin, haen as byte)p>`
Description	Initializes Port Expander using SPI communication.
Parameters	- `rstPort`: Port Expander's reset port - `rstPin`: Port Expander's reset pin - `haen`: Port Expander's hardware address
Returns	Nothing.
Requires	- `SPExpanderCS`: Chip Select line - `SPExpanderRST`: Reset line - `SPExpanderCS_Direction`: Direction of the Chip Select pin - `SPExpanderRST_Direction`: Direction of the Reset pin must be defined before using this function. SPI module needs to be initialized. See SPIx_Init and SPIx_Init_Advanced routines.

Example

```
' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
    SPExpanderCS  as sbit at LATF1_bit
    SPExpanderRST_Direction as sbit at TRISF0_bit
    SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections

...
' If Port Expander Library uses SPI1 module
SPI1_Init()                    ' initialize SPI module
Expander_Init_Advanced(PORTB, 0, 0)  ' initialize Port Expander
```

Notes	None.

Expander_Read_Byte

Prototype	`sub function Expander_Read_Byte(dim ModuleAddress, RegAddress as byte) as byte`
Description	The function reads byte from Port Expander.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `RegAddress`: Port Expander's internal register address
Returns	Byte read.
Requires	Port Expander must be initialized. See Expander_Init.

Example

```
' Read a byte from Port Expander's register
dim read_data as byte
...
read_data = Expander_Read_Byte(0,1)
```

Notes	None.

Expander_Write_Byte

Prototype	`sub procedure Expander_Write_Byte(dim ModuleAddress, RegAddress, Data as byte)`
Description	Routine writes a byte to Port Expander.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `RegAddress`: Port Expander's internal register address - `Data`: data to be written
Returns	Byte read.
Requires	Port Expander must be initialized. See Expander_Init.
Example	`' Write a byte to the Port Expander's register` `Expander_Write_Byte(0,1,$FF)`
Notes	None.

Expander_Read_PortA

Prototype	`sub function Expander_Read_PortA(dim ModuleAddress as byte) as byte`
Description	The function reads byte from Port Expander's PortA.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page
Returns	Byte read.
Requires	Port Expander must be initialized. See Expander_Init. Port Expander's PortA should be configured as input. See Expander_Set_DirectionPortA and Expander_Set_DirectionPortAB routines.

Example	`' Read a byte from Port Expander's PORTA`
	`dim read_data as byte`
	`...`
	`Expander_Set_DirectionPortA(0,$FF)    ' set expander's porta to be input`
	`...`
	`read_data = Expander_Read_PortA(0)`

Notes	None.

Expander_Read_PortB

Prototype	`sub function Expander_Read_PortB(dim ModuleAddress as byte) as byte`
Description	The function reads byte from Port Expander's PortB.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page
Returns	Byte read.
Requires	Port Expander must be initialized. See Expander_Init. Port Expander's PortB should be configured as input. See Expander_Set_DirectionPortB and Expander_Set_DirectionPortAB routines.

Example	`' Read a byte from Port Expander's PORTB`
	`dim read_data as byte`
	`...`
	`Expander_Set_DirectionPortB(0,$FF)      ' set expander's portb to be input`
	`...`
	`read_data = Expander_Read_PortB(0)`

Notes	None.

Expander_Read_PortAB

Prototype	`sub function Expander_Read_PortAB(dim ModuleAddress as byte) as word`
Description	The function reads word from Port Expander's ports. PortA readings are in the higher byte of the result. PortB readings are in the lower byte of the result.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page
Returns	Word read.
Requires	Port Expander must be initialized. See Expander_Init. Port Expander's PortA and PortB should be configured as inputs. See Expander_Set_DirectionPortA, Expander_Set_DirectionPortB and Expander_Set_DirectionPortAB routines.

Example	`' Read a byte from Port Expander's PORTA and PORTB`
	`dim read_data as word`
	`...`
	`Expander_Set_DirectionPortAB(0,$FFFF)      ' set expander's porta and portb to be input`
	`...`
	`read_data s= Expander_Read_PortAB(0)`

Notes	None.

Expander_Write_PortA

Prototype	`sub procedure Expander_Write_PortA(dim ModuleAddress, Data as byte)`
Description	The function writes byte to Port Expander's PortA.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `Data`: data to be written
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init. Port Expander's PortA should be configured as output. See Expander_Set_DirectionPortA and Expander_Set_DirectionPortAB routines.

Example

```
' Write a byte to Port Expander's PORTA

...
Expander_Set_DirectionPortA(0,$00)     ' set expander's porta to be output
...
Expander_Write_PortA(0, $AA)
```

Notes	None.

Expander_Write_PortB

Prototype	`sub procedure Expander_Write_PortB(dim ModuleAddress, Data as byte)`
Description	The function writes byte to Port Expander's PortB.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `Data`: data to be written
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init. Port Expander's PortB should be configured as output. See Expander_Set_DirectionPortB and Expander_Set_DirectionPortAB routines.

Example

```
' Write a byte to Port Expander's PORTB

...
Expander_Set_DirectionPortB(0,$00)    ' set expander's portb to be output
...
Expander_Write_PortB(0,$55)
```

Notes	None.

Expander_Write_PortAB

Prototype	`sub procedure Expander_Write_PortAB(dim ModuleAddress as byte, dim Data as word)`
Description	The function writes word to Port Expander's ports.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `Data`: data to be written. Data to be written to PortA are passed in `Data`'s higher byte. Data to be written to PortB are passed in `Data`'s lower byte
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init. Port Expander's PortA and PortB should be configured as outputs. See Expander_Set_DirectionPortA, Expander_Set_DirectionPortB and Expander_Set_DirectionPortAB routines.

Example	`' Write a byte to Port Expander's PORTA and PORTB`
	`...`
	`Expander_Set_DirectionPortAB(0, $0000)     ' set expander's porta and portb to be output`
	`...`
	`Expander_Write_PortAB(0, $AA55)`

Notes	None.

Expander_Set_DirectionPortA

Prototype	`sub procedure Expander_Set_DirectionPortA(dim ModuleAddress, Data as byte)`
Description	The function sets Port Expander's PortA direction.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `Data`: data to be written to the PortA direction register. Each bit corresponds to the appropriate pin of the PortA register. Set bit designates corresponding pin as input. Cleared bit designates corresponding pin as output.
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init.
Example	`' Set Port Expander's PORTA to be output` `Expander_Set_DirectionPortA(0,$00)`
Notes	None.

Expander_Set_DirectionPortB

Prototype	`sub procedure Expander_Set_DirectionPortB(dim ModuleAddress, Data as byte)`
Description	The function sets Port Expander's PortB direction.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `Data`: data to be written to the PortB direction register. Each bit corresponds to the appropriate pin of the PortB register. Set bit designates corresponding pin as input. Cleared bit designates corresponding pin as output.
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init.
Example	`' Set Port Expander's PORTB to be input` `Expander_Set_DirectionPortB(0,$FF)`
Notes	None.

Expander_Set_DirectionPortAB

Prototype	`sub procedure Expander_Set_DirectionPortAB(dim ModuleAddress, Direction as word)`
Description	The function sets Port Expander's PortA and PortB direction.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `Direction`: data to be written to direction registers. Data to be written to the PortA direction register are passed in `Direction`'s higher byte. Data to be written to the PortB direction register are passed in `Direction`'s lower byte. Each bit corresponds to the appropriate pin of the PortA/PortB register. Set bit designates corresponding pin as input. Cleared bit designates corresponding pin as output.
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init.
Example	`' Set Port Expander's PORTA to be output and PORTB to be input` `Expander_Set_DirectionPortAB(0,$00FF)`
Notes	None.

Expander_Set_PullUpsPortA

Prototype	`sub procedure Expander_Set_PullUpsPortA(dim ModuleAddress, Data as byte)`
Description	The function sets Port Expander's PortA pull up/down resistors.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `Data`: data for choosing pull up/down resistors configuration. Each bit corresponds to the appropriate pin of the PortA register. Set bit enables pull-up for corresponding pin.
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init.
Example	`' Set Port Expander's PORTA pull-up resistors` `Expander_Set_PullUpsPortA(0, $FF)`
Notes	None.

Expander_Set_PullUpsPortB

Prototype	`sub procedure Expander_Set_PullUpsPortB(dim ModuleAddress, Data as byte)`
Description	The function sets Port Expander's PortB pull up/down resistors.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `Data`: data for choosing pull up/down resistors configuration. Each bit corresponds to the appropriate pin of the PortB register. Set bit enables pull-up for corresponding pin.
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init.
Example	`' Set Port Expander's PORTB pull-up resistors` `Expander_Set_PullUpsPortB(0, 0xFF)`
Notes	None.

Expander_Set_PullUpsPortAB

Prototype	`sub procedure Expander_Set_PullUpsPortAB(dim ModuleAddress as byte, dim PullUps as word)`
Description	The function sets Port Expander's PortA and PortB pull up/down resistors.
Parameters	- `ModuleAddress`: Port Expander hardware address, see schematic at the bottom of this page - `PullUps`: data for choosing pull up/down resistors configuration. PortA pull up/down resistors configuration is passed in `PullUps`' higher byte. PortB pull up/down resistors configuration is passed in `PullUps`' lower byte. Each bit corresponds to the appropriate pin of the PortA/PortB register. Set bit enables pull-up for corresponding pin.
Returns	Nothing.
Requires	Port Expander must be initialized. See Expander_Init.
Example	`' Set Port Expander's PORTA and PORTB pull-up resistors` `Expander_Set_PullUpsPortAB(0, $FFFF)`
Notes	None.

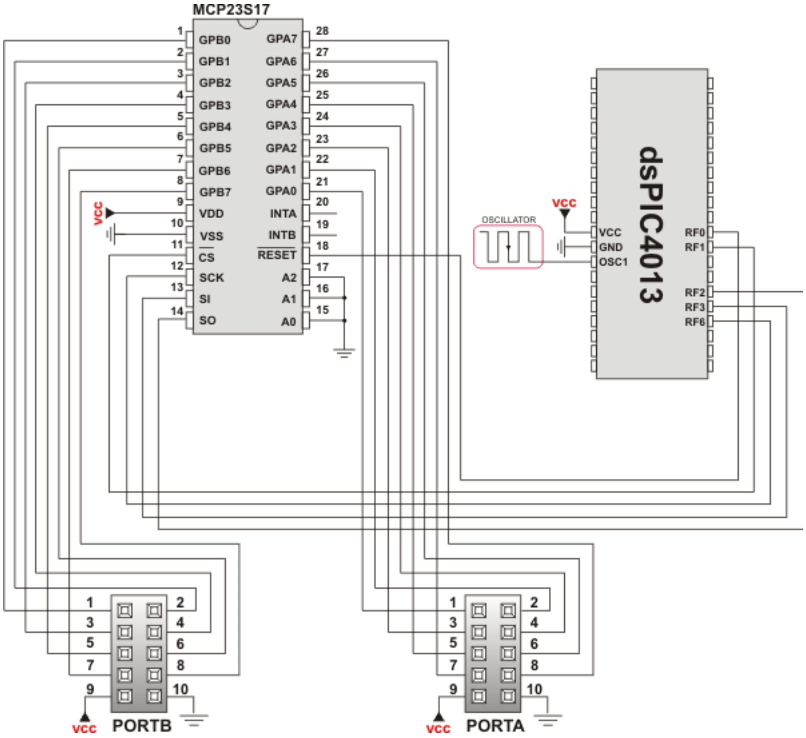
Library Example

The example demonstrates how to communicate with Port Expander MCP23S17. Note that Port Expander pins A2 A1 A0 are connected to GND so Port Expander Hardware Address is `0`.

Copy Code To Clipboard

```
program PortExpander

' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
    SPExpanderCS  as sbit at LATF1_bit
    SPExpanderRST_Direction as sbit at TRISF0_bit
    SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections

dim counter as word

main:
  ADPCFG = 0xFFFF                       ' initialize AN pins as digital

  TRISB  = 0x00
  LATB   = 0xFF

  ' If Port Expander Library uses SPI1 module
  SPI1_Init()                         ' Initialize SPI module used with PortExpander

  Expander_Init(0)                    ' Initialize Port Expander

  Expander_Set_DirectionPortA(0, 0x00)   ' Set Expander's PORTA to be output

  Expander_Set_DirectionPortB(0,0xFF)    ' Set Expander's PORTB to be input
  Expander_Set_PullUpsPortB(0,0xFF)      ' Set pull-ups to all of the Expander's PORTB
pins

  while ( TRUE )                         ' Endless loop
    Expander_Write_PortA(0, counter)     ' Write i to expander's PORTA
    Inc(counter)
    PORTB = Expander_Read_PortB(0)       ' Read expander's PORTB and write it to LEDs
    Delay_ms(100)
  wend

end.
```

HW Connection



Port Expander HW connection

PS/2 Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for communication with the common PS/2 keyboard.

Important:

- The library does not utilize interrupts for data retrieval, and requires the oscillator clock to be at least 6MHz.
- The pins to which a PS/2 keyboard is attached should be connected to the pull-up resistors.
- Although PS/2 is a two-way communication bus, this library does not provide MCU-to-keyboard communication; e.g. pressing the Caps Lock key will not turn on the Caps Lock LED.

External dependencies of PS/2 Library

The following variables must be defined in all projects using PS/2 Library:	**Description:**	**Example:**
`dim PS2_Data as sbit sfr external`	PS/2 Data line.	`dim PS2_Data as sbit at RB0_bit`
`dim PS2_Clock as sbit sfr external`	PS/2 Clock line.	`dim PS2_Clock as sbit at RB1_bit`
`dim PS2_Data_Direction as sbit sfr external`	Direction of the PS/2 Data pin.	`dim PS2_Data_Direction as sbit at TRISB0_bit`
`dim PS2_Clock_Direction as sbit sfr external`	Direction of the PS/2 Clock pin.	`dim PS2_Clock_Direction as sbit at TRISB1_bit`

Library Routines

- Ps2_Config
- Ps2_Key_Read

Ps2_Config

Prototype	`sub procedure Ps2_Config()`
Description	Initializes the MCU for work with the PS/2 keyboard.
Parameters	None.
Returns	Nothing.
Requires	Global variables: - `PS2_Data`: Data signal line - `PS2_Clock`: Clock signal line - `PS2_Data_Direction`: Direction of the Data pin - `PS2_Clock_Direction`: Direction of the Clock pin must be defined before using this function.

Example

```
' PS2 pinout definition
dim PS2_Data as sbit at RB0_bit
dim PS2_Clock as sbit at RB1_bit
dim PS2_Data_Direction as sbit at TRISB0_bit
dim PS2_Clock_Direction as sbit at TRISB1_bit
' End of PS2 pinout definition

...
Ps2_Config()          ' Init PS/2 Keyboard
```

Notes	None.

Ps2_Key_Read

Prototype	`sub function Ps2_Key_Read(dim byref value as byte, dim byref special as byte, dim byref pressed as byte) as word`
Description	The function retrieves information on key pressed.
Parameters	- `value`: holds the value of the key pressed. For characters, numerals, punctuation marks, and space `value` will store the appropriate ASCII code. Routine "recognizes" the function of Shift and Caps Lock, and behaves appropriately. For special function keys see Special Function Keys Table. - `special`: is a flag for special function keys (F1, Enter, Esc, etc). If key pressed is one of these, `special` will be set to 1, otherwise 0. - `pressed`: is set to 1 if the key is pressed, and 0 if it is released.
Returns	- `1` if reading of a key from the keyboard was successful - `0` if no key was pressed
Requires	PS/2 keyboard needs to be initialized. See Ps2_Config routine.

Example

```
dim value, special, pressed as word
...
' Press Enter to continue:
do {
  if (Ps2_Key_Read(value, special, pressed)) then
    if ((value = 13) and (special = 1)) then
      break
    end if
  end if
loop until (0=1)
```

Notes	None.

Special Function Keys

Key	Value returned
F1	1
F2	2
F3	3
F4	4
F5	5
F6	6
F7	7
F8	8
F9	9
F10	10
F11	11
F12	12
Enter	13
Page Up	14
Page Down	15
Backspace	16
Insert	17
Delete	18
Windows	19
Ctrl	20
Shift	21
Alt	22
Print Screen	23
Pause	24
Caps Lock	25
End	26
Home	27
Scroll Lock	28
Num Lock	29
Left Arrow	30
Right Arrow	31
Up Arrow	32
Down Arrow	33
Escape	34
Tab	35

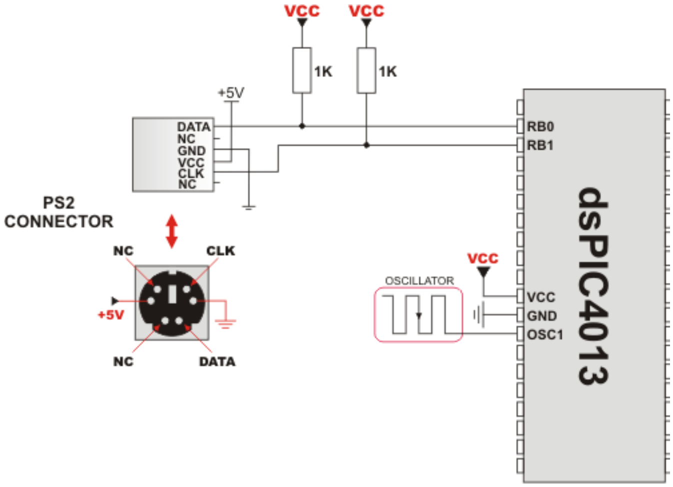
Library Example

This simple example reads values of the pressed keys on the PS/2 keyboard and sends them via UART.

Copy Code To Clipboard

```
program PS2_Example

dim keydata, special, down as byte

dim PS2_Data            as sbit at RB0_bit
    PS2_Clock           as sbit at RB1_bit
    PS2_Data_Direction  as sbit at TRISB0_bit
    PS2_Clock_Direction as sbit at TRISB1_bit

main:
  ADPCFG = 0xFFFF            ' Configure AN pins as digital I/O

  UART1_Init(19200)          ' Initialize UART module at 19200 bps


  Ps2_Config()               ' Init PS/2 Keyboard
  Delay_ms(100)               ' Wait for keyboard to finish
  UART1_Write_Text(“Ready”)    ' Ready
  UART1_Write(13)             ' Line Feed
  UART1_Write(10)             ' Carriage return


  while TRUE                                            ' Endless loop
    if Ps2_Key_Read(keydata, special, down) then      ' If data was read from PS/2
      if (down <> 0) and (keydata = 16) then          ' Backspace read
        UART1_Write(0x08)                        ' Send Backspace to usart terminal
      else
        if (down <> 0) and (keydata = 13) then        ' Enter read
          UART1_Write(10)                   ' Send carriage return to usart terminal
          UART1_Write(13) ' Uncomment this line if usart terminal also expects line feed
                                                      '   for new line transition
        else
          if (down <> 0) and (special = 0) and (keydata <> 0) then   ' Common key read
            UART1_Write(keydata)                      ' Send key to usart terminal
          end if
        end if
      end if
    end if
    Delay_ms(10)                                      ' Debounce period
  wend
end.
```

HW Connection



Example of PS2 keyboard connection

PWM Library

The CCP module is available with a number of dsPIC30/33 and PIC24 MCUs. mikroBasic PRO for dsPIC30/33 and PIC24 provides a library which simplifies using of the PWM HW Module.

Important: PWM module uses either Timer2 or Timer3 module.

Library Routines

- PWM_Init
- PWM_Set_Duty
- PWM_Start
- PWM_Stop

PWM_Init

Prototype	`sub function PWM_Init(dim freq_hz as longint, dim enable_channel_x, timer_prescale, use_timer_x as word) as word`
	
	`' 30F1010 and dsPIC33FJ06GS101/102/202 prototype`
	`sub function PWM_Init(dim freq_hz as longint, dim enable_channel_x, timer_prescale) as word`

Description	Initializes the PWM module with duty ratio 0.
Parameters	- `freq_hz`: PWM frequency in Hz (refer to device datasheet for correct values in respect with Fosc) - `enable_channel_x`: number of PWM channel to be initialized. Refer to MCU's datasheet for available PWM channels - `timer_prescale`: timer prescaler parameter. Valid values: `1`, `8`, `64`, and `256` - `use_timer_x`: timer to be used with the PWM module. Valid values: `2` (Timer2) and `3` (Timer3)
Returns	- `0xFFFF` - if timer settings are not valid - otherwise returns calculated timer period
Requires	MCU must have the HW PWM Module.

Example	`' Initializes the PWM module at 5KHz, channel 1, no clock prescale, timer2`
	`:`
	`dim pwm_period1 as word`
	`...`
	`pwm_period1 = PWM_Init(5000, 1, 0, 2)`

Notes	Number of available PWM channels depends on MCU. Refer to MCU datasheet for details.

PWM_Set_Duty

Prototype	`sub procedure PWM_Set_Duty(dim duty, channel as word)`
Description	The function changes PWM duty ratio.
Parameters	- `duty`: PWM duty ratio. Valid values: `0` to timer period returned by the PWM_Init function. - `channel`: number of PWM channel to change duty to.
Returns	Nothing.
Requires	MCU must have the HW PWM Module. PWM channel must be properly initialized. See PWM_Init routine.

Example	`' Set channel 1 duty ratio to 50%:`
	`dim pwm_period1 as word`
	`...`
	`PWM_Set_Duty(pwm_period1 div 2, 1)`

Notes	Number of available PWM channels depends on MCU. Refer to MCU datasheet for details.

PWM_Start

Prototype	`sub procedure PWM_Start(dim enable_channel_x as byte)`
Description	Starts PWM at requested channel.
Parameters	- `enable_channel_x`: number of PWM channel
Returns	Nothing.
Requires	MCU must have the HW PWM Module. PWM channel must be properly configured. See the PWM_Init and PWM_Set_Duty routines.
Example	`' start PWM at channel 1` `PWM_Start(1)`
Notes	Number of available PWM channels depends on MCU. Refer to MCU datasheet for details.

PWM_Stop

Prototype	`sub procedure PWM_Stop(dim disable_channel_x as byte)`
Description	Stops PWM at requested channel.
Parameters	- `disable_channel_x`: number of PWM channel
Returns	Nothing.
Requires	MCU must have the HW PWM Module.
Example	`' stop PWM at channel 1` `PWM_Stop(1)`
Notes	Number of available PWM channels depends on MCU. Refer to MCU datasheet for details.

Library Example

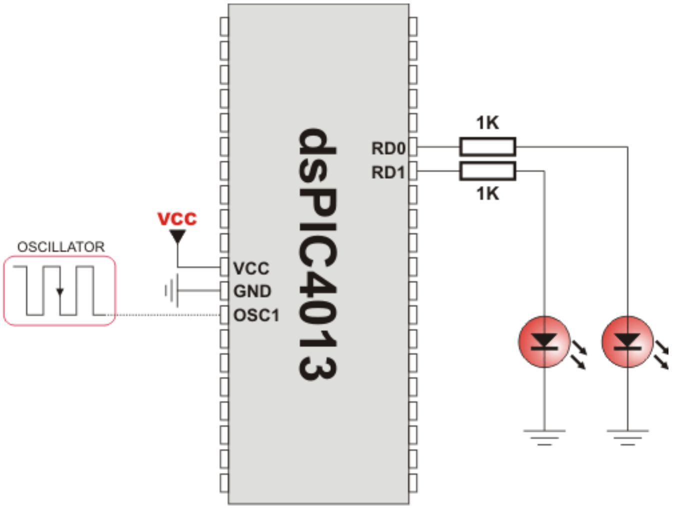
The example changes PWM duty ratio on channels 1 and 2 continuously. If LEDs are connected to channels 1 and 2, a gradual change of emitted light will be noticeable.

Copy Code To Clipboard

```
program Pwm_Demo
dim current_duty, old_duty, current_duty1, old_duty1 as word
    pwm_period1, pwm_period2 as word

sub procedure InitMain()
  ADPCFG = 0xFFFF                             ' initialize AN pins as digital
  TRISB = 0xFFFF                              ' configure PORTB pins as input
  PORTD = 0                                   ' set PORTD to 0
  TRISD = 0                                   ' designate PORTD pins as output
end sub

main:
  InitMain()
  current_duty  = 16                          ' initial value for current_duty
  current_duty1 = 16                          ' initial value for current_duty1
```

```
  pwm_period1 = PWM_Init(5000 , 1, 1, 2)
  pwm_period2 = PWM_Init(10000, 2, 1, 3)

  PWM_Start(1)
  PWM_Start(2)

  PWM_Set_Duty(current_duty,  1)              ' Set current duty for PWM1
  PWM_Set_Duty(current_duty1, 2)              ' Set current duty for PWM2

  while (TRUE)                                ' endless loop
    if RB0_bit = 1 then                       ' button on RB0 pressed
      Delay_ms(20)
      Inc(current_duty)                       ' increment current_duty
      if (current_duty > pwm_period1) then    ' if we increase current_duty greater then
possible pwm_period1 value
        current_duty = 0                      ' reset current_duty value to zero
      end if
      PWM_Set_Duty(current_duty,  1)          ' set newly acquired duty ratio
    end if

    if RB1_bit = 1 then                       ' button on RB1 pressed
      Delay_ms(20)
      Dec(current_duty)                       ' decrement current_duty
      if (current_duty > pwm_period1) then    ' if we decrease current_duty greater then
possible pwm_period1 value (overflow)
        current_duty = pwm_period1            ' set current_duty to max possible value
      end if
      PWM_Set_Duty(current_duty,  1)          ' set newly acquired duty ratio
    end if

    if RB2_bit = 1 then                       ' button on RB2 pressed
      Delay_ms(20)
      Inc(current_duty1)                      ' increment current_duty1
      if (current_duty1 > pwm_period2) then  ' if we increase current_duty1 greater then
possible pwm_period2 value
        current_duty1 = 0                     ' reset current_duty1 value to zero
      end if
      PWM_Set_Duty(current_duty1,  2)         ' set newly acquired duty ratio
    end if

    if RB3_bit = 1 then                       ' button on RB3 pressed
      Delay_ms(20)
      Dec(current_duty1)                      ' decrement current_duty1
      if (current_duty1 > pwm_period2) then  ' if we decrease current_duty1 greater then
possible pwm_period1 value (overflow)
        current_duty1 = pwm_period2           ' set current_duty to max possible value
      end if
      PWM_Set_Duty(current_duty1,  2)
    end if

       Delay_ms(5)                            ' slow down change pace a little
  wend
end.
```

HW Connection



PWM demonstration

PWM Motor Control Library

The PWM Motor Control module is available with a number of dsPIC30/33 MCUs. mikroBasic PRO for dsPIC30/33 and PIC24 provides a library which simplifies using the PWM Motor Control module.

Important:

- Number of PWM modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.
- PWM library routines require you to specify the module you want to use. To use the desired PWM module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.

Library Routines

- PWMx_Mc_Init
- PWMx_Mc_Set_Duty
- PWMx_Mc_Start
- PWMx_Mc_Stop

PWMx_Mc_Init

Prototype	`sub function PWMx_Mc_Init(dim freq_hz, pair_output_mode, enable_output_x, clock_prescale_output_postscale as word) as word`
Description	Initializes the Motor Control PWM module with duty ratio 0. The function calculates timer period, writes it to the MCU's PTPER register and returns it as the function result.
Parameters	- `freq_hz`: PWM frequency in Hz (refer to device datasheet for correct values in respect with Fosc) - `pair_output_mode`: output mode for output pin pairs: `1` = independent, `0` = complementary. If `pair_output_mode.B0` is equal to 1 then PWM channels PWM1L and PWM1H will be independent, If `pair_output_mode.B1` is equal to 0 then PWM channels PWM2L and PWM2H will be complementary, ... If `pair_output_mode.Bn` is equal to 1 then PWM channels PWM(n+1)L and PWM(n+1)H will be independent, If `pair_output_mode.Bn` is equal to 0 then PWM channels PWM(n+1)L and PWM(n+1)H will be complementary. - `enable_output_x`: bits <7..0> are enabling corresponding PWM channels <`PWM4H`, `PWM3H`, `PWM2H`, `PWM1H`, `PWM4L`, `PWM3L`, `PWM2L`, `PWM1L`>. If bit value is equal to `0` then corresponding PWM channel is disabled (pin is standard I/O). If bit value is equal to `1` then corresponding PWM channel is enabled (pin is PWM output). For detalied explanation consult the "Motor Control PWM Module" section in device datasheet - `clock_prescale_output_postscale`: PWM clock prescaler/postscaler settings. Values `<0..3> and <0..15>` correspond to prescaler/postscaler `<1:1, 1:4, 1:16, 1:64> and <1:1, 1:2, ..., 1:16>`
Returns	Calculated timer period.
Requires	The dsPIC30/33 MCU must have the Motor Control PWM module.

Example	`'Initializes the PWM1 module at 5KHz, complementary pin-pair output, output enabled on pins 4l..1l, no clock prescale and no clock postscale:`
	`dim duty_50 as word`
	`...`
	`duty_50 = PWM1_Mc_Init(5000, 1, $0F, 0)`

Notes	- Number of PWM modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library. - PWM library routines require you to specify the module you want to use. To use the desired PWM module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.

PWMx_Mc_Set_Duty

Prototype	`sub procedure PWM1_Mc_Set_Duty(dim duty, channel as word)`
	
	`' For dsPIC 33FJ MCUs that have PWM2 module :`
	`sub procedure PWM2_Mc_Set_Duty(dim duty as word)`

Description	The function changes PWM duty ratio.
Parameters	- `duty`: PWM duty ratio. Valid values: `0` to timer period returned by the PWMx_Mc_Init function. - `channel`: number of PWM channel to change duty to.
Returns	Nothing.
Requires	The dsPIC30/33 MCU must have the Motor Control PWM module. The PWM module needs to be initalized. See the PWMx_Mc_Init function.

Example	`' Set duty ratio to 50% at channel 1:`
	`PWM1_Mc_Init(5000,1,$F,0)`
	`...`
	`PWM1_Mc_Set_Duty(32767, 1)`

Notes	- Number of PWM modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library. - PWM library routines require you to specify the module you want to use. To use the desired PWM module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.

PWMx_Mc_Start

Prototype	`sub procedure PWMx_Mc_Start()`
Description	Starts the Motor Control PWM module (channels initialized in the PWMx_Mc_Init function).
Parameters	None.
Returns	Nothing.
Requires	The dsPIC30/33 MCU must have the Motor Control PWM module. The PWM module needs to be initalized. See the PWMx_Mc_Init function.
Example	`' start the Motor Control PWM1 module` `PWM1_Mc_Start()`
Notes	- Number of PWM modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library. - PWM library routines require you to specify the module you want to use. To use the desired PWM module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.

PWMx_Mc_Stop

Prototype	`sub procedure PWMx_Mc_Stop()`
Description	Stops the Motor Control PWM module.
Parameters	None.
Returns	Nothing.
Requires	The dsPIC30/33 MCU must have the Motor Control PWM module.
Example	`' stop the Motor Control PWM1 module` `PWM1_Mc_Stop()`
Notes	- Number of PWM modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library. - PWM library routines require you to specify the module you want to use. To use the desired PWM module, simply change the letter **x** in the routine prototype for a number from **1** to **2**.

Library Example

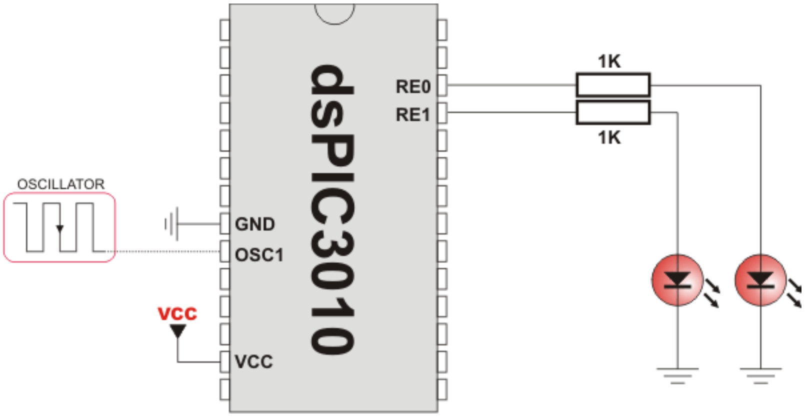
The example changes PWM duty ratio on channel 1 continually. If LED is connected to the channel 1, a gradual change of emitted light will be noticeable.

Copy Code To Clipboard

```
program PWM

dim pwm_period, current_duty as word

main:

  ADPCFG = 0xFFFF                               ' initialize AN pins as digital
  PORTB  = 0
  TRISB  = 0                                    ' initialize portb as output
  current_duty = 10
  Delay_ms(1000)

  pwm_period = PWM1_MC_Init(5000, 1, 0x01, 0)    ' Pwm_Mc_Init returns calculated timer
period.
  PWM1_MC_Set_Duty(current_duty, 1)
  PWM1_MC_Start()

  while (TRUE)                                  ' Endless loop
    if (RB0_bit) then                           ' Button on RB0 pressed
      Delay_ms(20)
      Inc(current_duty)                      ' Increment current_duty
      if (current_duty > pwm_period) then    ' If we increase current_duty greater then
possible pwm_period value
        current_duty = 0                     ' reset current_duty value to zero
      end if
      PWM1_MC_Set_Duty(current_duty,  1)     ' Set newly acquired duty ratio
    end if

if (RB1_bit) then                            ' Button on RB1 pressed
      Delay_ms(20)
      Dec(current_duty)                      ' Decrement current_duty
```

```
if (current_duty > pwm_period)  then         ' If we decrease current_duty greater then
possible pwm_period value (overflow)
        current_duty = pwm_period          ' set current_duty to max possible value
      end if
      PWM1_MC_Set_Duty(current_duty,  1)          ' Set newly acquired duty ratio
    end if
    Delay_ms(5)                                   ' Slow down change pace a little
  wend
end.
```

HW Connection



PWM Motor Control demonstration

RS-485 Library

RS-485 is a multipoint communication which allows multiple devices to be connected to a single bus. mikroBasic PRO for dsPIC30/33 and PIC24 provides a set of library routines for comfortable work with RS485 system using Master/Slave architecture. Master and Slave devices interchange packets of information. Each of these packets contains synchronization bytes, CRC byte, address byte and the data. Each Slave has unique address and receives only packets addressed to it. The Slave can never initiate communication.

It is the user’s responsibility to ensure that only one device transmits via 485 bus at a time.

The RS-485 routines require the UART module. Pins of UART need to be attached to RS-485 interface transceiver, such as LTC485 or similar (see schematic at the bottom of this page).

Library constants:

- START byte value = `150`
- STOP byte value = `169`
- Address `50` is the broadcast address for all Slaves (packets containing address `50` will be received by all Slaves except the Slaves with addresses `150` and `169`).

Important:

- The library uses the UART module for communication. The user must initialize the appropriate UART module before using the RS-485 Library.
- For MCUs with multiple UART modules it is possible to initialize them and then switch by using the UART_Set_Active routine.

Library Dependency Tree

External dependencies of RS-485 Library

The following variable must be defined in all projects using RS-485 Library:	Description:	Example:
`dim RS485_rxtx_pin as sbit sfr external`	Control RS-485 Transmit/Receive operation mode	`dim RS485_rxtx_pin as sbit at RF2_bit`
`dim RS485_rxtx_pin_direction as sbit sfr external`	Direction of the RS-485 Transmit/Receive pin	`dim RS485_rxtx_pin_direction as sbit at TRISF2_bit`

Library Routines

- RS485Master_Init
- RS485Master_Receive
- RS485Master_Send
- RS485Slave_Init
- RS485Slave_Receive
- RS485Slave_Send

RS485Master_Init

Prototype	`sub procedure RS485Master_Init()`
Description	Initializes MCU as a Master for RS-485 communication.
Parameters	None.
Returns	Nothing.
Requires	Global variables: - `RS485_rxtx_pin` - this pin is connected to RE/DE input of RS-485 transceiver(see schematic at the bottom of this page). RE/DE signal controls RS-485 transceiver operation mode. - `RS485_rxtx_pin_direction` - direction of the RS-485 Transmit/Receive pin. must be defined before using this routine. UART HW module needs to be initialized. See UARTx_Init.

Example

```
' RS485 module pinout
dim RS485_rxtx_pin as sbit at RF2_bit
dim RS485_rxtx_pin_direction as sbit at TRISF2_bit
' End of RS485 module pinout
...
UART1_Init(9600)      ' initialize UART1 module
RS485Master_Init()    ' intialize MCU as a Master for RS-485 communication
```

Notes	None

RS485Master_Receive

Prototype	`sub procedure RS485Master_Receive(dim byref data as byte[10])`
Description	Receives messages from Slaves. Messages are multi-byte, so this routine must be called for each byte received.
Parameters	- `data_buffer`: 7 byte buffer for storing received data. Data will be stored in the following manner: - `data_buffer[0..2]`: message content - `data_buffer[3]`: number of message bytes received, 1–3 - `data_buffer[4]`: is set to 255 when message is received - `data_buffer[5]`: is set to 255 if error has occurred - `data_buffer[6]`: address of the Slave which sent the message The routine automatically adjusts `data[4]` and `data[5]` upon every received message. These flags need to be cleared by software.
Returns	Nothing.
Requires	MCU must be initialized as a Master for RS-485 communication. See RS485Master_Init.

Example	`dim msg as byte[8]`
	`...`
	`RS485Master_Receive(msg)`

Notes	None

RS485Master_Send

Prototype	`sub procedure RS485Master_Send(dim byref buffer as byte[20], dim datalen as byte, dim address as byte)`
Description	Sends message to Slave(s). Message format can be found at the bottom of this page.
Parameters	- `data_buffer`: data to be sent - `datalen`: number of bytes for transmition. Valid values: 0...3. - `slave_address`: Slave(s) address
Returns	Nothing.
Requires	MCU must be initialized as a Master for RS-485 communication. See RS485Master_Init. It is the user's responsibility to ensure (by protocol) that only one device sends data via 485 bus at a time.

Example	`dim msg as byte[8]`
	`...`
	`' send 3 bytes of data to slave with address 0x12`
	`RS485Master_Send(msg, 3, 0x12)`

Notes	None

RS485Slave_Init

Prototype	`sub procedure RS485Slave_Init(dim slave_address as byte)`
Description	Initializes MCU as a Slave for RS-485 communication.
Parameters	- `Slave_address`: Slave address
Returns	Nothing.
Requires	Global variables: - `RS485_rxtx_pin` - this pin is connected to RE/DE input of RS-485 transceiver(see schematic at the bottom of this page). RE/DE signal controls RS-485 transceiver operation mode. Valid values: `1` (for transmitting) and `0` (for receiving - `RS485_rxtx_pin_direction` - direction of the RS-485 Transmit/Receive pin. must be defined before using this routine. UART HW module needs to be initialized. See UARTx_Init.

Example

Initialize MCU as a Slave with address 160:

```
' RS485 module pinout
dim RS485_rxtx_pin as sbit  at RF2_bit
dim RS485_rxtx_pin_direction as sbit  at TRISF2_bit
' End of RS485 module pinout
...
UART1_Init(9600)          ' initialize UART1 module
RS485Slave_Init(160)      ' intialize MCU as a Slave for RS-485 communication
with address 160
```

Notes	None

RS485Slave_Receive

Prototype	`sub procedure RS485Slave_Receive(dim byref data as byte[20])`
Description	Receives messages from Master. If Slave address and Message address field don't match then the message will be discarded. Messages are multi-byte, so this routine must be called for each byte received.
Parameters	- `data_buffer`: 6 byte buffer for storing received data, in the following manner: - `data_buffer[0..2]`: message content - `data_buffer[3]`: number of message bytes received, 1–3 - `data_buffer[4]`: is set to 255 when message is received - `data_buffer[5]`: is set to 255 if error has occurred The routine automatically adjusts `data[4]` and `data[5]` upon every received message. These flags need to be cleared by software.
Returns	Nothing.
Requires	MCU must be initialized as a Slave for RS-485 communication. See RS485Slave_Init.

Example	`dim msg as byte[8]`
	`...`
	`RS485Slave_Read(msg)`

Notes	None

RS485Slave_Send

Prototype	`sub procedure RS485Slave_Send(dim byref data as byte[20], dim datalen as byte)`
Description	Sends message to Master. Message format can be found at the bottom of this page.
Parameters	- `data_buffer`: data to be sent - `datalen`: number of bytes for transmition. Valid values: 0...3.
Returns	Nothing.
Requires	MCU must be initialized as a Slave for RS-485 communication. See RS485Slave_Init. It is the user's responsibility to ensure (by protocol) that only one device sends data via 485 bus at a time.

Example	`dim msg as byte[8]`
	`...`
	`' send 2 bytes of data to the Master`
	`RS485Slave_Send(msg, 2)`

Notes	None

Library Example

The example demonstrates working with the dsPIC as a Master node in RS-485 communication. Master sends message to Slave with address 160 and waits for a response. After the response is received, the first byte of received data is incremented and sent back to the Slave. The received data is displayed on PORTB while error on receiving (0xAA) and number of consecutive unsuccessful retries are displayed on PORTD. Hardware configurations in this example are made for the EasydsPIC4A board and dsPIC30F4013.

RS485 Master code:

Copy Code To Clipboard

```
program RS485_Master_Example

dim dat as byte[10]                          ' buffer for receving/sending messages
    i, j as byte
    cnt as longint

dim rs485_rxtx_pin  as sbit at RF2_bit                 ' set transcieve pin
    rs485_rxtx_pin_direction as sbit at TRISF2_bit     ' set transcieve pin direction

' Interrupt routine
sub procedure interrupt() org IVT_ADDR_U2RXINTERRUPT
  RS485Master_Receive(dat)
  U2RXIF_bit = 0                          ' ensure interrupt not pending
end sub

main:
  cnt = 0

  ADPCFG = 0xFFFF

  PORTB = 0
  PORTD = 0
  TRISB = 0
  TRISD = 0

  UART2_Init(9600)                        ' initialize UART2 module
  Delay_ms(100)

  RS485Master_Init()                      ' initialize MCU as Master

  dat[0] = 0xAA
  dat[1] = 0xF0
  dat[2] = 0x0F
  dat[4] = 0                              ' ensure that message received flag is 0
  dat[5] = 0                              ' ensure that error flag is 0
  dat[6] = 0

  RS485Master_Send(dat,1,160)

  URXISEL1_U2STA_bit = 0
  URXISEL1_U2STA_bit = 0
  NSTDIS_bit = 1                          ' no nesting of interrupts
  U2RXIF_bit = 0                          ' ensure interrupt not pending
```

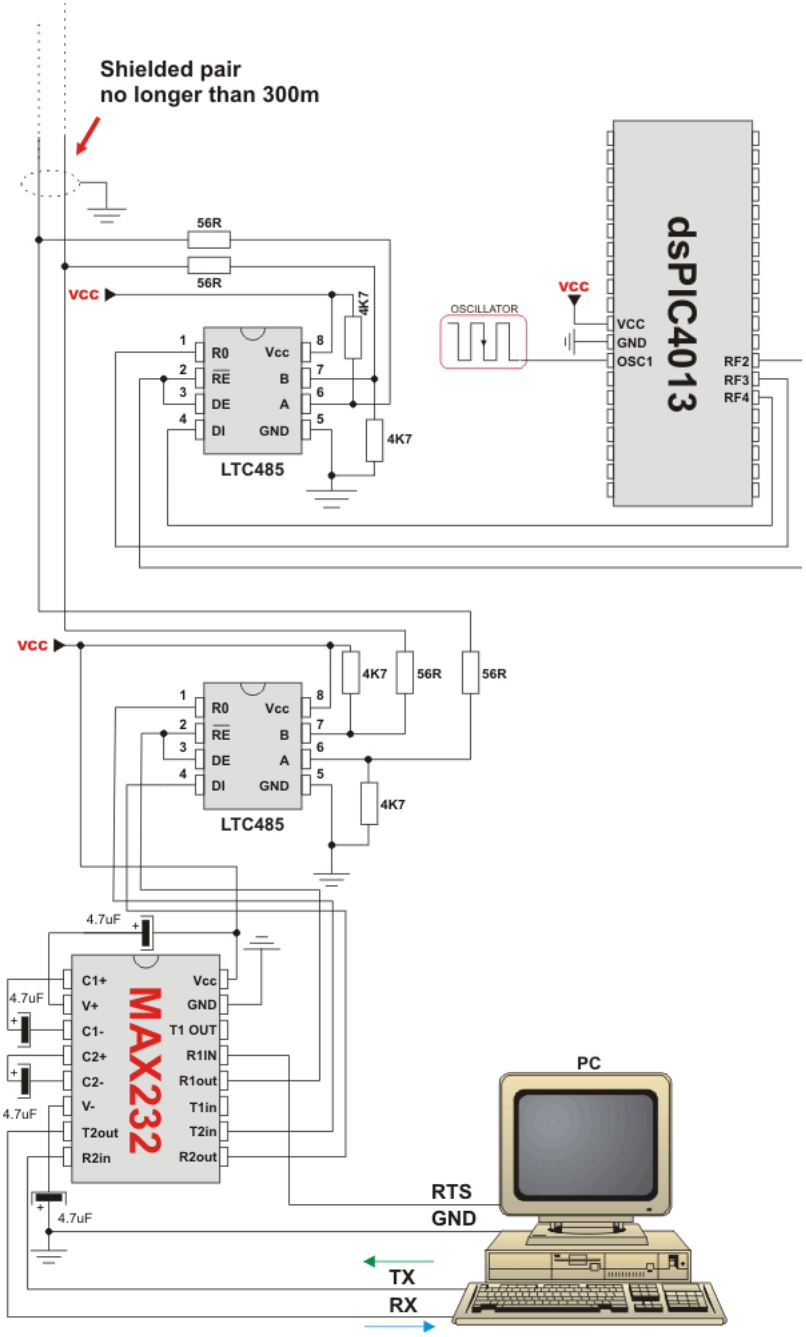
```
  U2RXIE_bit = 1                      ' enable intterupt

  while (TRUE)
                                      ' upon completed valid message receiving
                                      '   data[4] is set to 255

    Inc(cnt)
    if (dat[5] <> 0) then             ' if an error detected, signal it
      PORTD = 0xAA                    '   by setting portd to 0xAA
    end if
    if (dat[4] <> 0) then             ' if message received successfully
      cnt = 0
      dat[4] = 0                      ' clear message received flag
      j = dat[3]
      for i = 1 to dat[3]             ' show data on PORTB
        PORTB = dat[i-1]
      next i
      dat[0] = dat[0]+1               ' send back to master
      Delay_ms(1)
      RS485Master_Send(dat,1,160)
    end if

    if (cnt > 100000) then            ' if in 100000 poll-cycles the answer
      Inc(PORTD)                      '   was not detected, signal
      cnt = 0                         '   failure of send-message
      RS485Master_Send(dat,1,160)
      if (PORTD > 10) then               ' if sending failed 10 times
        RS485Master_Send(dat,1,50)    '   send message on broadcast address
      end if
    end if
  wend
end.
```

Copy Code To Clipboard

```
program RS485_Slave_Example

dim dat as byte[20]                            ' buffer for receving/sending messages
    i, j as byte

dim rs485_rxtx_pin as sbit at RF2_bit                    ' set transcieve pin
    rs485_rxtx_pin_direction as sbit at TRISF2_bit       ' set transcieve pin direction

' Interrupt routine
sub procedure interrupt() org IVT_ADDR_U2RXINTERRUPT
  RS485Slave_Receive(dat)
  U2RXIF_bit = 0                   ' ensure interrupt not pending
end sub

main:
  ADPCFG = 0xFFFF
```

```
  PORTB = 0
  PORTD = 0
  TRISB = 0
  TRISD = 0

  UART2_Init(9600)                 ' initialize UART2 module
  Delay_ms(100)

  RS485Slave_Init(160)             ' Intialize MCU as slave, address 160

  dat[0] = 0xAA
  dat[1] = 0xF0
  dat[2] = 0x0F
  dat[4] = 0                       ' ensure that message received flag is 0
  dat[5] = 0                       ' ensure that error flag is 0
  dat[6] = 0

  URXISEL1_U2STA_bit = 0
  URXISEL1_U2STA_bit = 0
  NSTDIS_bit = 1                   ' no nesting of interrupts
  U2RXIF_bit = 0                   ' ensure interrupt not pending
  U2RXIE_bit = 1                   ' enable intterupt

  while (TRUE)
    if (dat[5] <> 0) then          ' if an error detected, signal it by
      PORTD = 0xAA                 '   setting portd to 0xAA
      dat[5] = 0
    end if
    if (dat[4] <> 0) then          ' upon completed valid message receive
      dat[4] = 0                   '   data[4] is set to 0xFF
      j = dat[3]

      for i = 1 to dat[3]          ' show data on PORTB
        PORTB = dat[i-1]
      next i

      dat[0] = dat[0]+1            ' increment received dat[0]
      Delay_ms(1)
      RS485Slave_Send(dat,1)       '   and send it back to master
    end if
  wend
end.
```

HW Connection



Example of interfacing PC to dsPIC MCU via RS485 bus with LTC485 as RS-485 transceiver

Message format and CRC calculations

Q: How is CRC checksum calculated on RS485 master side?

Copy Code To Clipboard

```
_START_BYTE = 0x96;  ' 10010110
_STOP_BYTE  = 0xA9;  ' 10101001

PACKAGE:
--------
_START_BYTE 0x96
ADDRESS
DATALEN
[DATA1]             ' if exists
[DATA2]             ' if exists
[DATA3]             ' if exists
CRC
_STOP_BYTE  0xA9


DATALEN bits
------------
bit7 = 1  MASTER SENDS
       0  SLAVE  SENDS
bit6 = 1  ADDRESS WAS XORed with 1, IT WAS EQUAL TO _START_BYTE or _STOP_BYTE
       0  ADDRESS UNCHANGED
bit5 = 0  FIXED
bit4 = 1  DATA3 (if exists) WAS XORed with 1, IT WAS EQUAL TO _START_BYTE or _STOP_
BYTE
       0  DATA3 (if exists) UNCHANGED
bit3 = 1  DATA2 (if exists) WAS XORed with 1, IT WAS EQUAL TO _START_BYTE or _STOP_
BYTE
       0  DATA2 (if exists) UNCHANGED
bit2 = 1  DATA1 (if exists) WAS XORed with 1, IT WAS EQUAL TO _START_BYTE or _STOP_
BYTE
       0  DATA1 (if exists) UNCHANGED
bit1bit0 = 0 to 3 NUMBER OF DATA BYTES SEND

CRC generation :
----------------
crc_send = datalen  xor address
crc_send = crc_send xor data[0]    ' if exists
crc_send = crc_send xor data[1]    ' if exists
crc_send = crc_send xor data[2]    ' if exists
crc_send = crc_send not crc_send
if ((crc_send = START_BYTE) or (crc_send = STOP_BYTE)) then
   crc_send = crc_send + 1
end if
NOTE: DATALEN<4..0> can not take the START_BYTE<4..0> or STOP_BYTE<4..0> values.
```

Software I²C Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides routines for implementing Software I²C communication. These routines are hardware independent and can be used with any MCU. The Software I²C library enables you to use MCU as Master in I²C communication. Multi-master mode is not supported.

Important:

- This library implements time-based activities, so interrupts need to be disabled when using Software I²C.
- All Software I²C Library functions are blocking-call functions (they are waiting for I²C clock line to become logical one).
- The pins used for the Software I²C communication should be connected to the pull-up resistors. Turning off the LEDs connected to these pins may also be required.
- Every Software I²C library routine has its own counterpart in Hardware I²C library, except `I2C_Repeated_Start.` `Soft_I2C_Start` is used instead of `I2C_Repeated_Start`.
- Working clock frequency of the Software I²C is 20kHz.

External dependencies of Software I²C Library

The following variable must be defined in all projects using RS-485 Library:	**Description:**	**Example:**
`dim Soft_I2C_Scl as sbit sfr external`	Soft I²C Clock line.	`dim Soft_I2C_Scl as sbit at RF3_bit`
`dim Soft_I2C_Sda as sbit sfr external`	Soft I²C Data line.	`dim Soft_I2C_Sda as sbit at RF2_bit`
`dim Soft_I2C_Scl_Direction as sbit sfr external`	Direction of the Soft I²C Clock pin.	`dim Soft_I2C_Scl_Direction as sbit at TRISF3_bit`
`dim Soft_I2C_Sda_Direction as sbit sfr external`	Direction of the Soft I²C Data pin.	`dim Soft_I2C_Sda_Direction as sbit at TRISF2_bit`

Library Routines

- Soft_I2C_Init
- Soft_I2C_Start
- Soft_I2C_Read
- Soft_I2C_Write
- Soft_I2C_Stop
- Soft_I2C_Break

Soft_I2C_Init

Prototype	`sub procedure Soft_I2C_Init()`
Description	Configures the software I²C module.
Parameters	None.
Returns	Nothing.
Requires	Global variables: - `Soft_I2C_Scl`: Soft I²C clock line - `Soft_I2C_Sda`: Soft I²C data line - `Soft_I2C_Scl_Pin_Direction`: Direction of the Soft I²C clock pin - `Soft_I2C_Sda_Pin_Direction`: Direction of the Soft I²C data pin must be defined before using this function.

Example

```
' Software I2C connections
dim Soft_I2C_Scl           as sbit at RF3_bit
    Soft_I2C_Sda           as sbit at RF2_bit
    Soft_I2C_Scl_Direction as sbit at TRISF3_bit
    Soft_I2C_Sda_Direction as sbit at TRISF2_bit
' End Software I2C connections
...
Soft_I2C_Init()
```

Notes	None

Soft_I2C_Start

Prototype	`sub procedure Soft_I2C_Start()`
Description	Determines if the I²C bus is free and issues START signal.
Parameters	None.
Returns	Nothing.
Requires	Software I²C must be configured before using this function. See Soft_I2C_Init routine.

Example

```
' Issue START signal
Soft_I2C_Start()
```

Notes	None

Soft_I2C_Read

Prototype	`sub function Soft_I2C_Read(dim ack as word) as byte`
Description	Reads one byte from the slave.
Parameters	- `ack`: acknowledge signal parameter. If the `ack==0` *not acknowledge* signal will be sent after reading, otherwise *the acknowledge* signal will be sent.
Returns	One byte from the Slave.
Requires	Soft I^{2}C must be configured before using this function. See Soft_I2C_Init routine. Also, START signal needs to be issued in order to use this function. See Soft_I2C_Start routine.

Example	`dim take as byte`
	`...`
	`' Read data and send the not_acknowledge signal`
	`take = Soft_I2C_Read(0)`

Notes	None

Soft_I2C_Write

Prototype	`sub function Soft_I2C_Write(dim data_ as byte) as byte`
Description	Sends data byte via the I^{2}C bus.
Parameters	- `data_`: data to be sent
Returns	- `0` if there were no errors. - `1` if write collision was detected on the I^{2}C bus.
Requires	Soft I^{2}C must be configured before using this function. See Soft_I2C_Init routine. Also, START signal needs to be issued in order to use this function. See Soft_I2C_Start routine.

Example	`dim data_, error as byte`
	`...`
	`error = Soft_I2C_Write(data_)`
	`error = Soft_I2C_Write($A3)`

Notes	None

Soft_I2C_Stop

Prototype	`sub procedure Soft_I2C_Stop()`
Description	Issues STOP signal.
Parameters	None.
Returns	Nothing.
Requires	Soft I^{2}C must be configured before using this function. See Soft_I2C_Init routine.
Example	`' Issue STOP signal` `Soft_I2C_Stop()`
Notes	None

Soft_I2C_Break

Prototype	`sub procedure Soft_I2C_Break()`
Description	All Software I²C Library functions can block the program flow (see note at the top of this page). Calling this routine from interrupt will unblock the program execution. This mechanism is similar to WDT.
Parameters	None.
Returns	Nothing.
Requires	Nothing.

Example

```
dim data1, error, counter as byte

sub procedure Timer1Int() org IVT_ADDR_T1INTERRUPT
  counter = 0
  if (counter >= 20)
    Soft_I2C_Break()
    counter = 0                ' reset counter
  else
    Inc(counter)               ' increment counter
  end if

  T1IF_bit = 0                 ' Clear Timer1 overflow interrupt flag
end sub

main:

  ...

  ' try Soft_I2C_Init with blocking prevention mechanism
  IPC0   = IPC0 or 0x1000      ' Interrupt priority level = 1
  T1IE_bit = 1                 ' Enable Timer1 interrupts
  T1CON = 0x8030     ' Timer1 ON, internal clock FCY, prescaler 1:256
  Soft_I2C_Init()
  T1IE_bit = 0                 ' Disable Timer1 interrupts
end.
```

Notes	Interrupts should be disabled before using Software I²C routines again (see note at the top of this page).

Library Example

The example demonstrates use of the Software I²C Library. The dsPIC30/33 or PIC24 MCU is connected (SCL, SDA pins) to PCF8583 RTC (real-time clock). Program sends date/time to RTC.

Copy Code To Clipboard

```
program RTC_Read

dim seconds, minutes, hours, day, month_, year as byte     ' Global date/time variables

 ' Software I2C connections
dim Soft_I2C_Scl           as sbit at RF3_bit
    Soft_I2C_Sda           as sbit at RF2_bit
    Soft_I2C_Scl_Direction as sbit at TRISF3_bit
    Soft_I2C_Sda_Direction as sbit at TRISF2_bit
 ' End Software I2C connections

 ' LCD module connections
dim LCD_RS as sbit at LATD0_bit
dim LCD_EN as sbit at LATD1_bit
dim LCD_D4 as sbit at LATB0_bit
dim LCD_D5 as sbit at LATB1_bit
dim LCD_D6 as sbit at LATB2_bit
dim LCD_D7 as sbit at LATB3_bit

dim LCD_RS_Direction as sbit at TRISD0_bit
dim LCD_EN_Direction as sbit at TRISD1_bit
dim LCD_D4_Direction as sbit at TRISB0_bit
dim LCD_D5_Direction as sbit at TRISB1_bit
dim LCD_D6_Direction as sbit at TRISB2_bit
dim LCD_D7_Direction as sbit at TRISB3_bit
 ' End LCD module connections

 '---------------------- Reads time and date information from RTC (PCF8583)
sub procedure Read_Time()
  Soft_I2C_Start()                ' Issue start signal
  Soft_I2C_Write(0xA0)            ' Address PCF8583, see PCF8583 datasheet
  Soft_I2C_Write(2)               ' Start from address 2
  Soft_I2C_Start()                ' Issue repeated start signal
  Soft_I2C_Write(0xA1)            ' Address PCF8583 for reading R/W=1
  seconds = Soft_I2C_Read(1)      ' Read seconds byte
  minutes = Soft_I2C_Read(1)      ' Read minutes byte
  hours = Soft_I2C_Read(1)        ' Read hours byte
  day = Soft_I2C_Read(1)          ' Read year/day byte
  month_ = Soft_I2C_Read(0)       ' Read weekday/month byte}

  Soft_I2C_Stop()                 ' Issue stop signal}
end sub

 '--------------------- Formats date and time
sub procedure Transform_Time()
```

```
  seconds =  ((seconds and 0xF0) >> 4)*10 + (seconds and 0x0F)   ' Transform seconds
  minutes =  ((minutes and 0xF0) >> 4)*10 + (minutes and 0x0F)   ' Transform months
  hours   =  ((hours and 0xF0)  >> 4)*10  + (hours and 0x0F)     ' Transform hours
  year    =   (day and 0xC0) >> 6                                ' Transform year
  day     =  ((day and 0x30) >> 4)*10    + (day and 0x0F)        ' Transform day
  month_  =  ((month_ and 0x10)  >> 4)*10 + (month_ and 0x0F)    ' Transform month
end sub

'-------------------- Output values to LCD
sub procedure Display_Time()
  Lcd_Chr(1, 6, (day / 10)   + 48)     ' Print tens digit of day variable
  Lcd_Chr(1, 7, (day mod 10)   + 48)   ' Print oness digit of day variable
  Lcd_Chr(1, 9, (month_ / 10) + 48)
  Lcd_Chr(1,10, (month_ mod 10) + 48)
  Lcd_Chr(1,15,  year        + 57)     ' Print year vaiable + 9 (start from year 2009)

  Lcd_Chr(2, 6, (hours / 10)   + 48)
  Lcd_Chr(2, 7, (hours mod 10)   + 48)
  Lcd_Chr(2, 9, (minutes / 10) + 48)
  Lcd_Chr(2,10, (minutes mod 10) + 48)
  Lcd_Chr(2,12, (seconds / 10) + 48)
  Lcd_Chr(2,13, (seconds mod 10) + 48)
end sub

'------------------ Performs project-wide init
sub procedure Init_Main()
  ADPCFG = 0xFFFF            ' initialize AN pins as digital

  Soft_I2C_Init()            ' Initialize Soft I2C communication
  Lcd_Init()                 ' Initialize LCD
  Lcd_Cmd(_LCD_CLEAR)        ' Clear LCD display
  Lcd_Cmd(_LCD_CURSOR_OFF)   ' Turn cursor off

  Lcd_Out(1,1,"Date:")       ' Prepare and output static text on LCD
  Lcd_Chr(1,8,":")
  Lcd_Chr(1,11,":")
  Lcd_Out(2,1,"Time:")
  Lcd_Chr(2,8,":")
  Lcd_Chr(2,11,":")
  Lcd_Out(1,12,"200")
end sub

'----------------- Main procedure
main:

  Delay_ms(1000)

Init_Main()                 ' Perform initialization

  while TRUE                ' Endless loop
    Read_Time()             ' Read time from RTC(PCF8583)
    Transform_Time()        ' Format date and time
    Display_Time()          ' Prepare and display on LCD
  wend
end.
```

Software SPI Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides routines for implementing Software SPI communication. These routines are hardware independent and can be used with any MCU. The Software SPI Library provides easy communication with other devices via SPI: A/D converters, D/A converters, MAX7219, LTC1290, etc.

Library configuration:

- SPI to Master mode
- Clock value = 20 kHz.
- Data sampled at the middle of interval.
- Clock idle state low.
- Data sampled at the middle of interval.
- Data transmitted at low to high edge.

The library configures SPI to the master mode, clock = 20kHz, data sampled at the middle of interval, clock idle state low and data transmitted at low to high edge.

Important : The Software SPI library implements time-based activities, so interrupts need to be disabled when using it.

External dependencies of Software SPI Library

The following variables must be defined in all projects using Software SPI Library:	Description:	Example:
`dim SoftSpi_SDI as sbit sfr external`	Data In line.	`dim SoftSpi_SDI as sbit at RF2_bit`
`dim SoftSpi_SDO as sbit sfr external`	Data Out line.	`dim SoftSpi_SDO as sbit at LATF3_bit`
`dim SoftSpi_CLK as sbit sfr external`	Clock line.	`dim SoftSpi_CLK as sbit at LATF6_bit`
`dim SoftSpi_SDI_Direction as sbit sfr external`	Direction of the Data In pin.	`dim SoftSpi_SDI_Direction as sbit at TRISF2_bit`
`dim SoftSpi_SDO_Direction as sbit sfr external`	Direction of the Data Out pin	`dim SoftSpi_SDO_Direction as sbit at TRISF3_bit`
`dim SoftSpi_CLK_Direction as sbit sfr external`	Direction of the Clock pin.	`dim SoftSpi_CLK_Direction as sbit at TRISF6_bit`

Library Routines

- Soft_SPI_Init
- Soft_SPI_Read
- Soft_SPI_Write

Soft_SPI_Init

Prototype	`sub procedure Soft_SPI_Init()`
Description	Routine initializes the software SPI module.
Parameters	None.
Returns	Nothing.
Requires	Global variables: - `SoftSpi_SDI`: Data in line - `SoftSpi_SDO`: Data out line - `SoftSpi_CLK`: Data clock line - `SoftSpi_SDI_Direction`: Direction of the Data in pin - `SoftSpi_SDO_Direction`: Direction of the Data out pin - `SoftSpi_CLK_Direction`: Direction of the Data clock pin must be defined before using this function.

Example	`' DAC module connections`
	`dim SoftSpi_CLK as sbit at LATF6_bit`
	`dim SoftSpi_SDI as sbit at RF2_bit`
	`dim SoftSpi_SDO as sbit at LATF3_bit`
	`dim SoftSpi_CLK_Direction as sbit at TRISF6_bit`
	`dim SoftSpi_SDI_Direction as sbit at TRISF2_bit`
	`dim SoftSpi_SDO_Direction as sbit at TRISF3_bit`
	`' End DAC module connections`
	`...`
	`Soft_SPI_Init() ' Init Soft_SPI`

Notes	None.

Soft_SPI_Read

Prototype	`sub function Soft_SPI_Read(dim data_ as byte) as byte`
Description	This routine performs 3 operations simultaneously. It provides clock for the Software SPI bus, reads a byte and sends a byte.
Parameters	- `sdata`: data to be sent.
Returns	Byte received via the SPI bus.
Requires	Soft SPI must be initialized before using this function. See Soft_SPI_Init routine.

Example	`dim data_read, data_send as byte`
	`...`
	`' Read a byte and assign it to data_read variable`
	`' (data_send byte will be sent via SPI during the Read operation)`
	`data_read = Soft_SPI_Read(data_send)`

Notes	None

Soft_SPI_Write

Prototype	`sub procedure Soft_SPI_Write(dim data_ as byte)`
Description	This routine sends one byte via the Software SPI bus.
Parameters	- `sdata`: data to be sent.
Returns	Nothing.
Requires	Soft SPI must be initialized before using this function. See Soft_SPI_Init.
Example	`' Write a byte to the Soft SPI bus` `Soft_SPI_Write($AA)`
Notes	None

Library Example

This code demonstrates using library routines for Soft_SPI communication. Also, this example demonstrates working with Microchip's MCP4921 12-bit D/A converter.

Copy Code To Clipboard

```
program Soft_SPI

' DAC module connections
dim Chip_Select as sbit at LATF0_bit
    SoftSpi_CLK as sbit at LATF6_bit
    SoftSpi_SDI as sbit at RF2_bit
    SoftSpi_SDO as sbit at LATF3_bit

dim Chip_Select_Direction as sbit at TRISF0_bit
    SoftSpi_CLK_Direction as sbit at TRISF6_bit
    SoftSpi_SDI_Direction as sbit at TRISF2_bit
    SoftSpi_SDO_Direction as sbit at TRISF3_bit
' End DAC module connections

dim value as word

sub procedure InitMain()
  TRISB0_bit = 1                          ' Set RB0 pin as input
  TRISB1_bit = 1                          ' Set RB1 pin as input
  Chip_Select = 1                         ' Deselect DAC
  Chip_Select_Direction = 0               ' Set CS# pin as Output
  Soft_Spi_Init()                         ' Initialize Soft_SPI
end sub

' DAC increments (0..4095) --> output voltage (0..Vref)
sub procedure DAC_Output(dim valueDAC as word)
dim temp as byte volatile

  Chip_Select = 0                         ' Select DAC chip
```

```
  ' Send High Byte
  temp = word(valueDAC >> 8) and 0x0F      ' Store valueDAC[11..8] to temp[3..0]
  temp = temp or 0x30                      ' Define DAC setting, see MCP4921 datasheet
  Soft_SPI_Write(temp)                     ' Send high byte via Soft SPI

   ' Send Low Byte
  temp = valueDAC                          ' Store valueDAC[7..0] to temp[7..0]
  Soft_SPI_Write(temp)                     ' Send low byte via Soft SPI

  Chip_Select = 1                          ' Deselect DAC chip
end sub

main:

  ADPCFG = 0xFFFF                          ' Configure AN pins as digital

  InitMain()                               ' Perform main initialization

  value = 2048                             ' When program starts, DAC gives
                                           '   the output in the mid-range

  while (TRUE)                               ' Endless loop
    if ((RB0_bit) and (value < 4095)) then   ' If RB0 button is pressed
      Inc(value)                             ' increment value
    else
      if ((RB1_bit) and (value > 0)) then    ' If RB1 button is pressed
        Dec(value)                           ' decrement value
      end if
    end if

    DAC_Output(value)                        ' Send value to DAC chip
    Delay_ms(1)                              ' Slow down key repeat pace
  wend
end.
```

Software UART Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides library which implements Software UART communication. These routines are hardware independent and can be used with any MCU. The Software UART Library provides easy communication with other devices via the RS232 protocol.

Important: The Software UART library implements time-based activities, so interrupts need to be disabled when using it.

Library Routines

- Soft_UART_Init
- Soft_UART_Read
- Soft_UART_Write
- Soft_UART_Break

Soft_UART_Init

Prototype	`sub function Soft_UART_Init(dim byref port as word, dim rx, tx as word, dim baud_rate as longword, dim inverted as word)`
Description	Configures and initializes the software UART module. Software UART routines use Delay_Cyc routine. If requested baud rate is too low then calculated parameter for calling `Delay_Cyc` exceeds `Delay_Cyc` argument range. If requested baud rate is too high then rounding error of `Delay_Cyc` argument corrupts Software UART timings.
Parameters	- `port`: software UART port address - `rx`: receiver pin - `tx`: transmiter pin - `baud_rate`: requested baudrate. Maximum baud rate depends on the MCU's clock and working conditions - `inverted`: if set to non-zero value, indicates inverted logic on output
Returns	- `2` - error, requested baud rate is too low - `1` - error, requested baud rate is too high - `0` - successful initialization
Requires	Nothing.

Example	This will initialize software UART and establish the communication at 9600 bps:
	`' Initialize Software UART communication on pins RB1(Rx), RB2(Tx), at 9600 bps`
	`Soft_UART_Init(PORTB, 1, 2, 9600, 0)`

Notes	The Software UART library implements time-based activities, so interrupts need to be disabled when using it.

Soft_UART_Read

Prototype	`sub function Soft_UART_Read(dim byref error as byte) as byte`
Description	The function receives a byte via software UART. This is a blocking function call (waits for start bit). Programmer can unblock it by calling Soft_UART_Break routine.
Parameters	- `error`: Error flag. Error code is returned through this variable. Values: - `0` - no error - `1` - stop bit error - `255` - user abort, Soft_UART_Break called
Returns	Byte received via UART.
Requires	Software UART must be initialized before using this function. See the Soft_UART_Init routine.

Example

```
dim data_ as byte
    error as word
  ...
  ' wait until data is received
do
  data_ = Soft_UART_Read(error)
loop until (error = 0)
```

Notes	The Software UART library implements time-based activities, so interrupts need to be disabled when using it.

Soft_UART_Write

Prototype	`sub procedure Soft_UART_Write(dim udata as byte)`
Description	This routine sends one byte via the Software UART bus.
Parameters	- `udata`: data to be sent.
Returns	Nothing.
Requires	Software UART must be initialized before using this function. See the Soft_UART_Init routine. Be aware that during transmission, software UART is incapable of receiving data – data transfer protocol must be set in such a way to prevent loss of information.

Example

```
dim some_byte as byte
...
some_byte = $0A
' Write a byte via Soft UART
Soft_UART_Write(some_byte)
```

Notes	The Software UART library implements time-based activities, so interrupts need to be disabled when using it.

Soft_UART_Break

Prototype	`sub procedure Soft_UART_Break()`
Description	Soft_UART_Read is blocking routine and it can block the program flow. Calling `Soft_UART_Break` routine from the interrupt will unblock the program execution. This mechanism is similar to WDT.
Parameters	None.
Returns	Nothing.
Requires	Nothing.

Example

```
dim data1, error, counter as byte

sub procedure Timer1Int() org IVT_ADDR_T1INTERRUPT
  counter = 0
  if (counter >= 20) then
    Soft_UART_Break()
    counter = 0                ' reset counter
  else
    Inc(counter)               ' increment counter
  end if
  T1IF_bit = 0                 ' Clear Timer1 overflow interrupt flag
end sub

main:

  ...

  if (Soft_UART_Init(PORTF, 2, 3, 14400, 0) = 0) then
    Soft_UART_Write(0x55)
  end if
  ...

  ' try Soft_UART_Read with blocking prevention mechanism
  IPC0   = IPC0 or 0x1000      ' Interrupt priority level = 1
  T1IE_bit = 1                 ' Enable Timer1 interrupts
  T1CON = 0x8030         ' Timer1 ON, internal clock FCY, prescaler 1:256
  data1 = Soft_UART_Read(&error)
  T1IE_bit = 0                 ' Disable Timer1 interrupts
end.
```

Notes	The Software UART library implements time-based activities, so interrupts need to be disabled when using it.

Library Example

This example demonstrates simple data exchange via software UART. If MCU is connected to the PC, you can test the example from the mikroBasic PRO for dsPIC30/33 and PIC24 USART communication terminal, launch it from the drop-down menu **Tools › USART Terminal** or simply click the USART Terminal Icon .

Copy Code To Clipboard

```
program Soft_UART

dim error_ as byte
    counter, byte_read as byte              ' Auxiliary variables

main:
  ADPCFG = 0xFFFF                           ' Configure AN pins as digital I/O

  TRISB = 0x00                              ' Set PORTB as output (error signalization)
  PORTB = 0                                 ' No error

  error_ = Soft_UART_Init(PORTF, 2, 3, 14400, 0) ' Initialize Soft UART at 14400 bps
  if (error_ > 0) then
    PORTB = error_                          ' Signalize Init error
    while TRUE
      nop                                   ' Stop program
    wend
  end if
  Delay_ms(100)

  for counter = “z” to “A” step-1           ' Send bytes from “z” downto “A”
    Soft_UART_Write(counter)
    Delay_ms(100)
  next counter

  while TRUE                                ' Endless loop
    byte_read = Soft_UART_Read(error_)      ' Read byte, then test error flag
    if (error_ <> 0)  then                  ' If error was detected
      PORTB = error_                        ' signal it on PORTB
    else
      Soft_UART_Write(byte_read)         ' If error was not detected, return byte read
    end if
  wend
end.
```

Sound Library

mikroBasic PRO for dsPIC30/33 and PIC24 provides a Sound Library to supply users with routines necessary for sound signalization in their applications. Sound generation needs additional hardware, such as piezo-speaker (example of piezo-speaker interface is given on the schematic at the bottom of this page).

Library Routines

- Sound_Init
- Sound_Play

Sound_Init

Prototype	`sub procedure Sound_Init(dim byref snd_port, snd_pin as word)`
Description	Configures the appropriate MCU pin for sound generation.
Parameters	- `snd_port`: sound output port address - `snd_pin`: sound output pin
Returns	Nothing.
Requires	Nothing.
Example	`' Initialize the pin RD3 for playing sound` `Sound_Init(PORTD, 3)`
Notes	None.

Sound_Play

Prototype	`sub procedure Sound_Play(dim freq_in_hz, duration_ms as word)`
Description	Generates the square wave signal on the appropriate pin.
Parameters	- `freq_in_hz`: signal frequency in Hertz (Hz) - `duration_ms`: signal duration in miliseconds (ms)
Returns	Nothing.
Requires	In order to hear the sound, you need a piezo speaker (or other hardware) on designated port. Also, you must call Sound_Init to prepare hardware for output before using this function.
Example	`' Play sound of 1KHz in duration of 100ms` `Sound_Play(1000, 100)`
Notes	None.

Library Example

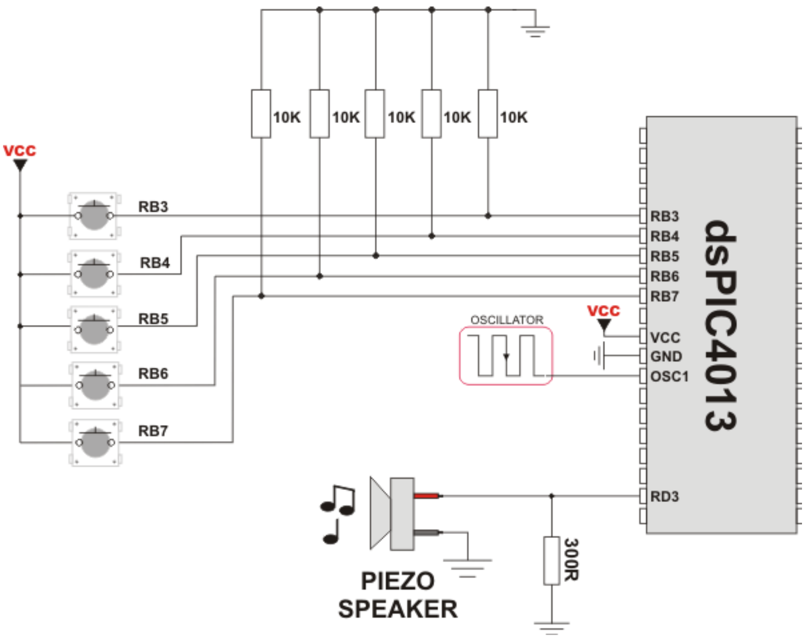
The example is a simple demonstration of how to use the Sound Library for playing tones on a piezo speaker.

Copy Code To Clipboard

```
program Sound

sub procedure Tone1()
  Sound_Play(659, 250)            ' Frequency = 659Hz, duration = 250ms
end sub

sub procedure Tone2()
  Sound_Play(698, 250)            ' Frequency = 698Hz, duration = 250ms
end sub

sub procedure Tone3()
  Sound_Play(784, 250)            ' Frequency = 784Hz, duration = 250ms
end sub

sub procedure Melody()            ' Plays the melody "Yellow house"
  Tone1() Tone2() Tone3() Tone3()
  Tone1() Tone2() Tone3() Tone3()
  Tone1() Tone2() Tone3()
  Tone1() Tone2() Tone3() Tone3()
  Tone1() Tone2() Tone3()
  Tone3() Tone3() Tone2() Tone2() Tone1()
end sub

sub procedure ToneA()             ' Tones used in Melody2 function
  Sound_Play( 880, 50)
end sub

sub procedure ToneC()
  Sound_Play(1046, 50)
end sub

sub procedure ToneE()
  Sound_Play(1318, 50)
end sub

sub procedure Melody2()                 ' Plays Melody2
dim counter as byte
  for counter = 9 to 1 step-1
    ToneA()
    ToneC()
    ToneE()
  next counter
end sub

main:
  ADPCFG = 0xFFFF                       ' Configure AN pins as digital I/O

  TRISB  = 0xF8                         ' Configure RB7..RB3 as input

  Sound_Init(PORTD, 3)
```

```
 Sound_Play(880, 1000)

  while TRUE                                 ' endless loop
    if (Button(PORTB,7,1,1)) then            ' If PORTB.7 is pressed play Tone1
      Tone1()
      while (RB7_bit <> 0)                   ' Wait for button to be released
        nop
      wend
    end if

    if (Button(PORTB,6,1,1)) then            ' If PORTB.6 is pressed play Tone1
      Tone2()
      while (RB6_bit <> 0)                   ' Wait for button to be released
        nop
      wend
    end if

    if (Button(PORTB,5,1,1)) then            ' If PORTB.5 is pressed play Tone1
      Tone3()
      while (RB5_bit <> 0)                   ' Wait for button to be released
        nop
      wend
    end if

    if (Button(PORTB,4,1,1)) then            ' If PORTB.4 is pressed play Tone1
      Melody2()
      while (RB4_bit <> 0)                   ' Wait for button to be released
        nop
      wend
    end if

    if (Button(PORTB,3,1,1)) then            ' If PORTB.3 is pressed play Tone1
      Melody()
      while (RB3_bit <> 0)                   ' Wait for button to be released
        nop
      wend
    end if
  wend
end.
```

HW Connection



Example of Sound Library

SPI Library

The SPI module is available with all dsPIC30/33 and PIC24 MCUs. mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for initializing the Slave mode and initializing and comfortable work with the Master mode. The dsPIC30/33 and PIC24 can easily communicate with other devices via SPI: A/D converters, D/A converters, MAX7219, LTC1290, etc.

Important:

- SPI library routines require you to specify the module you want to use. To select the desired SPI module, simply change the letter **x** in the routine prototype for a number from **1** to **3**.
- Number of SPI modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.
- Switching between the SPI modules in the SPI library is done by the SPI_Set_Active function (both SPI modules have to be previously initialized).

Library Routines

- SPIx_Init
- SPIx_Init_Advanced
- SPIx_Read
- SPIx_Write
- SPI_Set_Active

SPIx_Init

Prototype	`sub procedure SPIx_Init()`
Description	Configures and initializes the SPI module with default settings. Default settings: - Master mode - 8-bit data mode - secondary prescaler 1:1 - primary prescaler 64:1 - Slave Select disabled - input data sampled in the middle of interval - clock idle state low - Serial output data changes on transition from active clock state to idle clock state
Parameters	None.
Returns	Nothing.
Requires	MCU must have the SPI1 module.
Example	`' Initialize the SPI1 module with default settings` `SPI1_Init()`
Notes	SPI library routines require you to specify the module you want to use. To select the desired SPI module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of SPI modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library. Switching between the SPI modules in the SPI library is done by the SPI_Set_Active function (both SPI modules have to be previously initialized).

SPIx_Init_Advanced

Prototype

`sub procedure SPIx_Init_Advanced(dim master_mode, mode16, sec_prescaler, pri_prescaler, slave_select, data_sample, clock_idle, edge as word)`

Description

Configures and initializes the SPI module with user defined settings.

Parameters

Parameters `master_mode`, `mode16`, `sec_prescaler`, `pri_prescaler`, `slave_select`, `data_sample`, `clock_idle` and determine the working mode for SPI.

The `master_mode` parameter determines the working mode for SPI module.

Master/Slave mode	
Description	**Predefined library const**
`Master mode`	`_SPI_MASTER`
`Slave mode`	`_SPI_SLAVE`

The parameter `mode16` determines the data length mode, which can be 8-bits (per transmitions cycle) or 16-bits.

Data Length Mode	
Description	**Predefined library const**
`16-bit mode`	`_SPI_16_BIT`
`8-bit mode`	`_SPI_8_BIT`

The parameter `sec_prescaler` determines the value of the **secondary** SPI clock prescaler. Used only in the Master Mode.

Secondary SPI Clock Prescaler Value	
Description	**Predefined library const**
`Secondary Prescaler 1:1`	`_SPI_PRESCALE_SEC_1`
`Secondary Prescaler 1:2`	`_SPI_PRESCALE_SEC_2`
`Secondary Prescaler 1:3`	`_SPI_PRESCALE_SEC_3`
`Secondary Prescaler 1:4`	`_SPI_PRESCALE_SEC_4`
`Secondary Prescaler 1:5`	`_SPI_PRESCALE_SEC_5`
`Secondary Prescaler 1:6`	`_SPI_PRESCALE_SEC_6`
`Secondary Prescaler 1:7`	`_SPI_PRESCALE_SEC_7`
`Secondary Prescaler 1:8`	`_SPI_PRESCALE_SEC_8`

The parameter `pri_prescaler` determines the value of the **primary** SPI clock prescaler. Used only in the Master Mode.

Primary SPI Clock Prescaler Value	
Description	**Predefined library const**
`Primary Prescaler 1:1`	`_SPI_PRESCALE_PRI_1`
`Primary Prescaler 4:1`	`_SPI_PRESCALE_PRI_4`
`Primary Prescaler 16:1`	`_SPI_PRESCALE_PRI_16`
`Primary Prescaler 64:1`	`_SPI_PRESCALE_PRI_64`

Parameters

The parameter `slave_select` determines whether the Slave Select (SS) pin is used in communication. Valid in the Slave Mode only.

Slave Select Enable/Disable

Description	Predefined library const
`SS used for the Slave mode`	`_SPI_SS_ENABLE`
`SS not used for the Slave mode`	`_SPI_SS_DISABLE`

The parameter `data_sample` determines the sample moment (phase) of input data.

Data Sampling Moment

Description	Predefined library const
`Data sampled in the middle of data output time`	`_SPI_DATA_SAMPLE_MIDDLE`
`Data sampled at end of data output time`	`_SPI_DATA_SAMPLE_END`

The parameter `clock_idle` determines the behaviour of the SPI clock (CLK) line in IDLE phase.

Clock Polarity

Description	Predefined library const
`IDLE state is Lo, ACTIVE state is Hi`	`_SPI_CLK_IDLE_LOW`
`IDLE state is Hi, ACTIVE state is Lo`	`_SPI_CLK_IDLE_HIGH`

The parameter `edge` determines on which clock edge data is considered to be valid.

Clock Edge

Description	Predefined library const
`Data is valid on ACTIVE-to-IDLE transition`	`_SPI_ACTIVE_2_IDLE`
`Data is valid on IDLE-to-ACTIVE transition`	`_SPI_IDLE_2_ACTIVE`

Returns

Nothing.

Requires

MCU must have the SPI module.

Example

```
' Set SPI1 to the Master Mode, data length is 16-bit, clock = Fcy (no clock scaling), data sampled in the middle of interval, clock IDLE state high and data transmitted at low to high clock edge:
SPI1_Init_Advanced(_SPI_MASTER, _SPI_16_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_1, _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE)
```

Notes

SPI library routines require you to specify the module you want to use. To select the desired SPI module, simply change the letter x in the routine prototype for a number from 1 to 3.

Number of SPI modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

SPIx_Read

Prototype	`sub function SPIx_Read(dim data_out as word) as word`
Description	Reads one word or byte (depending on mode set by init routines) from the SPI bus.
Parameters	- `data_out`: dummy data for clock generation (see device Datasheet for SPI modules implementation details)
Returns	Received data.
Requires	Routine requires at least one SPI module. Used SPI module must be initialized before using this function. See the SPIx_Init and SPIx_Init_Advanced routines.

Example	`' read a byte from the SPI bus`
	`dim take, buffer as byte`
	`...`
	`take = SPI1_Read(buffer)`

Notes	SPI library routines require you to specify the module you want to use. To select the desired SPI module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of SPI modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

SPIx_Write

Prototype	`sub procedure SPIx_Write(dim data_out as word)`
Description	Writes one word or byte (depending on mode set by init routines) via the SPI bus.
Parameters	- `data_out`: data to be sent
Returns	Received data.
Requires	Routine requires at least one SPI module. Used SPI module must be initialized before using this function. See the SPIx_Init and SPIx_Init_Advanced routines.

Example	`' write a byte to the SPI bus`
	`dim buffer as byte`
	`...`
	`SPI1_Write(buffer)`

Notes	SPI library routines require you to specify the module you want to use. To select the desired SPI module, simply change the letter **x** in the routine prototype for a number from **1** to **3**. Number of SPI modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

SPI_Set_Active

Prototype	`sub procedure SPI_Set_Active(dim read_ptr as ^TSpi_Rd_Ptr, dim write_ptr as ^TSpi_Wr_Ptr)`
Description	Sets the active SPI module which will be used by the SPIx_Read and SPIx_Write routines.
Parameters	Parameters: - `read_ptr`: SPI1_Read handler - `write_ptr`: SPI1_Write handler
Returns	Nothing.
Requires	Routine is available only for MCUs with multiple SPI modules. Used SPI module must be initialized before using this function. See the SPIx_Init and SPIx_Init_Advanced routines.
Example	`SPI_Set_Active(@SPI1_Read, @SPI1_Write)  ' Sets the SPI1 module active`
Notes	Number of SPI modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

Library Example

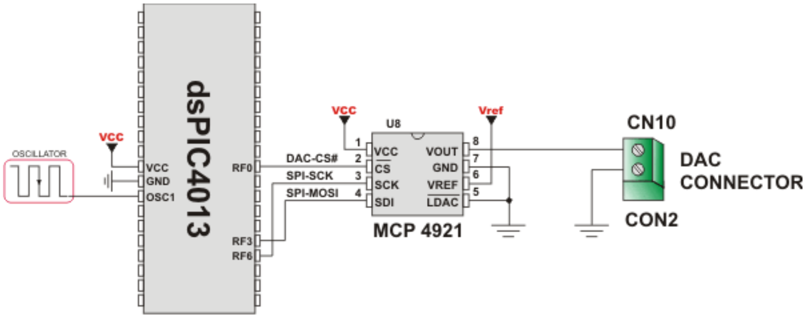
The code demonstrates how to use SPI library functions for communication between SPI2 module of the MCU and MCP4921 DAC chip.

Copy Code To Clipboard

```
program SPI

' DAC module connections
dim Chip_Select as sbit at LATF0_bit
    Chip_Select_Direction as sbit at TRISF0_bit
' End DAC module connections

dim value as word

sub procedure InitMain()
  TRISB0_bit = 1                          ' Set RB0 pin as input
  TRISB1_bit = 1                          ' Set RB1 pin as input
  Chip_Select = 1                         ' Deselect DAC
  Chip_Select_Direction = 0               ' Set CS# pin as Output
  SPI1_Init()                             ' Initialize SPI module
end sub

' DAC increments (0..4095) --> output voltage (0..Vref)
sub procedure DAC_Output(dim valueDAC as word)
dim temp as byte
  Chip_Select = 0                         ' Select DAC chip

' Send High Byte
  temp = word(valueDAC >> 8) and 0x0F     ' Store valueDAC[11..8] to temp[3..0]
  temp = temp or 0x30                     ' Define DAC setting, see MCP4921 datasheet
```

```
  SPI1_Write(temp)                         ' Send high byte via SPI

  ' Send Low Byte
  temp = valueDAC                          ' Store valueDAC[7..0] to temp[7..0]
  SPI1_Write(temp)                         ' Send low byte via SPI

  Chip_Select = 1                          ' Deselect DAC chip
end sub

main:
  ADPCFG = 0xFFFF                          ' Configure AN pins as digital

  InitMain()                               ' Perform main initialization

  value = 2048                             ' When program starts, DAC gives
                                           ' the output in the mid-range

  InitMain()                               ' Perform main initialization

  value = 2048                             ' When program starts, DAC gives
                                           '   the output in the mid-range

  while ( TRUE )                           ' Endless loop
    if ((RB0_bit) and (value < 4095)) then ' If RB0 button is pressed
      Inc(value)                           ' increment value
    else
      if ((RB1_bit) and (value > 0)) then  ' If RB1 button is pressed
        Dec(value)                         ' decrement value
      end if
    end if

    DAC_Output(value)                      ' Send value to DAC chip
    Delay_ms(1)                            ' Slow down key repeat pace
  wend
```

HW Connection



SPI HW connection

SPI Ethernet Library

The `ENC28J60` is a stand-alone Ethernet controller with an industry standard Serial Peripheral Interface (SPI). It is designed to serve as an Ethernet network interface for any controller equipped with SPI.

The `ENC28J60` meets all of the IEEE 802.3 specifications. It incorporates a number of packet filtering schemes to limit incoming packets. It also provides an internal DMA module for fast data throughput and hardware assisted IP checksum calculations. Communication with the host controller is implemented via two interrupt pins and the SPI, with data rates of up to 10 Mb/s. Two dedicated pins are used for LED link and network activity indication.

This library is designed to simplify handling of the underlying hardware (`ENC28J60`). It works with any dsPIC30/33 and PIC24 with integrated SPI and more than 4 Kb ROM memory. 38 to 40 MHz clock is recommended to get from 8 to 10 Mhz SPI clock, otherwise dsPIC30/33 and PIC24 should be clocked by `ENC28J60` clock output due to its silicon bug in SPI hardware. If you try lower dsPIC30/33 and PIC24 clock speed, there might be board hang or miss some requests.

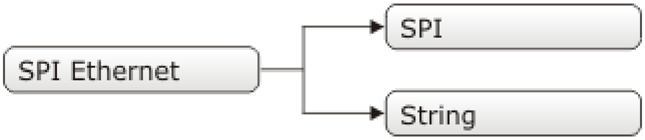
SPI Ethernet library supports:

- IPv4 protocol.
- ARP requests.
- ICMP echo requests.
- UDP requests.
- TCP requests (no stack, no packet reconstruction).
- ARP client with cache.
- DNS client.
- UDP client.
- DHCP client.
- packet fragmentation is **NOT** supported.

Important:

- Global library variable `SPI_Ethernet_userTimerSec` is used to keep track of time for all client implementations (ARP, DNS, UDP and DHCP). It is user responsibility to increment this variable each second in it’s code if any of the clients is used.
- For advanced users there is `__EthEnc28j60Private.mbas` unit in Uses folder of the compiler with description of all routines and global variables, relevant to the user, implemented in the SPI Ethernet Library.
- The appropriate hardware SPI module must be initialized before using any of the SPI Ethernet library routines. Refer to SPI Library.
- For MCUs with multiple SPI modules it is possible to initialize them and then switch by using the `SPI_Set_Active()` routine.

Library Dependency Tree



External dependencies of SPI Ethernet Library

The following variables must be defined in all projects using SPI Ethernet Library:	Description:	Example:
`dim SPI_Ethernet_CS as sbit sfr external`	ENC28J60 chip select pin.	`dim SPI_Ethernet_CS as sbit at RC1_bit`
`dim SPI_Ethernet_RST as sbit sfr external`	ENC28J60 reset pin.	`dim SPI_Ethernet_RST as sbit at RC0_bit`
`dim SPI_Ethernet_CS_Direction as sbit sfr external`	Direction of the ENC28J60 chip select pin.	`dim SPI_Ethernet_CS_Direction as sbit at TRISC1_bit`
`dim SPI_Ethernet_RST_Direction as sbit sfr external`	Direction of the ENC28J60 reset pin.	`dim SPI_Ethernet_RST_Direction as sbit at TRISC0_bit`
The following routines must be defined in all project using SPI Ethernet Library:	**Description:**	**Examples:**

`sub function SPI_Ethernet_UserTCP`
`(dim byref remoteHost as byte[4],`
`dim remotePort as word,`
`dim localPort as word,`
`dim reqLength as word,`
`dim byref flags as TEthPktFlags) as word`
	`TCP request handler.`	`Refer to the library example at the bottom of this page for code implementation.`

`sub function SPI_Ethernet_UserUDP(dim byref remoteHost as byte[4],`
`dim remotePort as word,`
`dim destPort as word,`
`dim reqLength as word,`
`dim byref flags as TEthPktFlags) as word`
	`UDP request handler.`	`Refer to the library example at the bottom of this page for code implementation.`

Library Routines

- SPI_Ethernet_Init
- SPI_Ethernet_Enable
- SPI_Ethernet_Disable
- SPI_Ethernet_doPacket
- SPI_Ethernet_putByte
- SPI_Ethernet_putBytes
- SPI_Ethernet_putString
- SPI_Ethernet_putConstString
- SPI_Ethernet_putConstBytes
- SPI_Ethernet_getByte
- SPI_Ethernet_getBytes
- SPI_Ethernet_UserTCP
- SPI_Ethernet_UserUDP
- SPI_Ethernet_setUserHandlers
- SPI_Ethernet_getIpAddress
- SPI_Ethernet_getGwIpAddress
- SPI_Ethernet_getDnsIpAddress
- SPI_Ethernet_getIpMask
- SPI_Ethernet_confNetwork
- SPI_Ethernet_arpResolve
- SPI_Ethernet_sendUDP

- SPI_Ethernet_dnsResolve
- SPI_Ethernet_initDHCP
- SPI_Ethernet_doDHCPLeaseTime
- SPI_Ethernet_renewDHCP

SPI_Ethernet_Init

Prototype	`sub procedure SPI_Ethernet_Init(dim mac as ^byte, dim ip as ^byte, dim fullDuplex as byte)`
Description	This is MAC module routine. It initializes `ENC28J60` controller. This function is internaly splited into 2 parts to help linker when coming short of memory. `ENC28J60` controller settings (parameters not mentioned here are set to default): - receive buffer start address : `0x0000`. - receive buffer end address : `0x19AD`. - transmit buffer start address: `0x19AE`. - transmit buffer end address : `0x1FFF`. - RAM buffer read/write pointers in auto-increment mode. - receive filters set to default: CRC + MAC Unicast + MAC Broadcast in OR mode. - flow control with TX and RX pause frames in full duplex mode. - frames are padded to `60` bytes + CRC. - maximum packet size is set to `1518`. - Back-to-Back Inter-Packet Gap: `0x15` in full duplex mode; `0x12` in half duplex mode. - Non-Back-to-Back Inter-Packet Gap: `0x0012` in full duplex mode; `0x0C12` in half duplex mode. - Collision window is set to `63` in half duplex mode to accomodate some `ENC28J60` revisions silicon bugs. - CLKOUT output is disabled to reduce EMI generation. - half duplex loopback disabled. - LED configuration: default (LEDA-link status, LEDB-link activity).
Parameters	- `mac`: RAM buffer containing valid MAC address. - `ip`: RAM buffer containing valid IP address. - `fullDuplex`: ethernet duplex mode switch. Valid values: `0` (half duplex mode) and `1` (full duplex mode).
Returns	Received data.
Requires	Global variables: - `SPI_Ethernet_CS`: Chip Select line - `SPI_Ethernet_CS_Direction`: Direction of the Chip Select pin - `SPI_Ethernet_RST`: Reset line - `SPI_Ethernet_RST_Direction`: Direction of the Reset pin must be defined before using this function. The SPI module needs to be initialized. See the SPIx_Init and SPIx_Init_Advanced routines.

Example

```
' mE ehternet NIC pinout
dim SPI_Ethernet_RST as sbit at LATF0_bit
dim SPI_Ethernet_CS as sbit at LATF1_bit
dim SPI_Ethernet_RST_Direction as sbit at TRISF0_bit
dim SPI_Ethernet_CS_Direction as sbit at TRISF1_bit
' end mE ehternet NIC pinout

const SPI_Ethernet_HALFDUPLEX = 0
const SPI_Ethernet_FULLDUPLEX = 1

  myMacAddr as byte[6]  ' my MAC address
  myIpAddr  as byte[4]  ' my IP addr
  ...
  myMacAddr[0] = 0x00
  myMacAddr[1] = 0x14
  myMacAddr[2] = 0xA5
  myMacAddr[3] = 0x76
  myMacAddr[4] = 0x19
  myMacAddr[5] = 0x3F

  myIpAddr[0]  = 192
  myIpAddr[1]  = 168
  myIpAddr[2]  = 20
  myIpAddr[3]  = 60

  SPI1_Init()
  SPI_Ethernet_Init(myMacAddr, myIpAddr, SPI_Ethernet_FULLDUPLEX)
```

Notes

None.

SPI_Ethernet_Enable

<table>
<tr><td>Prototype</td><td><code>sub procedure SPI_Ethernet_Enable(dim enFlt as byte)</code></td></tr>
<tr><td>Description</td><td><p>This is MAC module routine. This routine enables appropriate network traffic on the <code>ENC28J60</code> module by the means of it’s receive filters (unicast, multicast, broadcast, crc). Specific type of network traffic will be enabled if a corresponding bit of this routine’s input parameter is set. Therefore, more than one type of network traffic can be enabled at the same time. For this purpose, predefined library constants (see the table below) can be ORed to form appropriate input value.</p><p>Advanced filtering available in the <code>ENC28J60</code> module such as <code>Pattern Match</code>, <code>Magic Packet</code> and <code>Hash Table</code> can not be enabled by this routine. Additionally, all filters, except CRC, enabled with this routine will work in OR mode, which means that packet will be received if any of the enabled filters accepts it.</p><p>This routine will change receive filter configuration on-the-fly. It will not, in any way, mess with enabling/disabling receive/transmit logic or any other part of the <code>ENC28J60</code> module. The <code>ENC28J60</code> module should be properly cofigured by the means of SPI_Ethernet_Init routine.</p></td></tr>
<tr><td>Parameters</td><td><p>- <code>enFlt</code>: network traffic/receive filter flags. Each bit corresponds to the appropriate network traffic/receive filter:</p>
<table>
<tr><th>Bit</th><th>Mask</th><th>Description</th><th>Predefined library const</th></tr>
<tr><td>0</td><td><code>0x01</code></td><td>MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be enabled.</td><td><code>_SPI_Ethernet_BROADCAST</code></td></tr>
<tr><td>1</td><td><code>0x02</code></td><td>MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be enabled.</td><td><code>_SPI_Ethernet_MULTICAST</code></td></tr>
<tr><td>2</td><td><code>0x04</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>3</td><td><code>0x08</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>4</td><td><code>0x10</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>5</td><td><code>0x20</code></td><td>CRC check flag. When set, packets with invalid CRC field will be discarded.</td><td><code>_SPI_Ethernet_CRC</code></td></tr>
<tr><td>6</td><td><code>0x40</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>7</td><td><code>0x80</code></td><td>MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be enabled.</td><td><code>_SPI_Ethernet_UNICAST</code></td></tr>
</table></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Requires</td><td>Ethernet module has to be initialized. See SPI_Ethernet_Init.</td></tr>
<tr><td>Example</td><td><code>SPI_Ethernet_Enable(_SPI_Ethernet_CRC or _SPI_Ethernet_UNICAST) ' enable CRC checking and Unicast traffic</code></td></tr>
<tr><td>Notes</td><td><p>Advanced filtering available in the <code>ENC28J60</code> module such as <code>Pattern Match</code>, <code>Magic Packet</code> and <code>Hash Table</code> can not be enabled by this routine. Additionaly, all filters, except CRC, enabled with this routine will work in OR mode, which means that packet will be received if any of the enabled filters accepts it.</p><p>This routine will change receive filter configuration on-the-fly. It will not, in any way, mess with enabling/disabling receive/transmit logic or any other part of the <code>ENC28J60</code> module. The <code>ENC28J60</code> module should be properly cofigured by the means of SPI_Ethernet_Init routine.</p></td></tr>
</table>

SPI_Ethernet_Disable

Prototype

```
sub procedure SPI_Ethernet_Disable(dim disFlt as byte)
```

Description

This is MAC module routine. This routine disables appropriate network traffic on the `ENC28J60` module by the means of it's receive filters (unicast, multicast, broadcast, crc). Specific type of network traffic will be disabled if a corresponding bit of this routine's input parameter is set. Therefore, more than one type of network traffic can be disabled at the same time. For this purpose, predefined library constants (see the table below) can be ORed to form appropriate input value.

Parameters

- `disFlt`: network traffic/receive filter flags. Each bit corresponds to the appropriate network traffic/receive filter:

Bit	Mask	Description	Predefined library const
0	`0x01`	MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be disabled.	`_SPI_Ethernet_BROADCAST`
1	`0x02`	MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be disabled.	`_SPI_Ethernet_MULTICAST`
2	`0x04`	not used	`none`
3	`0x08`	not used	`none`
4	`0x10`	not used	`none`
5	`0x20`	CRC check flag. When set, CRC check will be disabled and packets with invalid CRC field will be accepted.	`_SPI_Ethernet_CRC`
6	`0x40`	not used	`none`
7	`0x80`	MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be disabled.	`_SPI_Ethernet_UNICAST`

Returns

Nothing.

Requires

Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
SPI_Ethernet_Disable(_SPI_Ethernet_CRC or _SPI_Ethernet_UNICAST)  ' disable CRC checking and Unicast traffic
```

Notes

Advanced filtering available in the `ENC28J60` module such as `Pattern Match`, `Magic Packet` and `Hash Table` can not be disabled by this routine.

This routine will change receive filter configuration on-the-fly. It will not, in any way, mess with enabling/disabling receive/transmit logic or any other part of the `ENC28J60` module. The `ENC28J60` module should be properly cofigured by the means of SPI_Ethernet_Init routine.

SPI_Ethernet_doPacket

Prototype	`sub function SPI_Ethernet_doPacket() as byte`
Description	This is MAC module routine. It processes next received packet if such exists. Packets are processed in the following manner: - ARP & ICMP requests are replied automatically. - upon TCP request the SPI_Ethernet_UserTCP function is called for further processing. - upon UDP request the SPI_Ethernet_UserUDP function is called for further processing.
Parameters	None.
Returns	- `0` - upon successful packet processing (zero packets received or received packet processed successfully). - `1` - upon reception error or receive buffer corruption. `ENC28J60` controller needs to be restarted. - `2` - received packet was not sent to us (not our IP, nor IP broadcast address). - `3` - received IP packet was not IPv4. - `4` - received packet was of type unknown to the library.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	see below
Notes	`SPI_Ethernet_doPacket` must be called as often as possible in user's code.

Example:

```
while TRUE
  ...
  SPI_Ethernet_doPacket()  ' process received packets
  ...
wend
```

SPI_Ethernet_putByte

Prototype	`sub procedure SPI_Ethernet_putByte(dim v as byte)`
Description	This is MAC module routine. It stores one byte to address pointed by the current `ENC28J60` write pointer (`EWRPT`).
Parameters	- `v`: value to store
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	see below
Notes	None.

Example:

```
dim data as byte
...
SPI_Ethernet_putByte(data)  ' put an byte into ENC28J60 buffer
```

SPI_Ethernet_putBytes

Prototype	`sub procedure SPI_Ethernet_putBytes(dim ptr as ^byte, dim n as word)`
Description	This is MAC module routine. It stores requested number of bytes into `ENC28J60` RAM starting from current `ENC28J60` write pointer (`EWRPT`) location.
Parameters	- `ptr`: RAM buffer containing bytes to be written into `ENC28J60` RAM. - `n`: number of bytes to be written.
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  buffer as byte[17]
  ...
  buffer = "mikroElektronika"
  ...
   SPI_Ethernet_putBytes(buffer, 16)  ' put an RAM array into ENC28J60 buffer
```

Notes	None.

SPI_Ethernet_putConstBytes

Prototype	`sub procedure SPI_Ethernet_putConstBytes(const ptr as ^byte, dim n as word)`
Description	This is MAC module routine. It stores requested number of const bytes into `ENC28J60` RAM starting from current `ENC28J60` write pointer (`EWRPT`) location.
Parameters	- `ptr`: const buffer containing bytes to be written into `ENC28J60` RAM. - `n`: number of bytes to be written.
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
const
  buffer as byte[17]
  ...
  buffer = "mikroElektronika"
  ...
  SPI_Ethernet_putConstBytes(buffer, 16)  ' put a const array into ENC28J60 buffer
```

Notes	None.

SPI_Ethernet_putString

Prototype	`sub function SPI_Ethernet_putString(dim ptr as ^byte) as word`
Description	This is MAC module routine. It stores whole string (excluding null termination) into `ENC28J60` RAM starting from current `ENC28J60` write pointer (`EWRPT`) location.
Parameters	- `ptr`: string to be written into `ENC28J60` RAM.
Returns	Number of bytes written into `ENC28J60` RAM.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  buffer as string[16]
  ...
  buffer = “mikroElektronika”
  ...
  SPI_Ethernet_putString(buffer)  ' put a RAM string into ENC28J60 buffer
```

Notes	None.

SPI_Ethernet_putConstString

Prototype	`sub function SPI_Ethernet_putConstString(const ptr as ^byte) as word`
Description	This is MAC module routine. It stores whole const string (excluding null termination) into `ENC28J60` RAM starting from current `ENC28J60` write pointer (`EWRPT`) location.
Parameters	- `ptr`: const string to be written into `ENC28J60` RAM.
Returns	Number of bytes written into `ENC28J60` RAM.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
const
  buffer as string[16]
  ...
  buffer = “mikroElektronika”
  ...
  SPI_Ethernet_putConstString(buffer)  ' put a const string into ENC28J60 buffer
```

Notes	None.

SPI_Ethernet_getByte

Prototype	`sub function SPI_Ethernet_getByte() as byte`
Description	This is MAC module routine. It fetches a byte from address pointed to by current `ENC28J60` read pointer (`ERDPT`).
Parameters	None.
Returns	Byte read from `ENC28J60` RAM.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim buffer as byte<>
...
buffer = SPI_Ethernet_getByte()  ' read a byte from ENC28J60 buffer
```

Notes	None.

SPI_Ethernet_getBytes

Prototype	`sub procedure SPI_Ethernet_getBytes(dim ptr as ^byte, dim addr as word, dim n as word)`
Description	This is MAC module routine. It fetches equested number of bytes from `ENC28J60` RAM starting from given address. If value of `0xFFFF` is passed as the address parameter, the reading will start from current `ENC28J60` read pointer (`ERDPT`) location.
Parameters	- `ptr`: buffer for storing bytes read from `ENC28J60` RAM. - `addr`: `ENC28J60` RAM start address. Valid values: `0..8192`. - `n`: number of bytes to be read.
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  buffer as byte[16]
  ...
  SPI_Ethernet_getBytes(buffer, 0x100, 16)  ' read 16 bytes, starting from 
address 0x100
```

Notes	None.

SPI_Ethernet_UserTCP

Prototype	`sub function SPI_Ethernet_UserTCP(dim remoteHost as ^byte, dim remotePort as word, dim localPort as word, dim reqLength as word, dim byref flags as TEthPktFlags) as word`
Description	This is TCP module routine. It is internally called by the library. The user accesses to the TCP request by using some of the SPI_Ethernet_get routines. The user puts data in the transmit buffer by using some of the SPI_Ethernet_put routines. The function must return the length in bytes of the TCP reply, or 0 if there is nothing to transmit. If there is no need to reply to the TCP requests, just define this function with return(0) as a single statement.
Parameters	- `remoteHost`: client's IP address. - `remotePort`: client's TCP port. - `localPort`: port to which the request is sent. - `reqLength`: TCP request data field length. - `flags`: structure consisted of two bit fields: Copy Code To Clipboard

```
structure TEthPktFlags
  dim canCloseTCP as boolean   ' flag which closes socket
   dim isBroadcast as boolean  ' flag which denotes that the IP package has 
been received via subnet broadcast address
end structure
```

Returns	- `0` - there should not be a reply to the request. - Length of TCP reply data field - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	This function is internally called by the library and should not be called by the user's code.
Notes	The function source code is provided with appropriate example projects. The code should be adjusted by the user to achieve desired reply.

SPI_Ethernet_UserUDP

Prototype	`sub function SPI_Ethernet_UserUDP(dim remoteHost as ^byte, dim remotePort as word, dim destPort as word, dim reqLength as word, dim byref flags as TEthPktFlags) as word`
Description	This is UDP module routine. It is internally called by the library. The user accesses to the UDP request by using some of the SPI_Ethernet_get routines. The user puts data in the transmit buffer by using some of the SPI_Ethernet_put routines. The function must return the length in bytes of the UDP reply, or 0 if nothing to transmit. If you don't need to reply to the UDP requests, just define this function with a return(0) as single statement.
Parameters	- `remoteHost`: client's IP address. - `remotePort`: client's port. - `destPort`: port to which the request is sent. - `reqLength`: UDP request data field length. - `flags`: structure consisted of two bit fields: Copy Code To Clipboard

```
structure TEthPktFlags
  dim canCloseTCP as boolean   ' flag which closes socket (not relevant to UDP)
   dim isBroadcast as boolean  ' flag which denotes that the IP package has been received via subnet broadcast address
end structure
```

Returns	- `0` - there should not be a reply to the request. - Length of UDP reply data field - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	This function is internally called by the library and should not be called by the user's code.
Notes	The function source code is provided with appropriate example projects. The code should be adjusted by the user to achieve desired reply.

SPI_Ethernet_setUserHandlers

Prototype	`sub procedure SPI_Ethernet_setUserHandlers(dim TCPHandler as ^TSPI_Ethernet_UserTCP, dim UDPHandler as ^TSPI_Ethernet_UserUDP)`
Description	Sets pointers to User TCP and UDP handler function implementations, which are automatically called by SPI Ethernet library.
Parameters	- `TCPHandler`: TCP request handler - `UDPHandler`: UDP request handler.
Returns	Nothing.
Requires	SPI_Ethernet_UserTCP and SPI_Ethernet_UserUDP have to be previously defined.
Example	`SPI_Ethernet_setUserHandlers(@SPI_Ethernet_UserTCP, @SPI_Ethernet_UserUDP)`
Notes	Since all libraries are built for SSA, SSA restrictions regarding function pointers dictate that modules that use SPI_Ethernet_setUserHandlers must also be built for SSA.

SPI_Ethernet_getIpAddress

Prototype	`sub function SPI_Ethernet_getIpAddress() as word`
Description	This routine should be used when DHCP server is present on the network to fetch assigned IP address.
Parameters	None.
Returns	Pointer to the global variable holding IP address.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  ipAddr as byte[4]   ' user IP address buffer
  ...
  memcpy(ipAddr, SPI_Ethernet_getIpAddress(), 4)   ' fetch IP address
```

Notes	User should always copy the IP address from the RAM location returned by this routine into it's own IP address buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_getGwIpAddress

Prototype	`sub function SPI_Ethernet_getGwIpAddress() as word`
Description	This routine should be used when DHCP server is present on the network to fetch assigned gateway IP address.
Parameters	None.
Returns	Pointer to the global variable holding DNS IP address.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  gwIpAddr as byte[4]   ' user gateway IP address buffer
  ...
   memcpy(gwIpAddr, SPI_Ethernet_getGwIpAddress(), 4)   ' fetch gateway IP
address
```

Notes	User should always copy the IP address from the RAM location returned by this routine into it's own gateway IP address buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_getDnsIpAddress

Prototype	`sub function SPI_Ethernet_getDnsIpAddress() as word`
Description	This routine should be used when DHCP server is present on the network to fetch assigned DNS IP address.
Parameters	None.
Returns	Pointer to the global variable holding DNS IP address.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  dnsIpAddr as byte[4]   ' user DNS IP address buffer
  ...
  memcpy(dnsIpAddr, SPI_Ethernet_getDnsIpAddress(), 4)   ' fetch DNS server
address
```

Notes	User should always copy the IP address from the RAM location returned by this routine into it's own DNS IP address buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_getIpMask

Prototype	`sub function SPI_Ethernet_getIpMask() as word`
Description	This routine should be used when DHCP server is present on the network to fetch assigned IP subnet mask.
Parameters	None.
Returns	Pointer to the global variable holding IP subnet mask.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init. Available for PIC18 family MCUs only.

Example

```
dim
  IpMask as byte[4]   ' user IP subnet mask buffer
  ...
  memcpy(IpMask, SPI_Ethernet_getIpMask(), 4)   ' fetch IP subnet mask
```

Notes	User should always copy the IP address from the RAM location returned by this routine into it's own IP subnet mask buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_confNetwork

Prototype	`sub procedure SPI_Ethernet_confNetwork(dim byref ipMask, gwIpAddr, dnsIpAddr as byte[4])`
Description	Configures network parameters (IP subnet mask, gateway IP address, DNS IP address) when DHCP is not used.
Parameters	- `ipMask`: IP subnet mask. - `gwIpAddr` gateway IP address. - `dnsIpAddr`: DNS IP address.
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  ipMask    as byte[4]   ' network mask (for example : 255.255.255.0)
  gwIpAddr  as byte[4]   ' gateway (router) IP address
  dnsIpAddr as byte[4]   ' DNS server IP address
  ...
  gwIpAddr[0]  = 192
  gwIpAddr[1]  = 168
  gwIpAddr[2]  = 20
  gwIpAddr[3]  = 6

  dnsIpAddr[0] = 192
  dnsIpAddr[1] = 168
  dnsIpAddr[2] = 20
  dnsIpAddr[3] = 100

  ipMask[0]    = 255
  ipMask[1]    = 255
  ipMask[2]    = 255
  ipMask[3]    = 0
  ...
   SPI_Ethernet_confNetwork(ipMask, gwIpAddr, dnsIpAddr) ' set network configuration parameters
```

Notes	The above mentioned network parameters should be set by this routine only if DHCP module is not used. Otherwise DHCP will override these settings.

SPI_Ethernet_arpResolve

Prototype	`sub function SPI_Ethernet_arpResolve(dim byref ip as byte[4], dim tmax as byte) as word`
Description	This is ARP module routine. It sends an ARP request for given IP address and waits for ARP reply. If the requested IP address was resolved, an ARP cash entry is used for storing the configuration. ARP cash can store up to 3 entries. For ARP cash structure refer to `“eth_enc28j60LibDef.mbas”` file in the compiler's Uses folder.
Parameters	- `ip`: IP address to be resolved. - `tmax`: time in seconds to wait for an reply.
Returns	- MAC address behind the IP address - the requested IP address was resolved. - `0` - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  IpAddr as byte[4]   ' IP address
  ...
  IpAddr[0] = 192
  IpAddr[0] = 168
  IpAddr[0] = 1
  IpAddr[0] = 1
  ...
  SPI_Ethernet_arpResolve(IpAddr, 5)  ' get MAC address behind the above IP
address, wait 5 secs for the response
```

Notes	The Ethernet services are not stopped while this routine waits for ARP reply. The incoming packets will be processed normaly during this time.

SPI_Ethernet_sendUDP

Prototype	`sub function SPI_Ethernet_sendUDP(dim byref destIP as byte[4], dim sourcePort, destPort as word, dim pkt as ^byte, dim pktLen as word) as byte`
Description	This is UDP module routine. It sends an UDP packet on the network.
Parameters	- `destIP`: remote host IP address. - `sourcePort`: local UDP source port number. - `destPort`: destination UDP port number. - `pkt`: packet to transmit. - `pktLen`: length in bytes of packet to transmit.
Returns	- `1` - UDP packet was sent successfully. - `0` - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  IpAddr as byte[4]   ' remote IP address
  ...
  IpAddr[0] = 192
  IpAddr[0] = 168
  IpAddr[0] = 1
  IpAddr[0] = 1
  ...
   SPI_Ethernet_sendUDP(IpAddr, 10001, 10001, “Hello”, 5)  ' send Hello
message to the above IP address, from UDP port 10001 to UDP port 10001
```

Notes	None.

SPI_Ethernet_dnsResolve

Prototype	`sub function SPI_Ethernet_dnsResolve(dim byref host as string, dim tmax as byte) as word`
Description	This is DNS module routine. It sends an DNS request for given host name and waits for DNS reply. If the requested host name was resolved, it's IP address is stored in library global variable and a pointer containing this address is returned by the routine. UDP port `53` is used as DNS port.
Parameters	- `host`: host name to be resolved. - `tmax`: time in seconds to wait for an reply.
Returns	- pointer to the location holding the IP address - the requested host name was resolved. - `0` - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
dim
  remoteHostIpAddr as string      ' user host IP address buffer
  ...
  ' SNTP server:
  ' Zurich, Switzerland: Integrated Systems Lab, Swiss Fed. Inst. of Technology
  ' 129.132.2.21: swisstime.ethz.ch
  ' Service Area: Switzerland and Europe
 memcpy(remoteHostIpAddr, SPI_Ethernet_dnsResolve(“swisstime.ethz.ch”, 5), 4)
```

Notes	The Ethernet services are not stopped while this routine waits for DNS reply. The incoming packets will be processed normaly during this time. User should always copy the IP address from the RAM location returned by this routine into it's own resolved host IP address buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_initDHCP

Prototype	`sub function SPI_Ethernet_initDHCP(dim tmax as byte) as byte`
Description	This is DHCP module routine. It sends an DHCP request for network parameters (IP, gateway, DNS addresses and IP subnet mask) and waits for DHCP reply. If the requested parameters were obtained successfully, their values are stored into the library global variables. These parameters can be fetched by using appropriate library IP get routines: - SPI_Ethernet_getIpAddress - fetch IP address. - SPI_Ethernet_getGwIpAddress - fetch gateway IP address. - SPI_Ethernet_getDnsIpAddress - fetch DNS IP address. - SPI_Ethernet_getIpMask - fetch IP subnet mask. UDP port `68` is used as DHCP client port and UDP port `67` is used as DHCP server port.
Parameters	- `tmax`: time in seconds to wait for an reply.
Returns	- `1` - network parameters were obtained successfully. - `0` - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
...
  SPI_Ethernet_initDHCP(5)  ' get network configuration from DHCP server, wait 5 sec for the response
  ...
```

Notes	The Ethernet services are not stopped while this routine waits for DNS reply. The incoming packets will be processed normaly during this time. When DHCP module is used, global library variable `SPI_Ethernet_userTimerSec` is used to keep track of time. It is user responsibility to increment this variable each second in it's code.

SPI_Ethernet_doDHCPLeaseTime

Prototype	`sub function SPI_Ethernet_doDHCPLeaseTime() as byte`
Description	This is DHCP module routine. It takes care of IP address lease time by decrementing the global lease time library counter. When this time expires, it's time to contact DHCP server and renew the lease.
Parameters	None.
Returns	- `0` - lease time has not expired yet. - `1` - lease time has expired, it's time to renew it.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.

Example

```
while true
  ...
  if(SPI_Ethernet_doDHCPLeaseTime() <> 0) then
    ...  ' it's time to renew the IP address lease
  end if
wend
```

Notes	None.

SPI_Ethernet_renewDHCP

Prototype	`sub function SPI_Ethernet_renewDHCP(dim tmax as byte) as byte`
Description	This is DHCP module routine. It sends IP address lease time renewal request to DHCP server.
Parameters	- `tmax`: time in seconds to wait for an reply.
Returns	- `1` - upon success (lease time was renewed). - `0` - otherwise (renewal request timed out).
Requires	Ethernet module has to be initialized. See SPI_Ethernet_Init.
Example	
Notes	None.

Example

```
while true
  ...
  if(SPI_Ethernet_doDHCPLeaseTime() <> 0) then
    SPI_Ethernet_renewDHCP(5)   ' it's time to renew the IP address lease,
with 5 secs for a reply
  end if
  ...
wend
```

Library Example

This code shows how to use the Ethernet mini library:

- the board will reply to ARP & ICMP echo requests
- the board will reply to UDP requests on any port:
  - returns the request in upper char with a header made of remote host IP & port number
- the board will reply to HTTP requests on port 80, GET method with pathnames:
  - / will return the HTML main page
  - /s will return board status as text string
  - /t0 ... /t7 will toggle RD0 to RD7 bit and return HTML main page
  - all other requests return also HTML main page.

Main program code:

```
program HTTP_Demo

' ***********************************
' * RAM variables
' *

' mE ehternet NIC pinout
dim
  SPI_Ethernet_Rst as sbit at LATF0_bit
  SPI_Ethernet_CS  as sbit at LATF1_bit
  SPI_Ethernet_Rst_Direction as sbit at TRISF0_bit
  SPI_Ethernet_CS_Direction  as sbit at TRISF1_bit
' end ethernet NIC definitions

dim myMacAddr   as byte[6]    ' my MAC address
    myIpAddr    as byte[4]    ' my IP address
```

```
    gwIpAddr     as byte[4]    ‘ gateway (router) IP address
    ipMask       as byte[4]    ‘ network mask (for example : 255.255.255.0)
    dnsIpAddr    as byte[4]    ‘ DNS server IP address

‘ ************************************************************
‘ * ROM constant strings
‘ *
const httpHeader as string[31]          = “HTTP/1.1 200 OK”+chr(10)+”Content-type: “
‘ HTTP header
const httpMimeTypeHTML as string[13]    = “text/html”+chr(10)+chr(10)  ‘ HTML MIME type
const httpMimeTypeScript as string[14] = “text/plain”+chr(10)+chr(10)  ‘ TEXT MIME type
const httpMethod as string[5]           = “GET /”
‘ *
‘ * web page, splited into 2 parts :
‘ * when coming short of ROM, fragmented data is handled more efficiently by linker
‘ *
‘  *  this  HTML  page  calls  the  boards  to  get  its  status,  and  builds  itself  with
javascript
‘ *
const indexPage as string[764] =
                    “<meta http-equiv=” + Chr(34) + “refresh”  + Chr(34) + “ content=”
                    + Chr(34) + “3;url=http://192.168.20.60”  + Chr(34) + “>” +
                    “<HTML><HEAD></HEAD><BODY>”+
                    “<h1>PIC + ENC28J60 Mini Web Server</h1>”+
                    “<a href=/>Reload</a>”+
                    “<script src=/s></script>”+
                    “<table><tr><td valign=top><table border=1 style=”+chr(34)+”font-
                    size:20px ;font-family: terminal ;”+chr(34)+”> “+
                    “<tr><th colspan=2>ADC</th></tr>”+
                 “<tr><td>AN0</td><td><script>document.write(AN0)</script></td></tr>”+
                 “<tr><td>AN1</td><td><script>document.write(AN1)</script></td></tr>”+
                    “</table></td><td><table border=1 style=”+chr(34)+”font-size:20px
                    ;font-family: terminal ;”+chr(34)+”> “+
                    “<tr><th colspan=2>PORTB</th></tr>”+
                    “<script>”+
                    “var str,i;”+
                    “str=”+chr(34)+chr(34)+”; “+
                    “for(i=2;i<10;i++)”+
            “{str+=”+chr(34)+”<tr><td bgcolor=pink>BUTTON #”+chr(34)+”+i+”+chr(34)+”</
       td>”+chr(34)+”; “+
                    “if(PORTB&(1<<i)){str+=”+chr(34)+”<td bgcolor=red>ON”+chr(34)+”;}”+
                    “else {str+=”+chr(34)+”<td bgcolor=#cccccc>OFF”+chr(34)+”;}”+
                    “str+=”+chr(34)+”</td></tr>”+chr(34)+”;}”+
                    “document.write(str) ;”+
                    “</script>”
const indexPage2 as string[470] =
                    “</table></td><td>”+
                    “<table border=1 style=”+chr(34)+”font-size:20px ;font-family:
                    terminal ;”+chr(34)+”> “+
                    “<tr><th colspan=3>PORTD</th></tr>”+
                    “<script>”+
                    “var str,i;”+
```

```
                   “str=”+chr(34)+chr(34)+”;  “+
                    “for(i=0;i<4;i++)”+
              “{str+=”+chr(34)+”<tr><td bgcolor=yellow>LED #”+chr(34)+”+i+”+chr(34)+”</
td>”+chr(34)+”;  “+
                    “if(PORTD&(1<<i)){str+=”+chr(34)+”<td bgcolor=red>ON”+chr(34)+”;}”+
                    “else {str+=”+chr(34)+”<td bgcolor=#cccccc>OFF”+chr(34)+”;}”+
                    “str+=”+chr(34)+”</td><td><a href=/t”+chr(34)+”+i+”+chr(34)+”>Toggl
e</a></td></tr>”+chr(34)+”;}”+
                    “document.write(str) ;”+
                    “</script>”+
                    “</table></td></tr></table>”+
            “This is HTTP request #<script>document.write(REQ)</script></BODY></HTML>”

dim    getRequest  as byte[15]    ' HTTP request buffer
       dyna        as char[30]    ' buffer for dynamic response
       httpCounter as word        ' counter of HTTP requests
       txt         as string[11]

' ******************************************
' * user defined functions
' *


' *
' * this function is called by the library
' * the user accesses to the HTTP request by successive calls to Spi_Ethernet_
getByte()
' * the user puts data in the transmit buffer by successive calls to Spi_Ethernet_
putByte()
' * the function must return the length in bytes of the HTTP reply, or 0 if nothing to
transmit
' *
' * if you don’t need to reply to HTTP requests,
' * just define this function with a return(0) as single statement
' *
' *
sub function Spi_Ethernet_UserTCP(dim byref remoteHost as byte[4],
 dim remotePort, localPort, reqLength as word, dim byref flags as TEthPktFlags) as word
  dim  i as word          ' my reply length
       bitMask as byte    ' for bit mask
       txt         as string[11]
    result = 0

    ' should we close tcp socket after response is sent?
    ' library closes tcp socket by default if canClose flag is not reset here
    '  canClose = 0 ' 0 - do not close socket
                    ' otherwise - close socket

if(localPort <> 80) then          ' I listen only to web request on port 80
      result = 0
      exit
    end if
```

```
    ' get 10 first bytes only of the request, the rest does not matter here
    for i = 0 to 10
      getRequest[i] = Spi_Ethernet_getByte()
    next i

    getRequest[i] = 0

    ' copy httpMethod to ram for use in memcmp routine
    for i = 0 to 4
      txt[i] = httpMethod[i]
    next i

    if(memcmp(@getRequest, @txt, 5) <> 0) then   ' only GET method is supported here
      result = 0
      exit
    end if

    Inc(httpCounter)                            ' one more request done

    if(getRequest[5] = "s") then                ' if request path name starts with s, store
dynamic data in transmit buffer
           ' the text string replied by this request can be interpreted as javascript
statements
      ' by browsers

      result = SPI_Ethernet_putConstString(@httpHeader)                 ' HTTP header
      result = result + SPI_Ethernet_putConstString(@httpMimeTypeScript)    ' with text
MIME type

      ' add AN0 value to reply
      WordToStr(ADC1_Get_Sample(0), dyna)
      txt   = "var AN0="
      result = result + Spi_Ethernet_putString(@txt)
      result = result + Spi_Ethernet_putString(@dyna)
      txt   = ";"
      result = result + Spi_Ethernet_putString(@txt)

      ' add AN1 value to reply
      WordToStr(ADC1_Get_Sample(1), dyna)
      txt   = "var AN1="
      result = result + Spi_Ethernet_putString(@txt)
      result = result + Spi_Ethernet_putString(@dyna)
      txt   = ";"
      result = result + Spi_Ethernet_putString(@txt)

      ' add PORTB value (buttons) to reply
      txt   = "var PORTB="
      result = result + Spi_Ethernet_putString(@txt)
      WordToStr(PORTB, dyna)
      result = result + Spi_Ethernet_putString(@dyna)
      txt   = ";"
      result = result + Spi_Ethernet_putString(@txt)
```

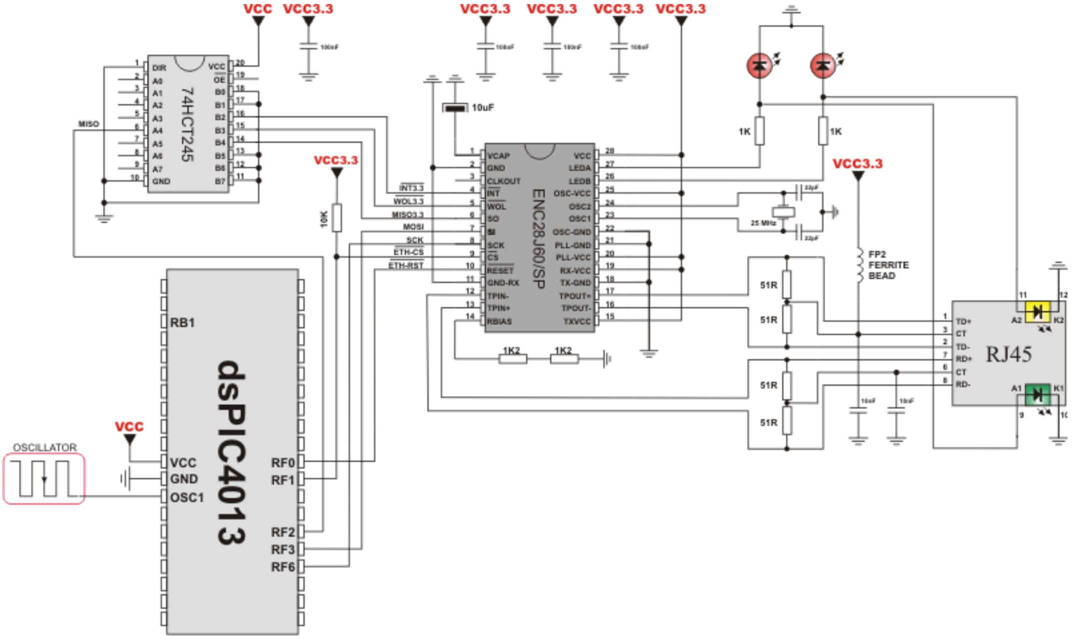
```
     ' add PORTD value (LEDs) to reply
      txt  = “var PORTD=”
      result = result + Spi_Ethernet_putString(@txt)
      WordToStr(PORTD, dyna)
      result = result + Spi_Ethernet_putString(@dyna)
      txt  = “;”
      result = result + Spi_Ethernet_putString(@txt)

      ' add HTTP requests counter to reply
      WordToStr(httpCounter, dyna)
      txt  = “var REQ=”
      result = result + Spi_Ethernet_putString(@txt)
      result = result + Spi_Ethernet_putString(@dyna)
      txt  = “;”
      result = result + Spi_Ethernet_putString(@txt)
    else
     if(getRequest[5] = “t”) then     ' if request path name starts with t, toggle PORTD
(LED) bit number that comes after
        bitMask = 0
       if(isdigit(getRequest[6]) <> 0) then   ' if 0 <= bit number <= 9, bits 8 & 9 does
not exist but does not matter
          bitMask = getRequest[6] - “0”                   ' convert ASCII to integer
          bitMask = 1 << bitMask                          ' create bit mask
          PORTD   = PORTD xor bitMask                  ' toggle PORTD with xor operator
        end if
      end if
    end if

    if(result = 0) then ' what do to by default
      result = SPI_Ethernet_putConstString(@httpHeader)             ' HTTP header
       result = result + SPI_Ethernet_putConstString(@httpMimeTypeHTML)    ' with HTML
MIME type
      result = result + SPI_Ethernet_putConstString(@indexPage)  ' HTML page first part
       result = result + SPI_Ethernet_putConstString(@indexPage2)   ' HTML page second
part
    end if

    ' return to the library with the number of bytes to transmit
end sub

' *
' * this function is called by the library
' * the user accesses to the UDP request by successive calls to Spi_Ethernet_getByte()
' * the user puts data in the transmit buffer by successive calls to Spi_Ethernet_putByte()
' * the function must return the length in bytes of the UDP reply, or 0 if nothing to transmit
' *
' * if you don’t need to reply to UDP requests,
' * just define this function with a return(0) as single statement
' *
' *
sub function Spi_Ethernet_UserUDP(dim byref remoteHost as byte[4],
                                dim remotePort, destPort, reqLength as word, dim byref
flags as TEthPktFlags) as word
```

```
  dim txt as string[5]
   result = 0
   ' reply is made of the remote host IP address in human readable format
   byteToStr(remoteHost[0], dyna)                ' first IP address byte
   dyna[3] = "."
   byteToStr(remoteHost[1], txt)                 ' second
   dyna[4] = txt[0]
   dyna[5] = txt[1]
   dyna[6] = txt[2]
   dyna[7] = "."
   byteToStr(remoteHost[2], txt)                 ' second
   dyna[8]  = txt[0]
   dyna[9]  = txt[1]
   dyna[10] = txt[2]

   dyna[11] = "."
   byteToStr(remoteHost[3], txt)                 ' second
   dyna[12] = txt[0]
   dyna[13] = txt[1]
   dyna[14] = txt[2]

   dyna[15] = ":"                                ' add separator

   ' then remote host port number
   WordToStr(remotePort, txt)
   dyna[16] = txt[0]
   dyna[17] = txt[1]
   dyna[18] = txt[2]
   dyna[19] = txt[3]
   dyna[20] = txt[4]
   dyna[21] = "["
   WordToStr(destPort, txt)
   dyna[22] = txt[0]
   dyna[23] = txt[1]
   dyna[24] = txt[2]
   dyna[25] = txt[3]
   dyna[26] = txt[4]
   dyna[27] = "]"
   dyna[28] = 0

   ' the total length of the request is the length of the dynamic string plus the
   text of the request
   result = 28 + reqLength

   ' puts the dynamic string into the transmit buffer
   Spi_Ethernet_putBytes(@dyna, 28)

   ' then puts the request string converted into upper char into the transmit buffer
   while(reqLength <> 0)
     Spi_Ethernet_putByte(Spi_Ethernet_getByte())
     reqLength = reqLength - 1
   wend

   ' back to the library with the length of the UDP reply
end sub
```

```
main:
  ADPCFG = 0xFFFD          ' Set AN pins as digital I/O except AN0 and AN1

  PORTB = 0
  TRISB = 0xFFFF           ' set PORTB as input for buttons and adc

  PORTD = 0
  TRISD = 0                ' set PORTD as output

  ADC1_Init()

  httpCounter = 0

   ' set mac address
  myMacAddr[0] = 0x00
  myMacAddr[1] = 0x14
  myMacAddr[2] = 0xA5
  myMacAddr[3] = 0x76
  myMacAddr[4] = 0x19
  myMacAddr[5] = 0x3F

   ' set IP address
  myIpAddr[0] = 192
  myIpAddr[1] = 168
  myIpAddr[2] = 20
  myIpAddr[3] = 60

   ' set gateway address
  gwIpAddr[0]  = 192
  gwIpAddr[1]  = 168
  gwIpAddr[2]  = 20
  gwIpAddr[3]  = 6

   ' set dns address
  dnsIpAddr[0] = 192
  dnsIpAddr[1] = 168
  dnsIpAddr[2] = 20
  dnsIpAddr[3] = 1

   ' set subnet mask
  ipMask[0]    = 255
  ipMask[1]    = 255
  ipMask[2]    = 255
  ipMask[3]    = 0

'   *
'   * starts ENC28J60 with :
'   * reset bit on PORTC.B0
'   * CS bit on PORTC.B1
'   * my MAC & IP address
'   * full duplex
'   *
```

```
SPI1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1, _SPI_PRESCALE_PRI_4,
                         _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_
IDLE_2_ACTIVE)
    SPI_Ethernet_Init(myMacAddr, myIpAddr, _SPI_Ethernet_FULLDUPLEX)  ' init ethernet
module
  SPI_Ethernet_setUserHandlers(@SPI_Ethernet_UserTCP, @SPI_Ethernet_UserUDP)  ' set user
handlers

   ' dhcp will not be used here, so use preconfigured addresses
  SPI_Ethernet_confNetwork(ipMask, gwIpAddr, dnsIpAddr)

  while TRUE                          ' do forever
    SPI_Ethernet_doPacket()           ' process incoming Ethernet packets

'      *
'      * add your stuff here if needed
'      * SPI_Ethernet_doPacket() must be called as often as possible
'      * otherwise packets could be lost
'      *
  wend
end.
```

HW Connection



SPI Ethernet ENC24J600 Library

The `ENC24J600` is a stand-alone Ethernet controller with an industry standard Serial Peripheral Interface (SPI). It is designed to serve as an Ethernet network interface for any controller equipped with SPI.

The `ENC24J600` meets all of the IEEE 802.3 specifications applicable to 10Base-T and 100Base-TX Ethernet. It incorporates a number of packet filtering schemes to limit incoming packets. It also provides an internal, 16-bit wide DMA module for fast data throughput and hardware assisted IP checksum calculations. Communication with the host controller is implemented via two interrupt pins and the SPI, with data rates of 10/100 Mb/s. Two dedicated pins are used for LED link and network activity indication.

This library is designed to simplify handling of the underlying hardware (`ENC24J600`). It works with any dsPIC30/33 and PIC24 with integrated SPI and more than 4 Kb ROM memory. 38 to 40 MHz clock is recommended to get from 8 to 10 Mhz SPI clock, otherwise dsPIC30/33 and PIC24 should be clocked by `ENC24J600` clock output due to its silicon bug in SPI hardware. If you try lower dsPIC30/33 and PIC24 clock speed, there might be board hang or miss some requests.

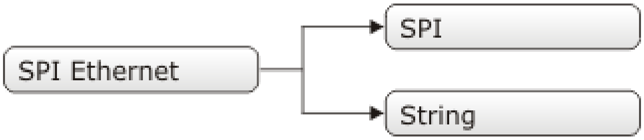
SPI Ethernet ENC24J600 library supports:

- IPv4 protocol.
- ARP requests.
- ICMP echo requests.
- UDP requests.
- TCP requests (no stack, no packet reconstruction).
- ARP client with cache.
- DNS client.
- UDP client.
- DHCP client.
- packet fragmentation is **NOT** supported.

Important:

- Global library variable `SPI_Ethernet_24j600_userTimerSec` is used to keep track of time for all client implementations (ARP, DNS, UDP and DHCP). It is user responsibility to increment this variable each second in it's code if any of the clients is used.
- For advanced users there is `__EthEnc24j600Private.mbas` unit in Uses folder of the compiler with description of all routines and global variables, relevant to the user, implemented in the SPI Ethernet ENC24J600 Library.
- The appropriate hardware SPI module must be initialized before using any of the SPI Ethernet ENC24J600 library routines. Refer to SPI Library.
- For MCUs with multiple SPI modules it is possible to initialize them and then switch by using the `SPI_Set_Active()` routine.

Library Dependency Tree



External dependencies of SPI Ethernet ENC24J600 Library

The following variables must be defined in all projects using SPI Ethernet ENC24J600 Library:	Description:	Example:
`dim SPI_Ethernet_24j600_CS as sbit sfr external`	ENC24J600 chip select pin.	`dim SPI_Ethernet_24j600_CS as sbit at RC1_bit`
`dim SPI_Ethernet_24j600_CS_Direction as sbit sfr external`	Direction of the ENC24J600 chip select pin.	`dim SPI_Ethernet_24j600_CS_Direction as sbit at TRISC1_bit`

The following routines must be defined in all project using SPI Ethernet ENC24J600 Library:	Description:	Example:

`sub function SPI_Ethernet_24j600_UserTCP(dim byref remoteHost as byte[4],`
`dim remotePort as word,`
`dim localPort as word,`
`dim reqLength as word,`
`dim byref flags as TEthj600PktFlags) as word`
	`TCP request handler.`	`Refer to the library example at the bottom of this page for code implementation.`

`sub function SPI_Ethernet_24j600_UserUDP(dim byref remoteHost as byte[4],`
`dim remotePort as word,`
`dim destPort as word,`
`dim reqLength as word,`
`dim byref flags as TEthJ600PktFlags) as word`
	`UDP request handler.`	`Refer to the library example at the bottom of this page for code implementation.`

Library Routines

- SPI_Ethernet_24j600_Init
- SPI_Ethernet_24j600_Enable
- SPI_Ethernet_24j600_Disable
- SPI_Ethernet_24j600_doPacket
- SPI_Ethernet_24j600_putByte
- SPI_Ethernet_24j600_putBytes
- SPI_Ethernet_24j600_putString
- SPI_Ethernet_24j600_putConstString
- SPI_Ethernet_24j600_putConstBytes
- SPI_Ethernet_24j600_getByte
- SPI_Ethernet_24j600_getBytes
- SPI_Ethernet_24j600_UserTCP
- SPI_Ethernet_24j600_UserUDP
- SPI_Ethernet_24j600_setUserHandlers
- SPI_Ethernet_24j600_getIpAddress
- SPI_Ethernet_24j600_getGwIpAddress
- SPI_Ethernet_24j600_getDnsIpAddress
- SPI_Ethernet_24j600_getIpMask
- SPI_Ethernet_24j600_confNetwork
- SPI_Ethernet_24j600_arpResolve
- SPI_Ethernet_24j600_sendUDP
- SPI_Ethernet_24j600_dnsResolve
- SPI_Ethernet_24j600_initDHCP
- SPI_Ethernet_24j600_doDHCPLeaseTime
- SPI_Ethernet_24j600_renewDHCP

SPI_Ethernet_24j600_Init

<table>
<tr><td>Prototype</td><td><code>sub procedure SPI_Ethernet_24j600_Init(dim mac as ^byte, dim ip as ^byte, dim fullDuplex as byte)</code></td></tr>
<tr><td>Description</td><td>This is MAC module routine. It initializes <code>ENC24J600</code> controller. This function is internally splited into 2 parts to help linker when coming short of memory.

<code>ENC24J600</code> controller settings (parameters not mentioned here are set to default):

- receive buffer start address : <code>0x0000</code>.
- receive buffer end address : <code>0x19AD</code>.
- transmit buffer start address: <code>0x19AE</code>.
- transmit buffer end address : <code>0x1FFF</code>.
- RAM buffer read/write pointers in auto-increment mode.
- receive filters set to default: CRC + MAC Unicast + MAC Broadcast in OR mode.
- flow control with TX and RX pause frames in full duplex mode.
- frames are padded to <code>60</code> bytes + CRC.
- maximum packet size is set to <code>1518</code>.
- Back-to-Back Inter-Packet Gap: <code>0x15</code> in full duplex mode; <code>0x12</code> in half duplex mode.
- Non-Back-to-Back Inter-Packet Gap: <code>0x0012</code> in full duplex mode; <code>0x0C12</code> in half duplex mode.
- Collision window is set to <code>63</code> in half duplex mode to accomodate some <code>ENC24J600</code> revisions silicon bugs.
- CLKOUT output is disabled to reduce EMI generation.
- half duplex loopback disabled.
- LED configuration: default (LEDA-link status, LEDB-link activity).</td></tr>
<tr><td>Parameters</td><td>- <code>mac</code>: RAM buffer containing valid MAC address.
- <code>ip</code>: RAM buffer containing valid IP address.
- <code>configuration</code>: ethernet negotiation, duplex and speed mode settings. For this purpose, predefined library constants (see the list below) can be combined using logical AND to form appropriate value:

<table>
<tr><th>Description:</th><th>Predefined library const</th></tr>
<tr><td>Set Auto-negotiation</td><td><code>SPI_Ethernet_24j600_AUTO_NEGOTIATION</code></td></tr>
<tr><td>Set manual negotiation.</td><td><code>SPI_Ethernet_24j600_MANUAL_NEGOTIATION</code></td></tr>
<tr><td>Set Half duplex Mode</td><td><code>SPI_Ethernet_24j600_HALFDUPLEX</code></td></tr>
<tr><td>Set Full duplex Mode</td><td><code>SPI_Ethernet_24j600_FULLDUPLEX</code></td></tr>
<tr><td>Set transmission speed of 10Mbps</td><td><code>SPI_Ethernet_24j600_SPD10</code></td></tr>
<tr><td>Set transmission speed of 100Mbps</td><td><code>SPI_Ethernet_24j600_SPD100</code></td></tr>
</table>

Note:

- It is advisable to use only the Auto-negotiation setting. If manual negotiation is used, then duplex and speed mode setting must be set also.
- Duplex and speed mode may be set only when using manual negotiation.</td></tr>
</table>

Returns	Nothing.
Requires	Global variables: - `SPI_Ethernet_24j600_CS:` Chip Select line - `SPI_Ethernet_24j600_CS_Direction:` Direction of the Chip Select pin - `SPI_Ethernet_24j600_RST:` Reset line - `SPI_Ethernet_24j600_RST_Direction:` Direction of the Reset pin must be defined before using this function. The SPI module needs to be initialized. See the SPIx_Init and SPIx_Init_Advanced routines.

Example

```
' mE ehternet NIC pinout
dim SPI_Ethernet_24j600_CS as sbit at LATF1_bit
dim SPI_Ethernet_24j600_CS_Direction as sbit at TRISF1_bit
' end mE ehternet NIC pinout

  myMacAddr as byte[6]  ' my MAC address
  myIpAddr  as byte[4]  ' my IP addr
  ...
  myMacAddr[0] = 0x00
  myMacAddr[1] = 0x14
  myMacAddr[2] = 0xA5
  myMacAddr[3] = 0x76
  myMacAddr[4] = 0x19
  myMacAddr[5] = 0x3F

  myIpAddr[0]  = 192
  myIpAddr[1]  = 168
  myIpAddr[2]  = 20
  myIpAddr[3]  = 60

  SPI1_Init()
  SPI_Ethernet_24j600_Init(myMacAddr, myIpAddr, SPI_Ethernet_24j600_MANUAL_
NEGOTIATION  and  SPI_Ethernet_24j600_FULLDUPLEX  and  SPI_Ethernet_24j600_
SPD100)
```

Notes	None.

SPI_Ethernet_24j600_Enable

<table>
<tr><td>Prototype</td><td><code>sub procedure SPI_Ethernet_24j600_Enable(dim enFlt as word)</code></td></tr>
<tr><td>Description</td><td>This is MAC module routine. This routine enables appropriate network traffic on the <code>ENC24J600</code> module by the means of it’s receive filters (unicast, multicast, broadcast, crc). Specific type of network traffic will be enabled if a corresponding bit of this routine’s input parameter is set. Therefore, more than one type of network traffic can be enabled at the same time. For this purpose, predefined library constants (see the table below) can be ORed to form appropriate input value.

Advanced filtering available in the <code>ENC24J600</code> module such as <code>Pattern Match</code>, <code>Magic Packet</code> and <code>Hash Table</code> can not be enabled by this routine. Additionaly, all filters, except CRC, enabled with this routine will work in OR mode, which means that packet will be received if any of the enabled filters accepts it.

This routine will change receive filter configuration on-the-fly. It will not, in any way, mess with enabling/disabling receive/transmit logic or any other part of the <code>ENC24J600</code> module. The <code>ENC24J600</code> module should be properly cofigured by the means of SPI_Ethernet_24j600_Init routine.</td></tr>
<tr><td>Parameters</td><td>- <code>enFlt</code>: network traffic/receive filter flags. Each bit corresponds to the appropriate network traffic/receive filter:
<table>
<tr><th>Bit</th><th>Mask</th><th>Description</th><th>Predefined library const</th></tr>
<tr><td>0</td><td><code>0x01</code></td><td>MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be enabled.</td><td><code>_SPI_Ethernet_24j600_BROADCAST</code></td></tr>
<tr><td>1</td><td><code>0x02</code></td><td>MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be enabled.</td><td><code>_SPI_Ethernet_24j600_MULTICAST</code></td></tr>
<tr><td>2</td><td><code>0x04</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>3</td><td><code>0x08</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>4</td><td><code>0x10</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>5</td><td><code>0x20</code></td><td>CRC check flag. When set, packets with invalid CRC field will be discarded.</td><td><code>_SPI_Ethernet_24j600_CRC</code></td></tr>
<tr><td>6</td><td><code>0x40</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>7</td><td><code>0x80</code></td><td>MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be enabled.</td><td><code>_SPI_Ethernet_24j600_UNICAST</code></td></tr>
</table>
</td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Requires</td><td>Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.</td></tr>
<tr><td>Example</td><td><code>SPI_Ethernet_24j600_Enable(_SPI_Ethernet_24j600_CRC or _SPI_Ethernet_24j600_UNICAST) ' enable CRC checking and Unicast traffic</code></td></tr>
<tr><td>Notes</td><td>Advanced filtering available in the <code>ENC24J600</code> module such as <code>Pattern Match</code>, <code>Magic Packet</code> and <code>Hash Table</code> can not be enabled by this routine. Additionaly, all filters, except CRC, enabled with this routine will work in OR mode, which means that packet will be received if any of the enabled filters accepts it.

This routine will change receive filter configuration on-the-fly. It will not, in any way, mess with enabling/disabling receive/transmit logic or any other part of the <code>ENC24J600</code> module. The <code>ENC24J600</code> module should be properly cofigured by the means of SPI_Ethernet_24j600_Init routine.</td></tr>
</table>

SPI_Ethernet_24j600_Disable

<table>
<tr><td>Prototype</td><td><code>sub procedure SPI_Ethernet_24j600_Disable(dim disFlt as word)</code></td></tr>
<tr><td>Description</td><td>This is MAC module routine. This routine disables appropriate network traffic on the <code>ENC24J600</code> module by the means of it’s receive filters (unicast, multicast, broadcast, crc). Specific type of network traffic will be disabled if a corresponding bit of this routine’s input parameter is set. Therefore, more than one type of network traffic can be disabled at the same time. For this purpose, predefined library constants (see the table below) can be ORed to form appropriate input value.</td></tr>
<tr><td>Parameters</td><td>- <code>disFlt</code>: network traffic/receive filter flags. Each bit corresponds to the appropriate network traffic/receive filter:
<table>
<tr><th>Bit</th><th>Mask</th><th>Description</th><th>Predefined library const</th></tr>
<tr><td>0</td><td><code>0x01</code></td><td>MAC Broadcast traffic/receive filter flag. When set, MAC broadcast traffic will be disabled.</td><td><code>_SPI_Ethernet_24j600_BROADCAST</code></td></tr>
<tr><td>1</td><td><code>0x02</code></td><td>MAC Multicast traffic/receive filter flag. When set, MAC multicast traffic will be disabled.</td><td><code>_SPI_Ethernet_24j600_MULTICAST</code></td></tr>
<tr><td>2</td><td><code>0x04</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>3</td><td><code>0x08</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>4</td><td><code>0x10</code></td><td>not used</td><td><code>none</code></td></tr>
<tr><td>5</td><td><code>0x20</code></td><td>CRC check flag. When set, CRC check will be disabled and packets with invalid CRC field will be accepted.</td><td><code>_SPI_Ethernet_24j600_CRC</code></td></tr>
<tr><td>6</td><td><code>0x40</code></td><td>not used</td><td>none</td></tr>
<tr><td>7</td><td><code>0x80</code></td><td>MAC Unicast traffic/receive filter flag. When set, MAC unicast traffic will be disabled.</td><td><code>_SPI_Ethernet_24j600_UNICAST</code></td></tr>
</table>
</td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Requires</td><td>Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.</td></tr>
<tr><td>Example</td><td><code>SPI_Ethernet_24j600_Disable(_SPI_Ethernet_24j600_CRC or _SPI_Ethernet_24j600_UNICAST) ' disable CRC checking and Unicast traffic</code></td></tr>
<tr><td>Notes</td><td>Advanced filtering available in the <code>ENC24J600</code> module such as <code>Pattern Match, Magic Packet</code> and <code>Hash Table</code> can not be disabled by this routine.
This routine will change receive filter configuration on-the-fly. It will not, in any way, mess with enabling/disabling receive/transmit logic or any other part of the <code>ENC24J600</code> module. The <code>ENC24J600</code> module should be properly cofigured by the means of SPI_Ethernet_24j600_Init routine.
The <code>ENC24J600</code> module should be properly cofigured by the means of SPI_Ethernet_24j600_Init routine.</td></tr>
</table>

SPI_Ethernet_24j600_doPacket

Prototype	`sub function SPI_Ethernet_24j600_doPacket() as byte`
Description	This is MAC module routine. It processes next received packet if such exists. Packets are processed in the following manner: - ARP & ICMP requests are replied automatically. - upon TCP request the SPI_Ethernet_24j600_UserTCP function is called for further processing. - upon UDP request the SPI_Ethernet_24j600_UserUDP function is called for further processing.
Parameters	None.
Returns	- `0` - upon successful packet processing (zero packets received or received packet processed successfully). - `1` - upon reception error or receive buffer corruption. `ENC24J600` controller needs to be restarted. - `2` - received packet was not sent to us (not our IP, nor IP broadcast address). - `3` - received IP packet was not IPv4. - `4` - received packet was of type unknown to the library.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
while TRUE
  ...
  SPI_Ethernet_24j600_doPacket()  ' process received packets
  ...
wend
```

Notes `SPI_Ethernet_24j600_doPacket` must be called as often as possible in user’s code.

SPI_Ethernet_24j600_putByte

Prototype	`sub procedure SPI_Ethernet_24j600_putByte(dim v as byte)`
Description	This is MAC module routine. It stores one byte to address pointed by the current `ENC24J600` write pointer (`EWRPT`).
Parameters	- `v`: value to store
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim data as byte
...
SPI_Ethernet_24j600_putByte(data)  ' put an byte into ENC24J600 buffer
```

Notes None.

SPI_Ethernet_24j600_putBytes

Prototype	`sub procedure SPI_Ethernet_24j600_putBytes(dim ptr as ^byte, dim n as word)`
Description	This is MAC module routine. It stores requested number of bytes into `ENC24J600` RAM starting from current `ENC24J600` write pointer (`EWRPT`) location.
Parameters	- `ptr`: RAM buffer containing bytes to be written into `ENC24J600` RAM. - `n`: number of bytes to be written.
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  buffer as byte[17]
  ...
  buffer = "mikroElektronika"
  ...
  SPI_Ethernet_24j600_putBytes(buffer, 16)  ' put an RAM array into ENC24J600 buffer
```

Notes	None.

SPI_Ethernet_24j600_putConstBytes

Prototype	`sub procedure SPI_Ethernet_24j600_putConstBytes(const ptr as ^byte, dim n as word)`
Description	This is MAC module routine. It stores requested number of const bytes into `ENC24J600` RAM starting from current `ENC24J600` write pointer (`EWRPT`) location.
Parameters	- `ptr`: const buffer containing bytes to be written into `ENC24J600` RAM. - `n`: number of bytes to be written.
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
const
  buffer as byte[17]
  ...
  buffer = "mikroElektronika"
  ...
   SPI_Ethernet_24j600_putConstBytes(buffer, 16)  ' put a const array into ENC24J600 buffer
```

Notes	None.

SPI_Ethernet_24j600_putString

Prototype	`sub function SPI_Ethernet_24j600_putString(dim ptr as ^byte) as word`
Description	This is MAC module routine. It stores whole string (excluding null termination) into `ENC24J600` RAM starting from current `ENC24J600` write pointer (`EWRPT`) location.
Parameters	- `ptr`: string to be written into `ENC24J600` RAM.
Returns	Number of bytes written into `ENC24J600` RAM.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  buffer as string[16]
  ...
  buffer = "mikroElektronika"
  ...
  SPI_Ethernet_24j600_putString(buffer)  ' put a RAM string into ENC24J600 buffer
```

Notes	None.

SPI_Ethernet_24j600_putConstString

Prototype	`sub function SPI_Ethernet_24j600_putConstString(const ptr as ^byte) as word`
Description	This is MAC module routine. It stores whole const string (excluding null termination) into `ENC24J600` RAM starting from current `ENC24J600` write pointer (`EWRPT`) location.
Parameters	- `ptr`: const string to be written into `ENC24J600` RAM.
Returns	Number of bytes written into `ENC24J600` RAM.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
const
  buffer as string[16]
  ...
  buffer = "mikroElektronika"
  ...
   SPI_Ethernet_24j600_putConstString(buffer)  ' put a const string into
ENC24J600 buffer
```

Notes	None.

SPI_Ethernet_24j600_getByte

Prototype	`sub function SPI_Ethernet_24j600_getByte() as byte`
Description	This is MAC module routine. It fetches a byte from address pointed to by current `ENC24J600` read pointer (`ERDPT`).
Parameters	None.
Returns	Byte read from `ENC24J600` RAM.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim buffer as byte<>
...
buffer = SPI_Ethernet_24j600_getByte()  ' read a byte from ENC24J600 buffer
```

Notes	None.

SPI_Ethernet_24j600_getBytes

Prototype	`sub procedure SPI_Ethernet_24j600_getBytes(dim ptr as ^byte, dim addr as word, dim n as word)`
Description	This is MAC module routine. It fetches equested number of bytes from `ENC24J600` RAM starting from given address. If value of `0xFFFF` is passed as the address parameter, the reading will start from current `ENC24J600` read pointer (`ERDPT`) location.
Parameters	- `ptr`: buffer for storing bytes read from `ENC24J600` RAM. - `addr`: ENC24J600 RAM start address. Valid values: `0..8192`. - `n`: number of bytes to be read.
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  buffer as byte[16]
  ...
  SPI_Ethernet_24j600_getBytes(buffer, 0x100, 16)  ' read 16 bytes, starting from address 0x100
```

Notes	None.

SPI_Ethernet_24j600_UserTCP

Prototype	`sub function SPI_Ethernet_24j600_UserTCP(dim byref remoteHost as byte[4], dim remotePort as word, dim localPort as word, dim reqLength as word, dim byref flags as TEthJ600PktFlags) as word`
Description	This is TCP module routine. It is internally called by the library. The user accesses to the TCP request by using some of the SPI_Ethernet_24j600_get routines. The user puts data in the transmit buffer by using some of the SPI_Ethernet_24j600_put routines. The function must return the length in bytes of the TCP reply, or 0 if there is nothing to transmit. If there is no need to reply to the TCP requests, just define this function with return(0) as a single statement.

Parameters

- `remoteHost`: client's IP address.
- `remotePort`: client's TCP port.
- `localPort`: port to which the request is sent.
- `reqLength`: TCP request data field length.
- `flags`: structure consisted of two bit fields :

Copy Code To Clipboard

```
structure TEthj600PktFlags
   dim canCloseTCP as boolean   ' flag which closes socket
   dim isBroadcast as boolean   ' flag which denotes that the IP package has been received via subnet broadcast address
end structure
```

Returns	- `0` - there should not be a reply to the request. - Length of TCP reply data field - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.
Example	This function is internally called by the library and should not be called by the user's code.
Notes	The function source code is provided with appropriate example projects. The code should be adjusted by the user to achieve desired reply.

SPI_Ethernet_24j600_UserUDP

Prototype	`sub function SPI_Ethernet_24j600_UserUDP(dim byref remoteHost as byte[4], dim remotePort as word, dim destPort as word, dim reqLength as word, dim byref flags as TEthJ600PktFlags) as word`
Description	This is UDP module routine. It is internally called by the library. The user accesses to the UDP request by using some of the SPI_Ethernet_24j600_get routines. The user puts data in the transmit buffer by using some of the SPI_Ethernet_24j600_put routines. The function must return the length in bytes of the UDP reply, or 0 if nothing to transmit. If you don't need to reply to the UDP requests, just define this function with a return(0) as single statement.
Parameters	- `remoteHost`: client's IP address. - `remotePort`: client's port. - `destPort`: port to which the request is sent. - `reqLength`: UDP request data field length. - `flags`: structure consisted of two bit fields: Copy Code To Clipboard

```
structure TEthj600PktFlags
  dim canCloseTCP as boolean   ' flag which closes socket (not relevant to UDP)
  dim isBroadcast as boolean   ' flag which denotes that the IP package has been received via subnet broadcast address
end structure
```

Returns	- `0` - there should not be a reply to the request. - Length of UDP reply data field - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.
Example	This function is internally called by the library and should not be called by the user's code.
Notes	The function source code is provided with appropriate example projects. The code should be adjusted by the user to achieve desired reply.

SPI_Ethernet_24j600_setUserHandlers

Prototype	`sub procedure SPI_Ethernet_24j600_setUserHandlers(dim TCPHandler as ^TSPI_Ethernet_24j600_UserTCP, dim UDPHandler as ^TSPI_Ethernet_24j600_UserUDP)`
Description	Sets pointers to User TCP and UDP handler function implementations, which are automatically called by SPI Ethernet ENC24J600 library.
Parameters	- `TCPHandler`: TCP request handler - `UDPHandler`: UDP request handler.
Returns	Nothing.
Requires	SPI_Ethernet_24j600_UserTCP and SPI_Ethernet_24j600_UserUDP have to be previously defined.
Example	`SPI_Ethernet_24j600_setUserHandlers(@SPI_Ethernet_24j600_UserTCP, @SPI_Ethernet_24j600_UserUDP)`
Notes	Since all libraries are built for SSA, SSA restrictions regarding function pointers dictate that modules that use SPI_Ethernet_24j600_setUserHandlers must also be built for SSA.

SPI_Ethernet_24j600_getIpAddress

Prototype	`sub function SPI_Ethernet_24j600_getIpAddress() as word`
Description	This routine should be used when DHCP server is present on the network to fetch assigned IP address.
Parameters	None.
Returns	Pointer to the global variable holding IP address.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  ipAddr as byte[4]   ' user IP address buffer
  ...
  memcpy(ipAddr, SPI_Ethernet_24j600_getIpAddress(), 4)  ' fetch IP address
```

Notes	User should always copy the IP address from the RAM location returned by this routine into it's own IP address buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_24j600_getGwIpAddress

Prototype	`sub function SPI_Ethernet_24j600_getGwIpAddress() as word`
Description	This routine should be used when DHCP server is present on the network to fetch assigned gateway IP address.
Parameters	None.
Returns	Pointer to the global variable holding gateway IP address.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  gwIpAddr as byte[4]   ' user gateway IP address buffer
  ...
 memcpy(gwIpAddr, SPI_Ethernet_24j600_getGwIpAddress(), 4)  ' fetch gateway IP address
```

Notes	User should always copy the IP address from the RAM location returned by this routine into it's own gateway IP address buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_24j600_getDnsIpAddress

Prototype	`sub function SPI_Ethernet_24j600_getDnsIpAddress() as word`
Description	This routine should be used when DHCP server is present on the network to fetch assigned DNS IP address.
Parameters	None.
Returns	Pointer to the global variable holding DNS IP address.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  dnsIpAddr as byte[4]   ' user DNS IP address buffer
  ...
  memcpy(dnsIpAddr, SPI_Ethernet_24j600_getDnsIpAddress(), 4)   ' fetch DNS server address
```

Notes	User should always copy the IP address from the RAM location returned by this routine into it's own DNS IP address buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_24j600_getIpMask

Prototype	`sub function SPI_Ethernet_24j600_getIpMask() as word`
Description	This routine should be used when DHCP server is present on the network to fetch assigned DNS IP address.
Parameters	None.
Returns	Pointer to the global variable holding DNS IP address.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init. Available for PIC18 family MCUs only.

Example

```
dim
  IpMask as byte[4]   ' user IP subnet mask buffer
  ...
  memcpy(IpMask, SPI_Ethernet_24j600_getIpMask(), 4) ' fetch IP subnet mask
```

Notes	User should always copy the IP address from the RAM location returned by this routine into it's own IP subnet mask buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_24j600_confNetwork

Prototype	`sub procedure SPI_Ethernet_24j600_confNetwork(dim byref ipMask, gwIpAddr, dnsIpAddr as byte[4])`
Description	Configures network parameters (IP subnet mask, gateway IP address, DNS IP address) when DHCP is not used.
Parameters	- `ipMask`: IP subnet mask. - `gwIpAddr` gateway IP address. - `dnsIpAddr`: DNS IP address.
Returns	Nothing.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  ipMask    as byte[4]   ' network mask (for example : 255.255.255.0)
  gwIpAddr  as byte[4]   ' gateway (router) IP address
  dnsIpAddr as byte[4]   ' DNS server IP address
  ...
  gwIpAddr[0]  = 192
  gwIpAddr[1]  = 168
  gwIpAddr[2]  = 20
  gwIpAddr[3]  = 6

  dnsIpAddr[0] = 192
  dnsIpAddr[1] = 168
  dnsIpAddr[2] = 20
  dnsIpAddr[3] = 100

  ipMask[0]    = 255
  ipMask[1]    = 255
  ipMask[2]    = 255
  ipMask[3]    = 0
  ...
 SPI_Ethernet_24j600_confNetwork(ipMask, gwIpAddr, dnsIpAddr)  ' set network
configuration parameters
```

Notes	The above mentioned network parameters should be set by this routine only if DHCP module is not used. Otherwise DHCP will override these settings.

SPI_Ethernet_24j600_arpResolve

Prototype	`sub function SPI_Ethernet_24j600_arpResolve(dim byref ip as byte[4], dim tmax as byte) as word`
Description	This is ARP module routine. It sends an ARP request for given IP address and waits for ARP reply. If the requested IP address was resolved, an ARP cash entry is used for storing the configuration. ARP cash can store up to 3 entries.
Parameters	- `ip`: IP address to be resolved. - `tmax`: time in seconds to wait for an reply.
Returns	- MAC address behind the IP address - the requested IP address was resolved. - `0` - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  IpAddr as byte[4]  ' IP address
  ...
  IpAddr[0] = 192
  IpAddr[0] = 168
  IpAddr[0] = 1
  IpAddr[0] = 1
  ...
   SPI_Ethernet_24j600_arpResolve(IpAddr, 5)  ' get MAC address behind the above IP address, wait 5 secs for the response
```

Notes	The Ethernet services are not stopped while this routine waits for ARP reply. The incoming packets will be processed normaly during this time.

SPI_Ethernet_24j600_sendUDP

Prototype	`sub function SPI_Ethernet_24j600_sendUDP(dim byref destIP as byte[4], dim sourcePort, destPort as word, dim pkt as ^byte, dim pktLen as word) as byte`
Description	This is UDP module routine. It sends an UDP packet on the network.
Parameters	- `destIP`: remote host IP address. - `sourcePort`: local UDP source port number. - `destPort`: destination UDP port number. - `pkt`: packet to transmit. - `pktLen`: length in bytes of packet to transmit.
Returns	- `1` - UDP packet was sent successfully. - `0` - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  IpAddr as byte[4]  ' remote IP address
  ...
  IpAddr[0] = 192
  IpAddr[0] = 168
  IpAddr[0] = 1
  IpAddr[0] = 1
  ...
 SPI_Ethernet_24j600_sendUDP(IpAddr, 10001, 10001, "Hello", 5)  ' send Hello message
```

Notes	None.

SPI_Ethernet_24j600_dnsResolve

Prototype	`sub function SPI_Ethernet_24j600_dnsResolve(dim byref host as string, dim tmax as byte) as word`
Description	This is DNS module routine. It sends an DNS request for given host name and waits for DNS reply. If the requested host name was resolved, it’s IP address is stored in library global variable and a pointer containing this address is returned by the routine. UDP port `53` is used as DNS port.
Parameters	- `host`: host name to be resolved. - `tmax`: time in seconds to wait for an reply.
Returns	- pointer to the location holding the IP address - the requested host name was resolved. - `0` - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
dim
  remoteHostIpAddr as string      ' user host IP address buffer
  ...
   ' SNTP server:
   '  Zurich,  Switzerland:  Integrated  Systems  Lab,  Swiss  Fed.  Inst.  of Technology
   ' 129.132.2.21: swisstime.ethz.ch
   ' Service Area: Switzerland and Europe
  memcpy(remoteHostIpAddr, SPI_Ethernet_24j600_dnsResolve(“swisstime.ethz.ch”, 5), 4)
```

Notes	The Ethernet services are not stopped while this routine waits for DNS reply. The incoming packets will be processed normaly during this time. User should always copy the IP address from the RAM location returned by this routine into it’s own resolved host IP address buffer. These locations should not be altered by the user in any case!

SPI_Ethernet_24j600_initDHCP

Prototype	`sub function SPI_Ethernet_24j600_initDHCP(dim tmax as byte) as byte`
Description	This is DHCP module routine. It sends an DHCP request for network parameters (IP, gateway, DNS addresses and IP subnet mask) and waits for DHCP reply. If the requested parameters were obtained successfully, their values are stored into the library global variables. These parameters can be fetched by using appropriate library IP get routines: - SPI_Ethernet_24j600_getIpAddress - fetch IP address. - SPI_Ethernet_24j600_getGwIpAddress - fetch gateway IP address. - SPI_Ethernet_24j600_getDnsIpAddress - fetch DNS IP address. - SPI_Ethernet_24j600_getIpMask - fetch IP subnet mask. UDP port `68` is used as DHCP client port and UDP port `67` is used as DHCP server port.
Parameters	- `tmax`: time in seconds to wait for an reply.
Returns	- `1` - network parameters were obtained successfully. - `0` - otherwise.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
...
   SPI_Ethernet_24j600_initDHCP(5)  ' get network configuration from DHCP server, wait 5 sec for the response
  ...
```

Notes	The Ethernet services are not stopped while this routine waits for DNS reply. The incoming packets will be processed normaly during this time. When DHCP module is used, global library variable `SPI_Ethernet_24j600_userTimerSec` is used to keep track of time. It is user responsibility to increment this variable each second in it's code.

SPI_Ethernet_24j600_doDHCPLeaseTime

Prototype	`sub function SPI_Ethernet_24j600_doDHCPLeaseTime() as byte`
Description	This is DHCP module routine. It takes care of IP address lease time by decrementing the global lease time library counter. When this time expires, it's time to contact DHCP server and renew the lease.
Parameters	None.
Returns	- `0` - lease time has not expired yet. - `1` - lease time has expired, it's time to renew it.
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
while true
  ...
  if(SPI_Ethernet_24j600_doDHCPLeaseTime() <> 0) then
    ... ' it's time to renew the IP address lease
  end if
wend
```

Notes	None.

SPI_Ethernet_24j600_renewDHCP

Prototype	`sub function SPI_Ethernet_24j600_renewDHCP(dim tmax as byte) as byte`
Description	This is DHCP module routine. It sends IP address lease time renewal request to DHCP server.
Parameters	- `tmax`: time in seconds to wait for an reply.
Returns	- `1` - upon success (lease time was renewed). - `0` - otherwise (renewal request timed out).
Requires	Ethernet module has to be initialized. See SPI_Ethernet_24j600_Init.

Example

```
while true
  ...
  if(SPI_Ethernet_24j600_doDHCPLeaseTime() <> 0) then
    SPI_Ethernet_24j600_renewDHCP(5)   ' it’s time to renew the IP address lease, with 5 secs for a reply
  end if
  ...
wend
```

Notes	None.

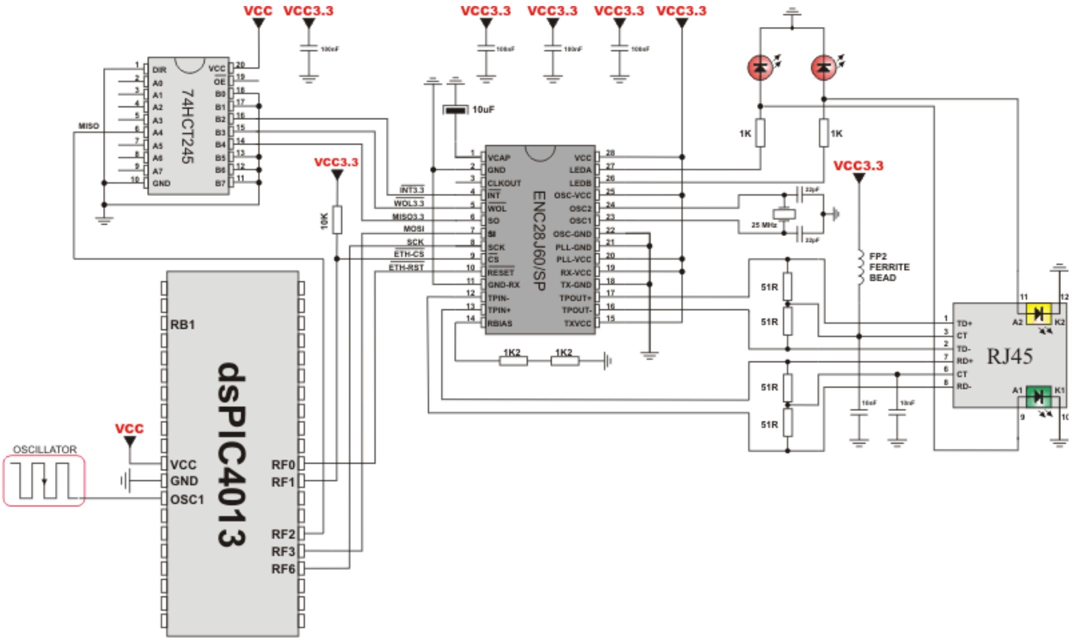
Library Example

This code shows how to use the Ethernet mini library:

-the board will reply to ARP & ICMP echo requests
-the board will reply to UDP requests on any port :
  - returns the request in upper char with a header made of remote host IP & port number
-the board will reply to HTTP requests on port 80, GET method with pathnames:
  - / will return the HTML main page
  - /s will return board status as text string
  - /t0 ... /t7 will toggle RD0 to RD7 bit and return HTML main page
  - all other requests return also HTML main page.

Main program code:

HW Connection



SPI Graphic Lcd Library

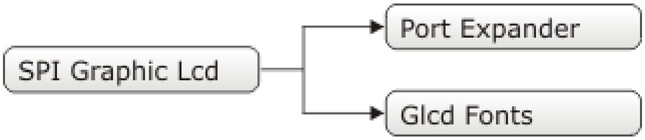
mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for operating Graphic Lcd 128x64 (with commonly used Samsung KS108/KS107 controller) via SPI interface.

For creating a custom set of Glcd images use Glcd Bitmap Editor Tool.

Important:

- When using this library with dsPIC33 and PIC24 family MCUs be aware of their voltage incompatibility with certain number of Samsung KS0108 based Glcd modules.
  So, additional external power supply for these modules may be required.
- Library uses the SPI module for communication. The user must initialize the appropriate SPI module before using the SPI Glcd Library.
- For MCUs with multiple SPI modules it is possible to initialize all of them and then switch by using the `SPI_Set_Active()` routine. See the SPI Library functions.
- This Library is designed to work with the mikroElektronika's Serial Lcd/Glcd Adapter Board pinout, see schematic at the bottom of this page for details.

Library Dependency Tree



External dependencies of SPI Lcd Library

The implementation of SPI Lcd Library routines is based on Port Expander Library routines.

External dependencies are the same as Port Expander Library external dependencies.

Library Routines

Basic routines:

- SPI_Glcd_Init
- SPI_Glcd_Set_Side
- SPI_Glcd_Set_Page
- SPI_Glcd_Set_X
- SPI_Glcd_Read_Data
- SPI_Glcd_Write_Data

Advanced routines:

- SPI_Glcd_Fill
- SPI_Glcd_Dot
- SPI_Glcd_Line
- SPI_Glcd_V_Line
- SPI_Glcd_H_Line

- SPI_Glcd_Rectangle
- SPI_Glcd_Rectangle_Round_Edges
- SPI_Glcd_Rectangle_Round_Edges_Fill
- SPI_Glcd_Box
- SPI_Glcd_Circle
- SPI_Glcd_Circle_Fill
- SPI_Glcd_Set_Font
- SPI_Glcd_Write_Char
- SPI_Glcd_Write_Text
- SPI_Glcd_Image
- SPI_Glcd_PartialImage

SPI_Glcd_Init

Prototype	`sub procedure SPI_Glcd_Init()`
Description	Initializes the Glcd module via SPI interface.
Parameters	- `DeviceAddress`: SPI expander hardware address, see schematic at the bottom of this page
Returns	Nothing.
Requires	Global variables: - `SPExpanderCS`: Chip Select line - `SPExpanderRST`: Reset line - `SPExpanderCS_Direction`: Direction of the Chip Select pin - `SPExpanderRST_Direction`: Direction of the Reset pin must be defined before using this function. The SPI module needs to be initialized. See SPIx_Init and SPIx_Init_Advanced routines.
Example	see below
Notes	None.

Example

```
' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
    SPExpanderCS  as sbit at LATF1_bit
    SPExpanderRST_Direction as sbit at TRISF0_bit
    SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections
...
' If Port Expander Library uses SPI1 module:
SPI1_Init()   ' Initialize SPI module used with PortExpander
SPI_Glcd_Init(0)
```

SPI_Glcd_Set_Side

Prototype	`sub procedure SPI_Glcd_Set_Side(dim x_pos as byte)`
Description	Selects Glcd side. Refer to the Glcd datasheet for detail explanation.
Parameters	- `x_pos`: position on x-axis. Valid values: 0..127 The parameter `x_pos` specifies the Glcd side: values from 0 to 63 specify the left side, values from 64 to 127 specify the right side.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.

Example	The following two lines are equivalent, and both of them select the left side of Glcd:
	`SPI_Glcd_Set_Side(0);`
	`SPI_Glcd_Set_Side(10);`

Notes	For side, x axis and page layout explanation see schematic at the bottom of this page.

SPI_Glcd_Set_Page

Prototype	`sub procedure SPI_Glcd_Set_Page(dim page as byte)`
Description	Selects page of Glcd.
Returns	- `page`: page number. Valid values: 0..7
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`SPI_Glcd_Set_Page(5)`
Notes	For side, x axis and page layout explanation see schematic at the bottom of this page.

SPI_Glcd_Set_X

Prototype	`sub procedure SPI_Glcd_Set_X(dim x_pos as byte)`
Description	Sets x-axis position to `x_pos` dots from the left border of Glcd within the selected side.
Parameters	- `x_pos`: position on x-axis. Valid values: 0..63
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`SPI_Glcd_Set_X(25)`
Notes	For side, x axis and page layout explanation see schematic at the bottom of this page.

SPI_Glcd_Read_Data

Prototype	`sub function SPI_Glcd_Read_Data() as byte`
Description	Reads data from the current location of Glcd memory and moves to the next location.
Returns	One byte from Glcd memory.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine. Glcd side, x-axis position and page should be set first. See the functions SPI_Glcd_Set_Side, SPI_Glcd_Set_X, and SPI_Glcd_Set_Page.
Parameters	None.

Example	`dim data_ as byte`
	`...`
	`data_ = SPI_Glcd_Read_Data()`

Notes	None.

SPI_Glcd_Write_Data

Prototype	`sub procedure SPI_Glcd_Write_Data(dim data_ as byte)`
Description	Writes one byte to the current location in Glcd memory and moves to the next location.
Parameters	- `data_`: data to be written
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine. Glcd side, x-axis position and page should be set first. See the functions SPI_Glcd_Set_Side, SPI_Glcd_Set_X, and SPI_Glcd_Set_Page.

Example	`dim data_ as byte`
	`...`
	`SPI_Glcd_Write_Data(data_)`

Notes	None.

SPI_Glcd_Fill

Prototype	`sub procedure SPI_Glcd_Fill(dim pattern as byte)`
Description	Fills Glcd memory with byte `pattern`. To clear the Glcd screen, use `SPI_Glcd_Fill(0)`. To fill the screen completely, use `SPI_Glcd_Fill(0xFF)`.
Parameters	- `pattern`: byte to fill Glcd memory with
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Clear screen` `SPI_Glcd_Fill(0)`
Notes	None.

SPI_Glcd_Dot

Prototype	`sub procedure SPI_Glcd_Dot(dim x_pos, y_pos, color as byte)`
Description	Draws a dot on Glcd at coordinates (`x_pos, y_pos`).
Parameters	- `x_pos`: x position. Valid values: 0..127 - `y_pos`: y position. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the dot state: 0 clears dot, 1 puts a dot, and 2 inverts dot state.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Invert the dot in the upper left corner` `SPI_Glcd_Dot(0, 0, 2)`
Notes	For x and y axis layout explanation see schematic at the bottom of this page..

SPI_Glcd_Line

Prototype	`sub procedure SPI_Glcd_Line(dim x_start, y_start, x_end, y_end as integer, dim color as byte)`
Description	Draws a line on Glcd. Parameters:
Parameters	- `x_start`: x coordinate of the line start. Valid values: 0..127 - `y_start`: y coordinate of the line start. Valid values: 0..63 - `x_end`: x coordinate of the line end. Valid values: 0..127 - `y_end`: y coordinate of the line end. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 Parameter `color` determines the line color: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Draw a line between dots (0,0) and (20,30)` `SPI_Glcd_Line(0, 0, 20, 30, 1)`
Notes	None.

SPI_Glcd_V_Line

Prototype	`sub procedure SPI_Glcd_V_Line(dim y_start, y_end, x_pos, color as byte)`
Description	Draws a vertical line on Glcd.
Parameters	- `y_start`: y coordinate of the line start. Valid values: 0..63 - `y_end`: y coordinate of the line end. Valid values: 0..63 - `x_pos`: x coordinate of vertical line. Valid values: 0..127 - `color`: color parameter. Valid values: 0..2 Parameter `color` determines the line color: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Draw a vertical line between dots (10,5) and (10,25)` `SPI_Glcd_V_Line(5, 25, 10, 1)`
Notes	None.

SPI_Glcd_H_Line

Prototype	`sub procedure SPI_Glcd_H_Line(dim x_start, x_end, y_pos, color as byte)`
Description	Draws a horizontal line on Glcd.
Parameters	- `x_start`: x coordinate of the line start. Valid values: 0..127 - `x_end`: x coordinate of the line end. Valid values: 0..127 - `y_pos`: y coordinate of horizontal line. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the line color: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Draw a horizontal line between dots (10,20) and (50,20)` `SPI_Glcd_H_Line(10, 50, 20, 1)`
Notes	None.

SPI_Glcd_Rectangle

Prototype	`sub procedure SPI_Glcd_Rectangle(dim x_upper_left, y_upper_left, x_bottom_right, y_bottom_right, color as byte)`
Description	Draws a rectangle on Glcd.
Parameters	- `x_upper_left`: x coordinate of the upper left rectangle corner. Valid values: 0..127 - `y_upper_left`: y coordinate of the upper left rectangle corner. Valid values: 0..63 - `x_bottom_right`: x coordinate of the lower right rectangle corner. Valid values: 0..127 - `y_bottom_right`: y coordinate of the lower right rectangle corner. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the rectangle border: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Draw a rectangle between dots (5,5) and (40,40)` `SPI_Glcd_Rectangle(5, 5, 40, 40, 1)`
Notes	None.

SPI_Glcd_Rectangle_Round_Edges

Prototype	`sub procedure Glcd_Rectangle_Round_Edges(dim x_upper_left as byte, dim y_upper_left as byte, dim x_bottom_right as byte, dim y_bottom_right as byte, dim radius as byte, dim color as byte)`
Description	Draws a rounded edge rectangle on Glcd.
Parameters	- `x_upper_left`: x coordinate of the upper left rectangle corner. Valid values: 0..127 - `y_upper_left`: y coordinate of the upper left rectangle corner. Valid values: 0..63 - `x_bottom_right`: x coordinate of the lower right rectangle corner. Valid values: 0..127 - `y_bottom_right`: y coordinate of the lower right rectangle corner. Valid values: 0..63 - `round_radius`: radius of the rounded edge. - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the rectangle border: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see SPI_Glcd_Init routine.
Example	`' Draw a rounded edge rectangle between dots (5,5) and (40,40) with the radius of 12` `SPI_Glcd_Rectangle_Round_Edges(5, 5, 40, 40, 12, 1)`
Notes	None.

SPI_Glcd_Rectangle_Round_Edges_Fill

Prototype	`sub procedure SPI_Glcd_Rectangle_Round_Edges_Fill(dim x_upper_left as byte, dim y_upper_left as byte, dim x_bottom_right as byte, dim y_bottom_right as byte, dim radius as byte, dim color as byte)`
Description	Draws a filled rounded edge rectangle on Glcd with color.
Parameters	- `x_upper_left`: x coordinate of the upper left rectangle corner. Valid values: 0..127 - `y_upper_left`: y coordinate of the upper left rectangle corner. Valid values: 0..63 - `x_bottom_right`: x coordinate of the lower right rectangle corner. Valid values: 0..127 - `y_bottom_right`: y coordinate of the lower right rectangle corner. Valid values: 0..63 - `round_radius`: radius of the rounded edge - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the rectangle border: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized, see SPI_Glcd_Init routine.
Example	`' Draws a filled rounded edge rectangle between dots (5,5) and (40,40) with the radius of 12` `SPI_Glcd_Rectangle_Round_Edges_Fill(5, 5, 40, 40, 12, 1)`
Notes	None.

SPI_Glcd_Box

Prototype	`sub procedure SPI_Glcd_Box(dim x_upper_left, y_upper_left, x_bottom_right, y_bottom_right, color as byte)`
Description	Draws a box on Glcd.
Parameters	- `x_upper_left`: x coordinate of the upper left box corner. Valid values: 0..127 - `y_upper_left`: y coordinate of the upper left box corner. Valid values: 0..63 - `x_bottom_right`: x coordinate of the lower right box corner. Valid values: 0..127 - `y_bottom_right`: y coordinate of the lower right box corner. Valid values: 0..63 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the box fill: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Draw a box between dots (5,15) and (20,40)` `SPI_Glcd_Box(5, 15, 20, 40, 1)`
Notes	None.

SPI_Glcd_Circle

Prototype	`sub procedure SPI_Glcd_Circle(dim x_center, y_center, radius as integer, dim color as byte)`
Description	Draws a circle on Glcd.
Parameters	- `x_center`: x coordinate of the circle center. Valid values: 0..127 - `y_center`: y coordinate of the circle center. Valid values: 0..63 - `radius`: radius size - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the circle line: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Draw a circle with center in (50,50) and radius=10` `SPI_Glcd_Circle(50, 50, 10, 1)`
Notes	None.

SPI_Glcd_Circle_FIll

Prototype	`sub procedure SPI_Glcd_Circle_Fill(dim x_center as integer, dim y_center as integer, dim radius as integer, dim color as byte)`
Description	Draws a filled circle on Glcd.
Parameters	- `x_center`: x coordinate of the circle center. Valid values: 0..127 - `y_center`: y coordinate of the circle center. Valid values: 0..63 - `radius`: radius size - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the circle : 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Draw a circle with center in (50,50) and radius=10` `SPI_Glcd_Circle_Fill(50, 50, 10, 1)`
Notes	None.

SPI_Glcd_Set_Font

Prototype	`sub procedure SPI_Glcd_Set_Font(dim activeFont as LongInt, dim aFontWidth, aFontHeight as byte, dim aFontOffs as word)`
Description	Sets font that will be used with SPI_Glcd_Write_Char and SPI_Glcd_Write_Text routines.
Parameters	None.
Returns	- `activeFont`: font to be set. Needs to be formatted as an array of char - `aFontWidth`: width of the font characters in dots. - `aFontHeight`: height of the font characters in dots. - `aFontOffs`: number that represents difference between the mikroBasic PRO for dsPIC30/33 and PIC24 character set and regular ASCII set (eg. if 'A' is 65 in ASCII character, and 'A' is 45 in the mikroBasic PRO for dsPIC30/33 and PIC24 character set, aFontOffs is 20). Demo fonts supplied with the library have an offset of 32, which means that they start with space. The user can use fonts given in the file `__Lib_GLCDFonts` file located in the Uses folder or create his own fonts. List of supported fonts: - `Font_Glcd_System3x5` - `Font_Glcd_System5x7` - `Font_Glcd_5x7` - `Font_Glcd_Character8x7` For the sake of the backward compatibility, these fonts are supported also: - `System3x5` (equivalent to `Font_Glcd_System3x5`) - `FontSystem5x7_v2` (equivalent to `Font_Glcd_System5x7`) - `font5x7` (equivalent to `Font_Glcd_5x7`) - `Character8x7` (equivalent to `Font_Glcd_Character8x7`)
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Use the custom 5x7 font "myfont" which starts with space (32):` `SPI_Glcd_Set_Font(@myfont, 5, 7, 32)`
Notes	None.

SPI_Glcd_Write_Char

Prototype	`sub procedure SPI_Glcd_Write_Char(dim chr1, x_pos, page_num, color as byte)`
Description	Prints character on Glcd.
Parameters	- `chr1`: character to be written - `x_pos`: character starting position on x-axis. Valid values: 0..(127-FontWidth) - `page_num`: the number of the page on which character will be written. Valid values: 0..7 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the character: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine. Use the SPI_Glcd_Set_Font to specify the font for display; if no font is specified, then the default `Font_Glcd_System5x7` font supplied with the library will be used.
Example	`' Write character 'C' on the position 10 inside the page 2:` `SPI_Glcd_Write_Char("C", 10, 2, 1)`
Notes	For x axis and page layout explanation see schematic at the bottom of this page.

SPI_Glcd_Write_Text

Prototype	`sub procedure SPI_Glcd_Write_Text(dim byref text as char[40], dim x_pos, page_num, color as byte)`
Description	Prints text on Glcd.
Parameters	- `text`: text to be written - `x_pos`: text starting position on x-axis. - `page_num`: the number of the page on which text will be written. Valid values: 0..7 - `color`: color parameter. Valid values: 0..2 The parameter `color` determines the color of the text: 0 white, 1 black, and 2 inverts each dot.
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine. Use the SPI_Glcd_Set_Font to specify the font for display; if no font is specified, then the default `Font_Glcd_System5x7` font supplied with the library will be used.
Example	`' Write text "Hello world!" on the position 10 inside the page 2:` `SPI_Glcd_Write_Text("Hello world!", 10, 2, 1)`
Notes	For x axis and page layout explanation see schematic at the bottom of this page.

SPI_Glcd_Image

Prototype	`sub procedure SPI_Glcd_Image(dim byref const image as byte)`
Description	Displays bitmap on Glcd.
Parameters	- `image`: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic PRO for dsPIC30/33 and PIC24 pointer to const and pointer to RAM equivalency).
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`SPI_Glcd_Image(@my_image)`
Notes	Use the mikroBasic PRO for dsPIC30/33 and PIC24 integrated Glcd Bitmap Editor, **Tools > Glcd Bitmap Editor**, to convert image to a constant array suitable for displaying on Glcd.

SPI_Glcd_PartialImage

Prototype	`sub procedure SPI_T6963C_PartialImage(dim x_left, y_top, width, height, picture_width, picture_height as word, const image as ^byte)`
Description	Displays a partial area of the image on a desired location.
Parameters	- `x_left`: x coordinate of the desired locations (upper left coordinate). - `y_top`: y coordinate of the desired location (upper left coordinate). - `width`: desired image width. - `height`: desired image height. - `picture_width`: width of the original image. - `picture_height`: height of the original image. - `image`: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic PRO for PIC pointer to const and pointer to RAM equivalency).
Returns	Nothing.
Requires	Glcd needs to be initialized for SPI communication, see SPI_Glcd_Init routine.
Example	`' Draws a 10x15 part of the image starting from the upper left corner on the coordinate (10,12). Original image size is 16x32.` `SPI_Glcd_PartialImage(10, 12, 10, 15, 16, 32, @image)`
Notes	Use the mikroBasic PRO for dsPIC30/33 and PIC24 integrated Glcd Bitmap Editor, Tools > Glcd Bitmap Editor, to convert image to a constant array suitable for displaying on Glcd.

Library Example

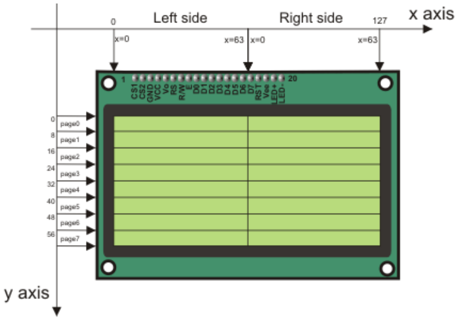
The example demonstrates how to communicate to KS0108 Glcd via the SPI module, using serial to parallel convertor MCP23S17.

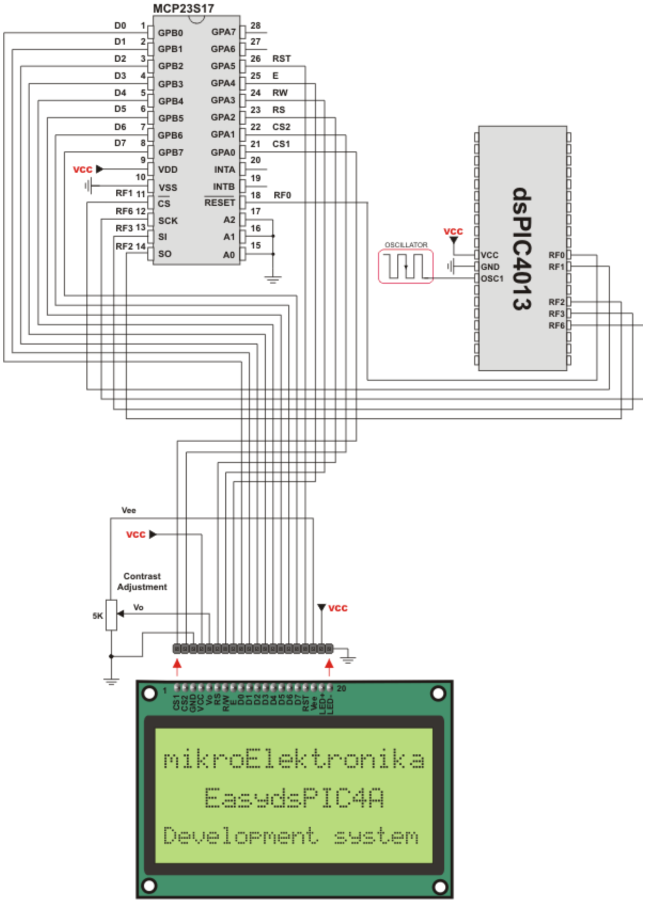
Copy Code To Clipboard

```
program SPI_Glcd

' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
    SPExpanderCS  as sbit at LATF1_bit
    SPExpanderRST_Direction as sbit at TRISF0_bit
    SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections

dim someText as char[20]
    counter as byte

sub procedure Delay2S
  Delay_ms(2000)
end sub

main:
  #DEFINE COMPLETE_EXAMPLE    ' comment this line to make simpler/smaller example
  ADPCFG = 0xFFFF                                  ' initialize AN pins as digital

   ' If Port Expander Library uses SPI1 module
  SPI1_Init()    ' Initialize SPI module used with PortExpander

  ' If Port Expander Library uses SPI2 module
  '  SPI2_Init()  ' Initialize SPI module used with PortExpander

  SPI_Glcd_Init(0)                                 ' Initialize Glcd via SPI
  SPI_Glcd_Fill(0x00)                              ' Clear Glcd

  while (TRUE)
    #IFDEF COMPLETE_EXAMPLE
    SPI_Glcd_Image(@truck_bmp)                     ' Draw image
    Delay2s() Delay2s()
    #ENDIF

    SPI_Glcd_Fill(0x00)                            ' Clear Glcd
    Delay2s

    SPI_Glcd_Box(62,40,124,63,1)                   ' Draw box
    SPI_Glcd_Rectangle(5,5,84,35,1)                ' Draw rectangle
    SPI_Glcd_Line(0, 0, 127, 63, 1)                ' Draw line
    Delay2s()
    counter = 5
    while (counter < 60)                           ' Draw horizontal and vertical line
      Delay_ms(250)
      SPI_Glcd_V_Line(2, 54, counter, 1)
      SPI_Glcd_H_Line(2, 120, counter, 1)
      counter = counter + 5
    wend
```

```
    Delay2s()

    #IFDEF COMPLETE_EXAMPLE
    SPI_Glcd_Fill(0x00)                            ' Clear Glcd
    SPI_Glcd_Set_Font(@Character8x7, 8, 7, 32)   ' Choose font
    SPI_Glcd_Write_Text("mikroE", 1, 7, 2)       ' Write string
    #ENDIF

    for counter = 1 to 10                          ' Draw circles
      SPI_Glcd_Circle(63,32, 3*counter, 1)
    next counter
    Delay2s()

    #IFDEF COMPLETE_EXAMPLE
    SPI_Glcd_Box(10,20, 70,63, 2)                  ' Draw box
    Delay2s()

    SPI_Glcd_Fill(0xFF)                            ' Fill Glcd

    SPI_Glcd_Set_Font(@Character8x7, 8, 7, 32)   ' Change font
    someText = "8x7 Font"
    SPI_Glcd_Write_Text(someText, 5, 0, 2)       ' Write string
    Delay2s()

    SPI_Glcd_Set_Font(@System3x5, 3, 5, 32)      ' Change font
    someText = "3X5 CAPITALS ONLY"
    SPI_Glcd_Write_Text(someText, 60, 2, 2)      ' Write string
    Delay2s()

    SPI_Glcd_Set_Font(@font5x7, 5, 7, 32)        ' Change font
    someText = "5x7 Font"
    SPI_Glcd_Write_Text(someText, 5, 4, 2)       ' Write string
    Delay2s()

    SPI_Glcd_Set_Font(@FontSystem5x7_v2, 5, 7, 32)   ' Change font
    someText = "5x7 Font (v2)"
    SPI_Glcd_Write_Text(someText, 50, 6, 2)          ' Write string
    Delay2s()
    #ENDIF
  wend
end.
```

HW Connection





SPI Glcd HW connection

SPI Lcd Library

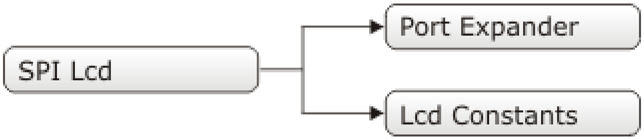
The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for communication with Lcd (with HD44780 compliant controllers) in 4-bit mode via SPI interface.

For creating a custom set of Lcd characters use Lcd Custom Character Tool.

Important:

- When using this library with dsPIC33 and PIC24 family MCUs be aware of their voltage incompatibility with certain number of Lcd modules.
  So, additional external power supply for these modules may be required.
- Library uses the SPI module for communication. The user must initialize the appropriate SPI module before using the SPI Lcd Library.
- For MCUs with multiple SPI modules it is possible to initialize all of them and then switch by using the `SPI_Set_Active()` routine. See the SPI Library functions.
- This Library is designed to work with the mikroElektronika's Serial Lcd Adapter Board pinout, see schematic at the bottom of this page for details.

Library Dependency Tree



External dependencies of SPI Lcd Library

The implementation of SPI Lcd Library routines is based on Port Expander Library routines.

External dependencies are the same as Port Expander Library external dependencies.

Library Routines

- SPI_Lcd_Config
- SPI_Lcd_Out
- SPI_Lcd_Out_Cp
- SPI_Lcd_Chr
- SPI_Lcd_Chr_Cp
- SPI_Lcd_Cmd

SPI_Lcd_Config

Prototype	`sub procedure SPI_Lcd_Config(dim DeviceAddress as byte)`
Description	Initializes the Lcd module via SPI interface.
Parameters	- `DeviceAddress`: SPI expander hardware address, see schematic at the bottom of this page
Returns	Nothing.
Requires	Global variables: - `SPExpanderCS`: Chip Select line - `SPExpanderRST`: Reset line - `SPExpanderCS_Direction`: Direction of the Chip Select pin - `SPExpanderRST_Direction`: Direction of the Reset pin must be defined before using this function. The SPI module needs to be initialized. See SPIx_Init and SPIx_Init_Advanced routines.

Example

```
' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
    SPExpanderCS  as sbit at LATF1_bit
    SPExpanderRST_Direction as sbit at TRISF0_bit
    SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections
...
' If Port Expander Library uses SPI1 module
SPI1_Init()                    ' Initialize SPI module used with PortExpander
SPI_Lcd_Config(0)              ' initialize lcd over spi interface
```

Notes	None.

SPI_Lcd_Out

Prototype	`sub procedure SPI_Lcd_Out(dim row, column as byte, dim byref text as string)`
Description	Prints text on the Lcd starting from specified position. Both string variables and literals can be passed as a text.
Parameters	- `row`: starting position row number - `column`: starting position column number - `text`: text to be written
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd_Config routine.

Example

```
' Write text “Hello!” on Lcd starting from row 1, column 3:
SPI_Lcd_Out(1, 3, “Hello!”)
```

Notes	None.

SPI_Lcd_Out_Cp

Prototype	`sub procedure SPI_Lcd_Out_CP(dim byref text as string)`
Description	Prints text on the Lcd at current cursor position. Both string variables and literals can be passed as a text.
Parameters	- `text`: text to be written
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd_Config routine.
Example	`' Write text "Here!" at current cursor position:` `SPI_Lcd_Out_CP("Here!")`
Notes	None.

SPI_Lcd_Chr

Prototype	`sub procedure SPI_Lcd_Chr(dim Row, Column, Out_Char as byte)`
Description	Prints character on Lcd at specified position. Both variables and literals can be passed as character.
Parameters	- `Row`: writing position row number - `Column`: writing position column number - `Out_Char`: character to be written
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd_Config routine.
Example	`' Write character "i" at row 2, column 3:` `SPI_Lcd_Chr(2, 3, "i")`
Notes	None.

SPI_Lcd_Chr_Cp

Prototype	`sub procedure SPI_Lcd_Chr_CP(dim Out_Char as byte)`
Description	Prints character on Lcd at current cursor position. Both variables and literals can be passed as character.
Parameters	- `Out_Char`: character to be written
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd_Config routine.
Example	`' Write character "e" at current cursor position:` `SPI_Lcd_Chr_Cp("e")`
Notes	None.

SPI_Lcd_Cmd

Prototype	`sub procedure SPI_Lcd_Cmd(dim out_char as byte)`
Description	Sends command to Lcd.
Parameters	- `out_char`: command to be sent
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd_Config routine.
Example	`' Clear Lcd display:` `SPI_Lcd_Cmd(_LCD_CLEAR)`
Notes	Predefined constants can be passed to the routine, see Available SPI Lcd Commands.

Available SPI Lcd Commands

SPI Lcd Command	Purpose
`_LCD_FIRST_ROW`	Move cursor to the 1st row
`_LCD_SECOND_ROW`	Move cursor to the 2nd row
`_LCD_THIRD_ROW`	Move cursor to the 3rd row
`_LCD_FOURTH_ROW`	Move cursor to the 4th row
`_LCD_CLEAR`	Clear display
`_LCD_RETURN_HOME`	Return cursor to home position, returns a shifted display to its original position. Display data RAM is unaffected.
`_LCD_CURSOR_OFF`	Turn off cursor
`_LCD_UNDERLINE_ON`	Underline cursor on
`_LCD_BLINK_CURSOR_ON`	Blink cursor on
`_LCD_MOVE_CURSOR_LEFT`	Move cursor left without changing display data RAM
`_LCD_MOVE_CURSOR_RIGHT`	Move cursor right without changing display data RAM
`_LCD_TURN_ON`	Turn Lcd display on
`_LCD_TURN_OFF`	Turn Lcd display off
`_LCD_SHIFT_LEFT`	Shift display left without changing display data RAM
`_LCD_SHIFT_RIGHT`	Shift display right without changing display data RAM

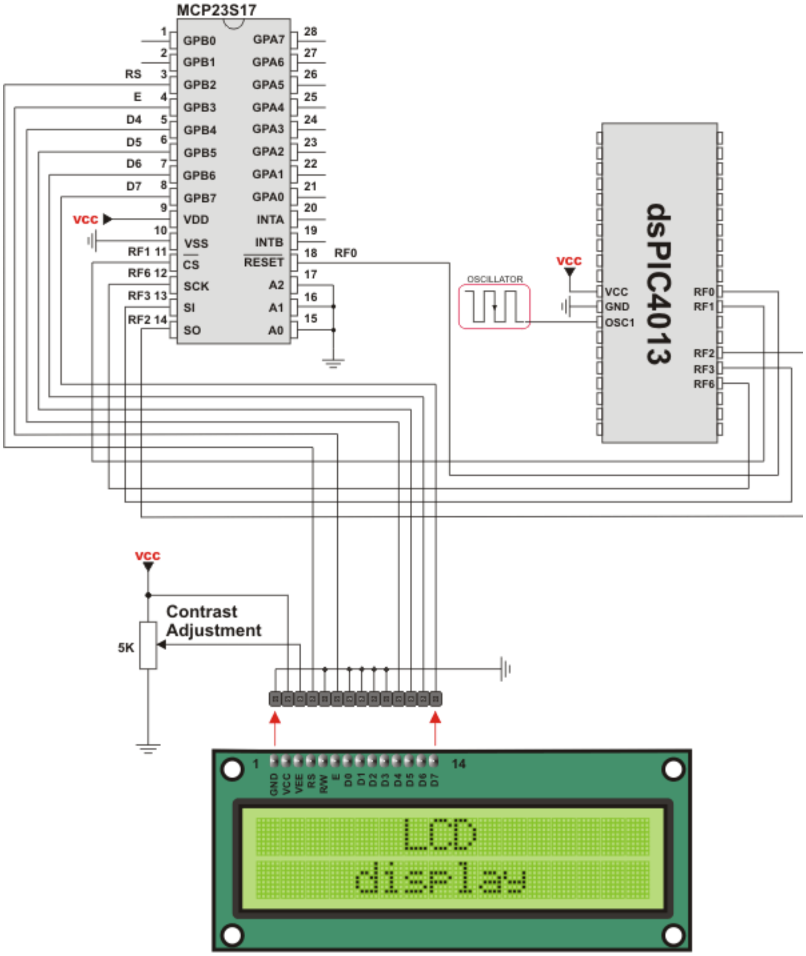
Library Example

Default Pin Configuration

Use `SPI_Lcd_Init` for default pin settings (see the first figure below).

Copy Code To Clipboard

```
program Spi_Lcd

dim text as char[16]
dim counter as byte

' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
dim SPExpanderCS  as sbit at LATF1_bit
dim SPExpanderRST_Direction as sbit at TRISF0_bit
dim SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections

sub procedure Move_Delay()              ' Function used for text moving
  Delay_ms(500)                         ' You can change the moving speed here
end sub

main:
  text = "mikroElektronika"
  ADPCFG = 0xFFFF                       ' initialize AN pins as digital
  SPI1_Init()                           ' Initialize SPI
  Spi_Lcd_Config(0)                     ' Initialize LCD over SPI interface
  Spi_Lcd_Cmd(_LCD_CLEAR)               ' Clear display
  Spi_Lcd_Cmd(_LCD_CURSOR_OFF)          ' Turn cursor off
  Spi_Lcd_Out(1,6, "mikroE")            ' Print text to LCD, 1st row, 6th column
  Spi_Lcd_Chr_CP("!")                   ' Append '!'
  Spi_Lcd_Out(2,1, text)                ' Print text to LCD, 2nd row, 1st column

  '  Spi_Lcd_Out(3,1,'mikroE')          ' For LCD with more than two rows
  '  Spi_Lcd_Out(4,15,'mikroE')         ' For LCD with more than two rows

  ' Moving text
  for counter = 0 to 3                  ' Move text to the right 4 times
    Spi_Lcd_Cmd(_LCD_SHIFT_RIGHT)
    Move_Delay()
  next counter

  while TRUE                            ' Endless loop
    for counter = 0 to 6                ' Move text to the left 7 times
      Spi_Lcd_Cmd(_LCD_SHIFT_LEFT)
      Move_Delay()
    next counter

    for counter = 0 to 6                ' Move text to the right 7 times
      Spi_Lcd_Cmd(_LCD_SHIFT_RIGHT)
      Move_Delay()
    next counter
  wend
end.
```



Lcd HW connection by default initialization (using SPI_Lcd_Init)

SPI Lcd8 (8-bit interface) Library

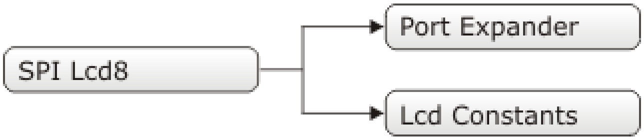
The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for communication with Lcd (with HD44780 compliant controllers) in 8-bit mode via SPI interface.

For creating a custom set of Lcd characters use Lcd Custom Character Tool.

Important:

- When using this library with dsPIC33 and PIC24 family MCUs be aware of their voltage incompatibility with certain number of Lcd modules.
  So, additional external power supply for these modules may be required.
- The library uses the SPI module for communication. The user must initialize the appropriate SPI module before using the SPI Lcd8 Library.
- For MCUs with multiple SPI modules it is possible to initialize all of them and then switch by using the `SPI_Set_Active()` routine. See the SPI Library functions.
- This Library is designed to work with the mikroElektronika's Serial Lcd/Glcd Adapter Board pinout, see schematic at the bottom of this page for details.

Library Dependency Tree



External dependencies of SPI Lcd Library

The implementation of SPI Lcd Library routines is based on Port Expander Library routines.

External dependencies are the same as Port Expander Library external dependencies.

Library Routines

- SPI_Lcd8_Config
- SPI_Lcd8_Out
- SPI_Lcd8_Out_Cp
- SPI_Lcd8_Chr
- SPI_Lcd8_Chr_Cp
- SPI_Lcd8_Cmd

SPI_Lcd8_Config

Prototype	`sub procedure SPI_Lcd8_Config(dim DeviceAddress as byte)`
Description	Initializes the Lcd module via SPI interface.
Parameters	- `DeviceAddress`: SPI expander hardware address, see schematic at the bottom of this page
Returns	Nothing.
Requires	Global variables: - `SPExpanderCS`: Chip Select line - `SPExpanderRST`: Reset line - `SPExpanderCS_Direction`: Direction of the Chip Select pin - `SPExpanderRST_Direction`: Direction of the Reset pin must be defined before using this function. The SPI module needs to be initialized. See SPIx_Init and SPIx_Init_Advanced routines.
Example	
Notes	None.

Example:

```
' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
    SPExpanderCS  as sbit at LATF1_bit
    SPExpanderRST_Direction as sbit at TRISF0_bit
    SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections
...
SPI1_Init()                             ' Initialize SPI interface
SPI_Lcd8_Config(0)                      ' Intialize Lcd in 8bit mode via spi
```

SPI_Lcd8_Out

Prototype	`sub procedure SPI_Lcd8_Out(dim row, column as byte, dim byref text as string)`
Description	Prints text on Lcd starting from specified position. Both string variables and literals can be passed as a text.
Parameters	- `row`: starting position row number - `column`: starting position column number - `text`: text to be written
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd8_Config routine.
Example	
Notes	None.

Example:

```
' Write text "Hello!" on Lcd starting from row 1, column 3:
SPI_Lcd8_Out(1, 3, "Hello!")
```

SPI_Lcd8_Out_Cp

Prototype	`sub procedure SPI_Lcd8_Out_CP(dim byref text as string)`
Description	Prints text on Lcd at current cursor position. Both string variables and literals can be passed as a text.
Parameters	- `text`: text to be written
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd8_Config routine.
Example	`' Write text “Here!” at current cursor position:` `SPI_Lcd8_Out_Cp(“Here!”)`
Notes	None.

SPI_Lcd8_Chr

Prototype	`sub procedure SPI_Lcd8_Chr(dim row, column, out_char as byte)`
Description	Prints character on Lcd at specified position. Both variables and literals can be passed as character.
Parameters	- `row`: writing position row number - `column`: writing position column number - `out_char`: character to be written
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd8_Config routine.
Example	`' Write character “i” at row 2, column 3:` `SPI_Lcd8_Chr(2, 3, “i”)`
Notes	None.

SPI_Lcd8_Chr_Cp

Prototype	`sub procedure SPI_Lcd8_Chr_CP(dim out_char as byte)`
Description	Prints character on Lcd at current cursor position. Both variables and literals can be passed as character.
Parameters	- `out_char`: character to be written
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd8_Config routine.

Example	Print “e” at current cursor position:
	`' Write character “e” at current cursor position:`
	`SPI_Lcd8_Chr_Cp(“e”)`

Notes	None.

SPI_Lcd8_Cmd

Prototype	`sub procedure SPI_Lcd8_Cmd(dim out_char as byte)`
Description	Sends command to Lcd.
Parameters	- `out_char`: command to be sent
Returns	Nothing.
Requires	Lcd needs to be initialized for SPI communication, see SPI_Lcd8_Config routine.
Example	`' Clear Lcd display:` `SPI_Lcd8_Cmd(_LCD_CLEAR)`
Notes	Predefined constants can be passed to the routine, see Available SPI Lcd8 Commands.

Available SPI Lcd8 Commands

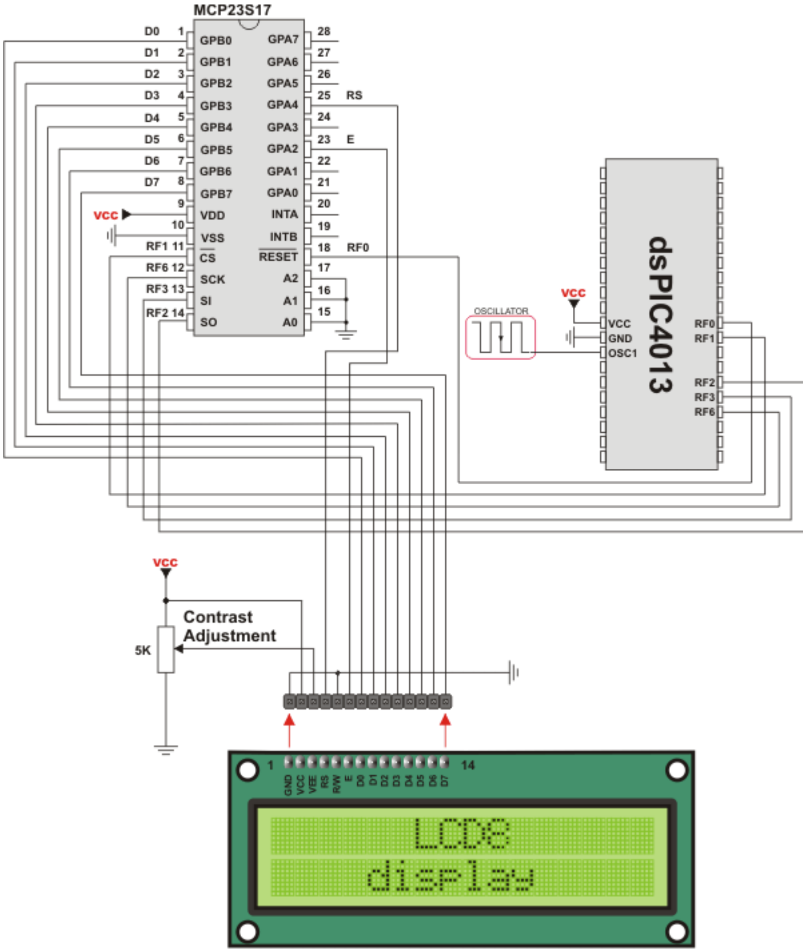
SPI Lcd8 Command	Purpose
`_LCD_FIRST_ROW`	Move cursor to the 1st row
`_LCD_SECOND_ROW`	Move cursor to the 2nd row
`_LCD_THIRD_ROW`	Move cursor to the 3rd row
`_LCD_FOURTH_ROW`	Move cursor to the 4th row
`_LCD_CLEAR`	Clear display
`_LCD_RETURN_HOME`	Return cursor to home position, returns a shifted display to its original position. Display data RAM is unaffected.
`_LCD_CURSOR_OFF`	Turn off cursor
`_LCD_UNDERLINE_ON`	Underline cursor on
`_LCD_BLINK_CURSOR_ON`	Blink cursor on
`_LCD_MOVE_CURSOR_LEFT`	Move cursor left without changing display data RAM
`_LCD_MOVE_CURSOR_RIGHT`	Move cursor right without changing display data RAM
`_LCD_TURN_ON`	Turn Lcd display on
`_LCD_TURN_OFF`	Turn Lcd display off
`_LCD_SHIFT_LEFT`	Shift display left without changing display data RAM
`_LCD_SHIFT_RIGHT`	Shift display right without changing display data RAM

Library Example

This example demonstrates how to communicate Lcd in 8-bit mode via the SPI module, using serial to parallel convertor MCP23S17.

Copy Code To Clipboard

```
program Spi_Lcd8

dim text as char[16]
dim counter as byte

' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
dim SPExpanderCS  as sbit at LATF1_bit
dim SPExpanderRST_Direction as sbit at TRISF0_bit
dim SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections

sub procedure Move_Delay()             ' Function used for text moving
  Delay_ms(500)                        ' You can change the moving speed here
end sub

main:
  text = “mikroElektronika”
  ADPCFG = 0xFFFF                      ' initialize AN pins as digital
  SPI1_Init()                          ' Initialize SPI
  Spi_Lcd8_Config(0)                   ' Initialize LCD over SPI interface
  Spi_Lcd8_Cmd(_LCD_CLEAR)             ' Clear display
  Spi_Lcd8_Cmd(_LCD_CURSOR_OFF)        ' Turn cursor off
  Spi_Lcd8_Out(1,6, “mikroE”)          ' Print text to LCD, 1st row, 6th column
  Spi_Lcd8_Chr_CP(“!”)                 ' Append ‘!’
  Spi_Lcd8_Out(2,1, text)              ' Print text to LCD, 2nd row, 1st column

  '  Spi_Lcd8_Out(3,1,’mikroE’)        ' For LCD with more than two rows
  '  Spi_Lcd8_Out(4,15,’mikroE’)       ' For LCD with more than two rows

  ' Moving text
  for counter = 0 to 3                 ' Move text to the right 4 times
    Spi_Lcd8_Cmd(_LCD_SHIFT_RIGHT)
    Move_Delay()
  next counter

  while TRUE                           ' Endless loop
    for counter = 0 to 6               ' Move text to the left 7 times
      Spi_Lcd8_Cmd(_LCD_SHIFT_LEFT)
      Move_Delay()
    next counter

 for counter = 0 to 6                  ' Move text to the right 7 times
      Spi_Lcd8_Cmd(_LCD_SHIFT_RIGHT)
      Move_Delay()
    next counter
  wend
end.
```



SPI Lcd8 HW connection

SPI T6963C Graphic Lcd Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for working with Glcds based on TOSHIBA T6963C controller via SPI interface. The Toshiba T6963C is a very popular Lcd controller for the use in small graphics modules. It is capable of controlling displays with a resolution up to 240x128. Because of its low power and small outline it is most suitable for mobile applications such as PDAs, MP3 players or mobile measurement equipment. Although this controller is small, it has a capability of displaying and merging text and graphics and it manages all interfacing signals to the displays Row and Column drivers.

For creating a custom set of Glcd images use Glcd Bitmap Editor Tool.

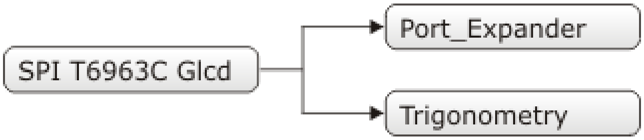
Important:

- When using this library with dsPIC33 and PIC24 family MCUs be aware of their voltage incompatibility with certain number of T6963C based Glcd modules. So, additional external power supply for these modules may be required.
- Glcd size based initialization routines can be found in setup library files located in the Uses folder.
- The user must make sure that used MCU has appropriate ports and pins. If this is not the case the user should adjust initialization routines.
- The library uses the SPI module for communication. The user must initialize the appropriate SPI module before using the SPI T6963C Glcd Library.
- For MCUs with multiple SPI modules it is possible to initialize both of them and then switch by using the `SPI_Set_Active()` routine. See the SPI Library functions.
- This Library is designed to work with mikroElektronika's Serial Glcd 240x128 and 240x64 Adapter Boards pinout, see schematic at the bottom of this page for details.
- To use constants located in `__Lib_SPIT6963C_Const.mbas` file, user must include it the source file: `include __Lib_SPIT6963C_Const`.

Some mikroElektronika's adapter boards have pinout different from T6369C datasheets. Appropriate relations between these labels are given in the table below:

Adapter Board	T6369C datasheet
RS	C/D
R/W	/RD
E	/WR

Library Dependency Tree



External dependencies of SPI T6963C Graphic Lcd Library

The implementation of SPI T6963C Graphic Lcd Library routines is based on Port Expander Library routines.

External dependencies are the same as Port Expander Library external dependencies.

Library Routines

- SPI_T6963C_config
- SPI_T6963C_writeData
- SPI_T6963C_writeCommand
- SPI_T6963C_setPtr
- SPI_T6963C_waitReady
- SPI_T6963C_fill
- SPI_T6963C_dot
- SPI_T6963C_write_char
- SPI_T6963C_write_text
- SPI_T6963C_line
- SPI_T6963C_rectangle
- SPI_T6963C_rectangle_round_edges
- SPI_T6963C_rectangle_round_edges_fill
- SPI_T6963C_box
- SPI_T6963C_circle
- SPI_T6963C_circle_fill
- SPI_T6963C_image
- SPI_T6963C_PartialImage
- SPI_T6963C_sprite
- SPI_T6963C_set_cursor
- SPI_T6963C_clearBit
- SPI_T6963C_setBit
- SPI_T6963C_negBit
- SPI_T6963C_displayGrPanel
- SPI_T6963C_displayTxtPanel
- SPI_T6963C_setGrPanel
- SPI_T6963C_setTxtPanel
- SPI_T6963C_panelFill
- SPI_T6963C_grFill
- SPI_T6963C_txtFill
- SPI_T6963C_cursor_height
- SPI_T6963C_graphics
- SPI_T6963C_text
- SPI_T6963C_cursor
- SPI_T6963C_cursor_blink

SPI_T6963C_config

Prototype	`sub procedure SPI_T6963C_config(dim width, height, fntW as word, dim DeviceAddress as byte, dim wr, rd, cd, rst as byte)`

Description	Initializes T6963C Graphic Lcd controller.
	Display RAM organization:
	The library cuts RAM into panels: a complete panel is one graphics panel followed by a text panel (see schematic below).
	`+---------------------+ /\`
	`+ GRAPHICS PANEL #0   +  |`
	`+                     +  |`
	`+                     +  |`
	`+                     +  |`
	`+---------------------+  | PANEL 0`
	`+ TEXT PANEL #0       +  |`
	`+                     + \/`
	`+---------------------+ /\`
	`+ GRAPHICS PANEL #1   +  |`
	`+                     +  |`
	`+                     +  |`
	`+                     +  |`
	`+---------------------+  | PANEL 1`
	`+ TEXT PANEL #1       +  |`
	`+                     +  |`
	`+---------------------+ \/`

Parameters	- `width`: width of the Glcd panel - `height`: height of the Glcd panel - `fntW`: font width - `DeviceAddress`: SPI expander hardware address, see schematic at the bottom of this page - `wr`: write signal pin on Glcd control port - `rd`: read signal pin on Glcd control port - `cd`: command/data signal pin on Glcd control port - `rst`: reset signal pin on Glcd control port
Returns	Nothing.
Requires	Global variables: - `SPExpanderCS`: Chip Select line - `SPExpanderRST`: Reset line - `SPExpanderCS_Direction`: Direction of the Chip Select pin - `SPExpanderRST_Direction`: Direction of the Reset pin must be defined before using this function. The SPI module needs to be initialized. See the SPIx_Init and SPIx_Init_Advanced routines.

Example

```
' Port Expander module connections
dim SPExpanderRST as sbit at LATF0_bit
    SPExpanderCS  as sbit at LATF1_bit
    SPExpanderRST_Direction as sbit at TRISF0_bit
    SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections
...
' Initialize SPI module
SPI1_Init()
SPI_T6963C_Config(240, 64, 8, 0, 0, 1, 3, 4)
```

Notes	None.

SPI_T6963C_writeData

Prototype	`sub procedure SPI_T6963C_writeData(dim data_ as byte)`
Description	Writes data to T6963C controller via SPI interface.
Parameters	- `data_`: data to be written
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_writeData(data_)`
Notes	None.

SPI_T6963C_writeCommand

Prototype	`sub procedure SP` `I_T6963C_writeCommand(dim data_ as byte)`
Description	Writes command to T6963C controller via SPI interface.
Parameters	- `data_`: command to be written
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_writeCommand(SPI_T6963C_CURSOR_POINTER_SET)`
Notes	None.

SPI_T6963C_setPtr

Prototype	`sub procedure SPI_T6963C_setPtr(dim p as word, dim c as byte)`
Description	Sets the memory pointer `p` for command `p`.
Parameters	- `p`: address where command should be written - `c`: command to be written
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_setPtr(SPI_T6963C_grHomeAddr + start, SPI_T6963C_ADDRESS_POINTER_SET)`
Notes	None.

SPI_T6963C_waitReady

Prototype	`sub procedure SPI_T6963C_waitReady()`
Description	Pools the status byte, and loops until Toshiba Glcd module is ready.
Parameters	None.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_waitReady()`
Notes	None.

SPI_T6963C_fill

Prototype	`sub procedure SPI_T6963C_fill(dim v as byte, dim start, len as word)`
Description	Fills controller memory block with given byte.
Parameters	- `v`: byte to be written - `start`: starting address of the memory block - `len`: length of the memory block in bytes
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_fill(0x33,0x00FF,0x000F)`
Notes	None.

SPI_T6963C_dot

Prototype	`sub procedure SPI_T6963C_dot(dim x, y as integer, dim color as byte)`
Description	Writes a char in the current text panel of Glcd at coordinates (x, y).
Returns	- `x`: dot position on x-axis - `y`: dot position on y-axis - `color`: color parameter. Valid values: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_dot(x0, y0, SPI_T6963C_BLACK)`
Notes	None.

SPI_T6963C_write_char

Prototype	`sub procedure SPI_T6963C_write_char(dim c, x, y, mode as byte)`
Description	Writes a char in the current text panel of Glcd at coordinates (x, y).
Parameters	- `c`: char to be written - `x`: char position on x-axis - `y`: char position on y-axis - `mode`: mode parameter. Valid values: Valid values: SPI_T6963C_ROM_MODE_OR, SPI_T6963C_ROM_MODE_XOR, SPI_T6963C_ROM_MODE_AND and SPI_T6963C_ROM_MODE_TEXT Mode parameter explanation: - OR Mode: In the OR-Mode, text and graphics can be displayed and the data is logically “OR-ed”. This is the most common way of combining text and graphics for example labels on buttons. - XOR-Mode: In this mode, the text and graphics data are combined via the logical “exclusive OR”. This can be useful to display text in negative mode, i.e. white text on black background. - AND-Mode: The text and graphic data shown on display are combined via the logical “AND function”. - TEXT-Mode: This option is only available when displaying just a text. The Text Attribute values are stored in the graphic area of display memory. For more details see the T6963C datasheet.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_write_char(“A”,22,23,SPI_T6963C_ROM_MODE_AND)`
Notes	None.

SPI_T6963C_write_text

Prototype	`sub procedure SPI_T6963C_write_text(dim byref str as byte[10], dim x, y, mode as byte)`
Description	Writes text in the current text panel of Glcd at coordinates (x, y).
Parameters	- `str`: text to be written - `x`: text position on x-axis - `y`: text position on y-axis - `mode`: mode parameter. Valid values: SPI_T6963C_ROM_MODE_OR, SPI_T6963C_ROM_MODE_XOR, SPI_T6963C_ROM_MODE_AND and SPI_T6963C_ROM_MODE_TEXT Mode parameter explanation: - OR Mode: In the OR-Mode, text and graphics can be displayed and the data is logically “OR-ed”. This is the most common way of combining text and graphics for example labels on buttons. - XOR-Mode: In this mode, the text and graphics data are combined via the logical “exclusive OR”. This can be useful to display text in negative mode, i.e. white text on black background. - AND-Mode: The text and graphic data shown on the display are combined via the logical “AND function”. - TEXT-Mode: This option is only available when displaying just a text. The Text Attribute values are stored in the graphic area of display memory. For more details see the T6963C datasheet.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_write_text(“GLCD LIBRARY DEMO, WELCOME !”, 0, 0, SPI_T6963C_ROM_MODE_XOR)`
Notes	None.

SPI_T6963C_line

Prototype	`sub procedure SPI_T6963C_line(dim x0, y0, x1, y1 as integer, dim pcolor as byte)`
Description	Draws a line from (x0, y0) to (x1, y1).
Parameters	- `x0`: x coordinate of the line start - `y0`: y coordinate of the line end - `x1`: x coordinate of the line start - `y1`: y coordinate of the line end - `pcolor`: color parameter. Valid values: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_line(0, 0, 239, 127, SPI_T6963C_WHITE)`
Notes	None.

SPI_T6963C_rectangle

Prototype	`sub procedure SPI_T6963C_rectangle(dim x0, y0, x1, y1 as integer, dim pcolor as byte)`
Description	Draws a rectangle on Glcd.
Parameters	- `x0`: x coordinate of the upper left rectangle corner - `y0`: y coordinate of the upper left rectangle corner - `x1`: x coordinate of the lower right rectangle corner - `y1`: y coordinate of the lower right rectangle corner - `pcolor`: color parameter. Valid values: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_rectangle(20, 20, 219, 107, SPI_T6963C_WHITE)`
Notes	None.

SPI_T6963C_rectangle_round_edges

Prototype	`sub procedure SPI_T6963C_rectangle_round_edges(dim x0 as integer, dim y0 as integer, dim x1 as integer, dim y1 as integer, dim radius as integer, dim pcolor as byte)`
Description	Draws a rounded edge rectangle on Glcd.
Parameters	- `x0`: x coordinate of the upper left rectangle corner - `y0`: y coordinate of the upper left rectangle corner - `x1`: x coordinate of the lower right rectangle corner - `y1`: y coordinate of the lower right rectangle corner - `round_radius`: radius of the rounded edge. - `pcolor`: color parameter. Valid values: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_rectangle_round_edges(20, 20, 219, 107, 12, SPI_T6963C_WHITE)`
Notes	None.

SPI_T6963C_rectangle_round_edges_fill

Prototype	`sub procedure SPI_T6963C_rectangle_round_edges_fill(dim x0 as integer, dim y0 as integer, dim x1 as integer, dim y1 as integer, dim radius as integer, dim pcolor as byte)`
Description	Draws a filled rounded edge rectangle on Glcd.
Parameters	- `x0`: x coordinate of the upper left rectangle corner - `y0`: y coordinate of the upper left rectangle corner - `x1`: x coordinate of the lower right rectangle corner - `y1`: y coordinate of the lower right rectangle corner - `round_radius`: radius of the rounded edge - `pcolor`: color parameter. Valid values: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_rectangle_round_edges_fill(20, 20, 219, 107, 12, SPI_T6963C_WHITE)`
Notes	None.

SPI_T6963C_box

Prototype	`sub procedure SPI_T6963C_box(dim x0, y0, x1, y1 as integer, dim pcolor as byte)`
Description	Draws a box on the Glcd
Parameters	- `x0`: x coordinate of the upper left box corner - `y0`: y coordinate of the upper left box corner - `x1`: x coordinate of the lower right box corner - `y1`: y coordinate of the lower right box corner - `pcolor`: color parameter. Valid values: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_box(0, 119, 239, 127, SPI_T6963C_WHITE)`
Notes	None.

SPI_T6963C_circle

Prototype	`sub procedure SPI_T6963C_circle(dim x, y as integer, dim r as longint, dim pcolor as word)`
Description	Draws a circle on the Glcd.
Parameters	- `x`: x coordinate of the circle center - `y`: y coordinate of the circle center - `r`: radius size - `pcolor`: color parameter. Valid values: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_circle(120, 64, 110, SPI_T6963C_WHITE)`
Notes	None.

SPI_T6963C_circle_fill

Prototype	`sub procedure SPI_T6963C_circle_fill(dim x as integer, dim y as integer, dim r as longint, dim pcolor as byte)`
Description	Draws a filled circle on the Glcd.
Parameters	- `x`: x coordinate of the circle center - `y`: y coordinate of the circle center - `r`: radius size - `pcolor`: color parameter. Valid values: SPI_T6963C_BLACK and SPI_T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_circle_fill(120, 64, 110, SPI_T6963C_WHITE)`
Notes	None.

SPI_T6963C_image

Prototype	`sub procedure SPI_T6963C_image(dim pic as ^ const byte)`
Description	Displays bitmap on Glcd.
Parameters	- `pic`: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic PRO for dsPIC30/33 and PIC24 pointer to const and pointer to RAM equivalency).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_image(my_image)`
Notes	Image dimension must match the display dimension. Use the integrated Glcd Bitmap Editor (menu option **Tools › Glcd Bitmap Editor**) to convert image to a constant array suitable for displaying on Glcd.

SPI_T6963C_PartialImage

Prototype	`sub procedure SPI_T6963C_PartialImage(dim x_left, y_top, width, height, picture_width, picture_height as word, const image as ^byte)`
Description	Displays a partial area of the image on a desired location.
Parameters	- `x_left`: x coordinate of the desired location (upper left coordinate). - `y_top`: y coordinate of the desired location (upper left coordinate). - `width`: desired image width. - `height`: desired image height. - `picture_width`: width of the original image. - `picture_height`: height of the original image. - `image`: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic PRO for PIC pointer to const and pointer to RAM equivalency).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' Draws a 10x15 part of the image starting from the upper left corner on the coordinate (10,12). Original image size is 16x32.` `SPI_T6963C_PartialImage(10, 12, 10, 15, 16, 32, @image)`
Notes	Image dimension must match the display dimension. Use the integrated Glcd Bitmap Editor (menu option **Tools › Glcd Bitmap Editor**) to convert image to a constant array suitable for displaying on Glcd.

SPI_T6963C_sprite

Prototype	`sub procedure SPI_T6963C_sprite(dim px, py as byte, dim const pic as ^byte, dim sx, sy as byte)`
Description	Fills graphic rectangle area (px, py) to (px+sx, py+sy) with custom size picture.
Parameters	- `px`: x coordinate of the upper left picture corner. Valid values: multiples of the font width - `py`: y coordinate of the upper left picture corner - `pic`: picture to be displayed - `sx`: picture width. Valid values: multiples of the font width - `sy`: picture height
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_sprite(76, 4, einstein, 88, 119)  ' draw a sprite`
Notes	If `px` and `sx` parameters are not multiples of the font width they will be scaled to the nearest lower number that is a multiple of the font width.

SPI_T6963C_set_cursor

Prototype	`sub procedure SPI_T6963c_set_cursor(dim x, y as byte)`
Description	Sets cursor to row x and column y.
Parameters	- `x`: cursor position row number - `y`: cursor position column number
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963c_set_cursor(cposx, cposy)`
Notes	None.

SPI_T6963C_clearBit

Prototype	`sub procedure SPI_T6963C_clearBit(dim b as byte)`
Description	Clears control port bit(s).
Parameters	- `b`: bit mask. The function will clear bit `x` on control port if bit `x` in bit mask is set to 1.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' clear bits 0 and 1 on control port` `SPI_T6963C_clearBit(0x0003)`
Notes	None.

SPI_T6963C_setBit

Prototype	`sub procedure SPI_T6963C_setBit(dim b as byte)`
Description	Sets control port bit(s).
Parameters	- `b`: bit mask. The function will set bit `x` on control port if bit `x` in bit mask is set to 1.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' set bits 0 and 1 on control port` `SPI_T6963C_setBit(0x0003)`
Notes	None.

SPI_T6963C_negBit

Prototype	`sub procedure SPI_T6963C_negBit(dim b as byte)`
Description	Negates control port bit(s).
Parameters	- `b`: bit mask. The function will negate bit `x` on control port if bit `x` in bit mask is set to 1.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' negate bits 0 and 1 on control port` `SPI_T6963C_negBit(0x0003)`
Notes	None.

SPI_T6963C_displayGrPanel

Prototype	`sub procedure SPI_T6963C_displayGrPanel(dim n as word)`
Description	Display selected graphic panel.
Parameters	- `n`: graphic panel number. Valid values: `0` and `1`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' display graphic panel 1` `SPI_T6963C_displayGrPanel(1)`
Notes	None.

SPI_T6963C_displayTxtPanel

Prototype	`sub procedure SPI_T6963C_displayTxtPanel(dim n as word)`
Description	Display selected text panel.
Parameters	- `n`: text panel number. Valid values: `0` and `1`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' display text panel 1` `SPI_T6963C_displayTxtPanel(1)`
Notes	None.

SPI_T6963C_setGrPanel

Prototype	`sub procedure SPI_T6963C_setGrPanel(dim n as word)`
Description	Compute start address for selected graphic panel and set appropriate internal pointers. All subsequent graphic operations will be preformed at this graphic panel.
Parameters	- `n`: graphic panel number. Valid values: `0` and `1`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' set graphic panel 1 as current graphic panel.` `SPI_T6963C_setGrPanel(1)`
Notes	None.

SPI_T6963C_setTxtPanel

Prototype	`sub procedure SPI_T6963C_setTxtPanel(dim n as word)`
Description	Compute start address for selected text panel and set appropriate internal pointers. All subsequent text operations will be preformed at this text panel.
Parameters	- `n`: text panel number. Valid values: `0` and `1`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' set text panel 1 as current text panel.` `SPI_T6963C_setTxtPanel(1)`
Notes	None.

SPI_T6963C_panelFill

Prototype	`sub procedure SPI_T6963C_panelFill(dim v as word)`
Description	Fill current panel in full (graphic+text) with appropriate value (0 to clear).
Parameters	- `v`: value to fill panel with.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' clear current panel` `SPI_T6963C_panelFill(0)`
Notes	None.

SPI_T6963C_grFill

Prototype	`sub procedure SPI_T6963C_grFill(dim v as word)`
Description	Fill current graphic panel with appropriate value (0 to clear).
Parameters	- `v`: value to fill graphic panel with.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' clear current graphic panel` `SPI_T6963C_grFill(0)`
Notes	None.

SPI_T6963C_txtFill

Prototype	`sub procedure SPI_T6963C_txtFill(dim v as word)`
Description	Fill current text panel with appropriate value (0 to clear).
Parameters	- `v`: this value increased by 32 will be used to fill text panel.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' clear current text panel` `SPI_T6963C_txtFill(0)`
Notes	None.

SPI_T6963C_cursor_height

Prototype	`sub procedure SPI_T6963C_cursor_height(dim n as byte)`
Description	Set cursor size.
Parameters	- `n`: cursor height. Valid values: `0..7`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`SPI_T6963C_cursor_height(7)`
Notes	None.

SPI_T6963C_graphics

Prototype	`sub procedure SPI_T6963C_graphics(dim n as word)`
Description	Enable/disable graphic displaying.
Parameters	- `n`: graphic enable/disable parameter. Valid values: `0` (disable graphic dispaying) and `1` (enable graphic displaying).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' enable graphic displaying` `SPI_T6963C_graphics(1)`
Notes	None.

SPI_T6963C_text

Prototype	`sub procedure SPI_T6963C_text(dim n as word)`
Description	Enable/disable text displaying.
Parameters	- `n`: text enable/disable parameter. Valid values: `0` (disable text dispaying) and `1` (enable text displaying).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' enable text displaying` `SPI_T6963C_text(1)`
Notes	None.

SPI_T6963C_cursor

Prototype	`sub procedure SPI_T6963C_cursor(dim n as word)`
Description	Set cursor on/off.
Parameters	- `n`: on/off parameter. Valid values: `0` (set cursor off) and `1` (set cursor on).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' set cursor on` `SPI_T6963C_cursor(1)`
Notes	None.

SPI_T6963C_cursor_blink

Prototype	`sub procedure SPI_T6963C_cursor_blink(dim n as word)`
Description	Enable/disable cursor blinking.
Parameters	- `n`: cursor blinking enable/disable parameter. Valid values: `0` (disable cursor blinking) and `1` (enable cursor blinking).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See SPI_T6963C_Config routine.
Example	`' enable cursor blinking` `SPI_T6963C_cursor_blink(1)`
Notes	None.

Library Example

The following drawing demo tests advanced routines of the SPI T6963C Glcd library. Hardware configurations in this example are made for the EasydsPIC4A board and dsPIC30F4013.

Copy Code To Clipboard

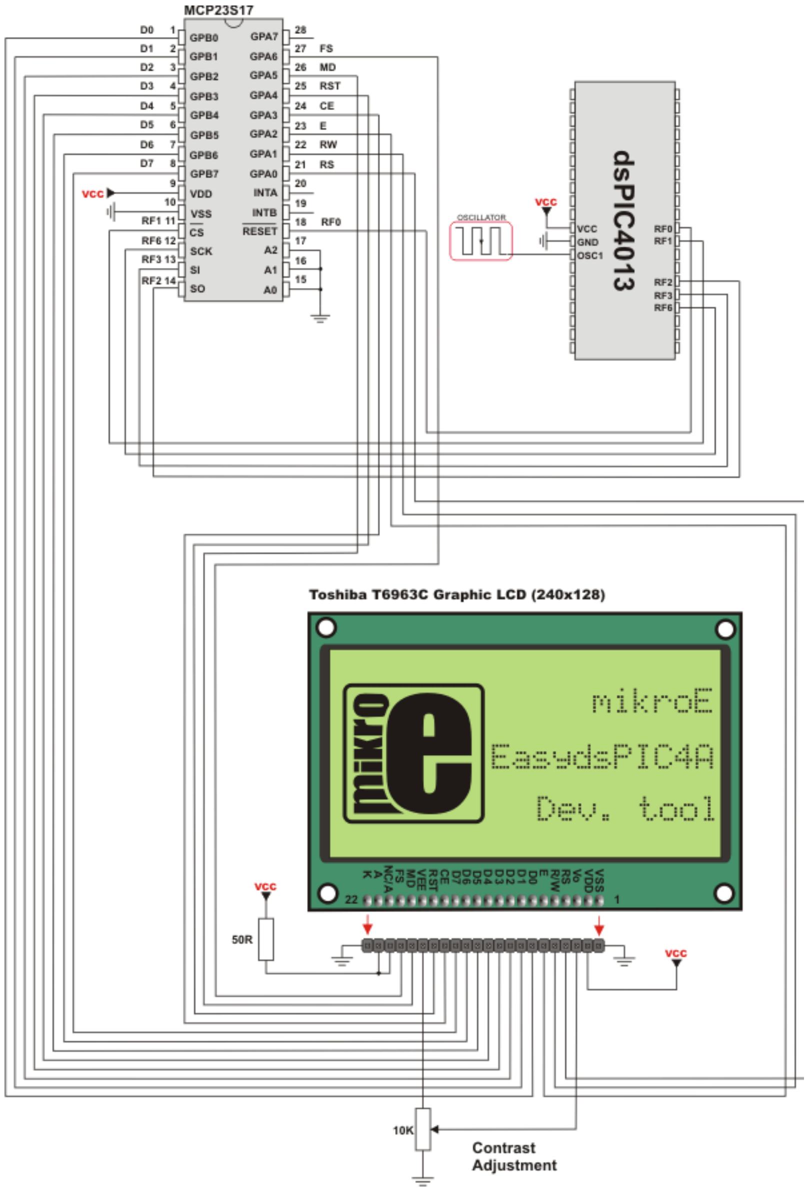
```
program SPI_T6963C_240x128

include __Lib_SPIT6963C_Const

dim
' Port Expander module connections
   SPExpanderRST as sbit at LATF0_bit
   SPExpanderCS  as sbit at LATF1_bit
   SPExpanderRST_Direction as sbit at TRISF0_bit
   SPExpanderCS_Direction  as sbit at TRISF1_bit
' End Port Expander module connections

dim   panel as byte         ' current panel
          i as word         ' general purpose register
      curs  as byte         ' cursor visibility
      cposx,
      cposy as word         ' cursor x-y position
      txt, txt1 as string[29]

main:

  txt1 = “ EINSTEIN WOULD HAVE LIKED mE”
  txt  = “ GLCD LIBRARY DEMO, WELCOME !”

  #DEFINE COMPLETE_EXAMPLE       ' comment this line to make simpler/smaller example
  ADPCFG = 0xFFFF                              ' initialize AN pins as digital

  TRISB8_bit = 1               ' Set RB8 as input
  TRISB9_bit = 1               ' Set RB9 as input
  TRISB10_bit = 1              ' Set RB10 as input
```

```
  TRISB11_bit = 1                ' Set RB11 as input
  TRISB12_bit = 1                ' Set RB12 as input

  panel = 0
  i = 0
  curs = 0
  cposx = 0
  cposy = 0

  ' Initialize SPI module
  SPI1_Init()

  ' ' If Port Expander Library uses SPI2 module
  ' Pass pointer to SPI Read function of used SPI module

  ' Initialize SPI module used with PortExpander
  ' SPI2_Init_Advanced(_SPI_MASTER, _SPI_FCY_DIV2, _SPI_CLK_HI_TRAILING)

'    * init display for 240 pixel width and 128 pixel height
'    * 8 bits character width
'    * data bus on MCP23S17 portB
'    * control bus on MCP23S17 portA
'    * bit 2 is !WR
'    * bit 1 is !RD
'    * bit 0 is !CD
'    * bit 4 is RST
'    * chip enable, reverse on, 8x8 font internaly set in library

  ' Initialize SPI Toshiba 240x128
  SPI_T6963C_Config(240, 128, 8, 0, 2, 1, 0, 4)
  Delay_ms(1000)

'      * Enable both graphics and text display at the same time

  SPI_T6963C_graphics(1)
  SPI_T6963C_text(1)

'      * Text messages
  SPI_T6963C_write_text(txt, 0, 0, SPI_T6963C_ROM_MODE_XOR)
  SPI_T6963C_write_text(txt1, 0, 15, SPI_T6963C_ROM_MODE_XOR)

'      * Cursor
  SPI_T6963C_cursor_height(8)          ' 8 pixel height
  SPI_T6963C_set_cursor(0, 0)          ' Move cursor to top left
  SPI_T6963C_cursor(0)                 ' Cursor off

'      * Draw rectangles
  SPI_T6963C_rectangle(0, 0, 239, 127, SPI_T6963C_WHITE)
  SPI_T6963C_rectangle(20, 20, 219, 107, SPI_T6963C_WHITE)
  SPI_T6963C_rectangle(40, 40, 199, 87, SPI_T6963C_WHITE)
  SPI_T6963C_rectangle(60, 60, 179, 67, SPI_T6963C_WHITE)
```

```
'     * Draw a cross
  SPI_T6963C_line(0, 0, 239, 127, SPI_T6963C_WHITE)
  SPI_T6963C_line(0, 127, 239, 0, SPI_T6963C_WHITE)

'     * Draw solid boxes
  SPI_T6963C_box(0, 0, 239, 8, SPI_T6963C_WHITE)
  SPI_T6963C_box(0, 119, 239, 127, SPI_T6963C_WHITE)

'     * Draw circles
  #IFDEF COMPLETE_EXAMPLE
  SPI_T6963C_circle(120, 64, 10, SPI_T6963C_WHITE)
  SPI_T6963C_circle(120, 64, 30, SPI_T6963C_WHITE)
  SPI_T6963C_circle(120, 64, 50, SPI_T6963C_WHITE)
  SPI_T6963C_circle(120, 64, 70, SPI_T6963C_WHITE)
  SPI_T6963C_circle(120, 64, 90, SPI_T6963C_WHITE)
  SPI_T6963C_circle(120, 64, 110, SPI_T6963C_WHITE)
  SPI_T6963C_circle(120, 64, 130, SPI_T6963C_WHITE)

  SPI_T6963C_sprite(76, 4, @einstein_bmp, 88, 119)   ' Draw a sprite

  SPI_T6963C_setGrPanel(1)                           ' Select other graphic panel

  SPI_T6963C_image(@mikroE_240x128_bmp)      ' Fill the graphic screen with a picture
  #ENDIF

  while (TRUE)                               ' Endless loop


'     * If RB8 is pressed, toggle the display between graphic panel 0 and graphic 1
    if(RB8_bit <> 0) then
      SPI_T6963C_graphics(1)
      SPI_T6963C_text(0)
      Delay_ms(300)

'           * If RB9 is pressed, display only graphic panel
    #IFDEF COMPLETE_EXAMPLE
    else
      if (RB9_bit <> 0) then
        Inc(panel)
        panel = panel and 1
        SPI_T6963C_displayGrPanel(panel)
        Delay_ms(300)
    #ENDIF

'           * If RB10 is pressed, display only text panel
      else
        if (RB10_bit <> 0) then
          SPI_T6963C_graphics(0)
          SPI_T6963C_text(1)
          Delay_ms(300)
```

```
'           * If RB11 is pressed, display text and graphic panels
        else
          if (RB11_bit <> 0) then
            SPI_T6963C_graphics(1)
            SPI_T6963C_text(1)
            Delay_ms(300)

'           * If RB12 is pressed, change cursor
          else
            if(RB12_bit <> 0) then
              Inc(curs)
              if (curs = 3) then
                curs = 0
              end if

              select case curs
                case 0
                    ' no cursor
                    SPI_T6963C_cursor(0)

                case 1
                    ' blinking cursor
                    SPI_T6963C_cursor(1)
                    SPI_T6963C_cursor_blink(1)
                case 2
                    ' non blinking cursor
                    SPI_T6963C_cursor(1)
                    SPI_T6963C_cursor_blink(0)
              end select

              Delay_ms(300)
            end if
          end if
        end if
      #IFDEF COMPLETE_EXAMPLE
      end if
      #ENDIF
    end if

'     * Move cursor, even if not visible
    Inc(cposx)

    if (cposx = SPI_T6963C_txtCols) then
      cposx = 0
      Inc(cposy)
      if (cposy = SPI_T6963C_grHeight div SPI_T6963C_CHARACTER_HEIGHT) then
        cposy = 0
      end if
    end if
    SPI_T6963C_set_cursor(cposx, cposy)

    Delay_ms(100)
  wend
end.
```

HW Connection



SPI T6963C Glcd HW connection

T6963C Graphic Lcd Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for working with Glcds based on TOSHIBA T6963C controller. The Toshiba T6963C is a very popular Lcd controller for the use in small graphics modules. It is capable of controlling displays with a resolution up to 240x128. Because of its low power and small outline it is most suitable for mobile applications such as PDAs, MP3 players or mobile measurement equipment. Although small, this contoller has a capability of displaying and merging text and graphics and it manages all the interfacing signals to the displays Row and Column drivers.

For creating a custom set of Glcd images use Glcd Bitmap Editor Tool.

Important:

- When using this library with dsPIC33 and PIC24 family of MCUs be aware of their voltage incompatibility with certain number of T6963C based Glcd modules. So, additional external power supply for these modules may be required.
- ChipEnable(CE), FontSelect(FS) and Reverse(MD) have to be set to appropriate levels by the user outside of the T6963C_Init() function. See the Library Example code at the bottom of this page.
- Glcd size based initialization routines can be found in setup library files located in the Uses folder.
- The user must make sure that used MCU has appropriate ports and pins. If this is not the case the user should adjust initialization routines.

Some mikroElektronika's adapter boards have pinout different from T6369C datasheets. Appropriate relations between these labels are given in the table below:

Adapter Board	T6369C datasheet
RS	C/D
R/W	/RD
E	/WR

Library Dependency Tree

External dependencies of T6963C Graphic Lcd Library

The following variables must be defined in all projects using T6963C Graphic Lcd library:	Description:	Example:
`dim T6963C_dataPort as byte sfr external`	T6963C Data Port.	`dim T6963C_dataPort as byte at PORTB`
`dim T6963C_ctrlwr as sbit sfr external`	Write signal.	`dim T6963C_ctrlwr as sbit at LATF2_bit`
`dim T6963C_ctrlrd as sbit sfr external`	Read signal.	`dim T6963C_ctrlrd as sbit at LATF1_bit`
`dim T6963C_ctrlcd as sbit sfr external`	Command/Data signal.	`dim T6963C_ctrlcd as sbit at LATF0_bit`
`dim T6963C_ctrlrst as sbit sfr external`	Reset signal.	`dim T6963C_ctrlrst as sbit at LATF4_bit`
`dim T6963C_ctrlwr_Direction as sbit sfr external`	Direction of the Write pin.	`dim T6963C_ctrlwr_Direction as sbit at TRISF2_bit`
`dim T6963C_ctrlrd_Direction as sbit sfr external`	Direction of the Read pin.	`dim T6963C_ctrlrd_Direction as sbit at TRISF1_bit`
`dim T6963C_ctrlcd_Direction as sbit sfr external`	Direction of the Command/ Data pin.	`dim T6963C_ctrlcd_Direction as sbit at TRISF0_bit`
`dim T6963C_ctrlrst_Direction as sbit sfr external`	Direction of the Reset pin.	`dim T6963C_ctrlrst_Direction as sbit at TRISF4_bit`

Library Routines

- T6963C_init
- T6963C_writeData
- T6963C_writeCommand
- T6963C_setPtr
- T6963C_waitReady
- T6963C_fill
- T6963C_dot
- T6963C_write_char
- T6963C_write_text
- T6963C_line
- T6963C_rectangle
- T6963C_rectangle_round_edges
- T6963C_rectangle_round_edges_fill
- T6963C_box
- T6963C_circle
- T6963C_circle_fill
- T6963C_image
- T6963C_PartialImage
- T6963C_sprite
- T6963C_set_cursor
- T6963C_displayGrPanel
- T6963C_displayTxtPanel
- T6963C_setGrPanel
- T6963C_setTxtPanel
- T6963C_panelFill
- T6963C_grFill
- T6963C_txtFill
- T6963C_cursor_height
- T6963C_graphics
- T6963C_text
- T6963C_cursor
- T6963C_cursor_blink

T6963C_init

Prototype	`sub procedure T6963C_init(dim width, height, fntW as word, dim byref data as word, dim byref cntrl as word, dim wr, rd, cd, rst as word)`
Description	Initalizes the Graphic Lcd controller. Display RAM organization: The library cuts the RAM into panels: a complete panel is one graphics panel followed by a text panel (see schematic below).

```
+---------------------+ /\
+ GRAPHICS PANEL #0   +  |
+                     +  |
+                     +  |
+                     +  |
+---------------------+  | PANEL 0
+ TEXT PANEL #0       +  |
+                     + \/
+---------------------+ /\
+ GRAPHICS PANEL #1   +  |
+                     +  |
+                     +  |
+                     +  |
+---------------------+  | PANEL 1
+ TEXT PANEL #1       +  |
+                     +  |
+---------------------+ \/
```

Parameters	- `width`: width of the Glcd panel - `height`: height of the Glcd panel - `fntW`: font width
Returns	Nothing.
Requires	Global variables: - `T6963C_dataPort`: Data Port - `T6963C_ctrlwr`: Write signal pin - `T6963C_ctrlrd`: Read signal pin - `T6963C_ctrlcd`: Command/Data signal pin - `T6963C_ctrlrst`: Reset signal pin - `T6963C_ctrlwr_Direction`: Direction of Write signal pin - `T6963C_ctrlrd_Direction`: Direction of Read signal pin - `T6963C_ctrlcd_Direction`: Direction of Command/Data signal pin - `T6963C_ctrlrst_Direction`: Direction of Reset signal pin must be defined before using this function.

Example

```
' T6963C module connections
dim T6963C_dataPort as byte at PORTB                 ' DATA port

dim T6963C_ctrlwr  as sbit at LATF2_bit              ' WR write signal
dim T6963C_ctrlrd  as sbit at LATF1_bit              ' RD read signal
dim T6963C_ctrlcd  as sbit at LATF0_bit         ' CD command/data signal
dim T6963C_ctrlrst as sbit at LATF4_bit              ' RST reset signal
dim T6963C_ctrlwr_Direction  as sbit at TRISF2_bit        ' WR write signal
direction
dim T6963C_ctrlrd_Direction  as sbit at TRISF1_bit        ' RD read signal
direction
dim T6963C_ctrlcd_Direction  as sbit at TRISF0_bit        ' CD command/data
signal direction
dim T6963C_ctrlrst_Direction as sbit at TRISF4_bit        ' RST reset signal
direction

' Signals not used by library, they are set in main function
dim T6963C_ctrlce as sbit at LATF3_bit                      ' CE signal
dim T6963C_ctrlfs as sbit at LATF6_bit                      ' FS signal
dim T6963C_ctrlmd as sbit at LATF5_bit                      ' MD signal
dim T6963C_ctrlce_Direction as sbit at TRISF3_bit    ' CE signal direction
dim T6963C_ctrlfs_Direction as sbit at TRISF6_bit    ' FS signal direction
dim T6963C_ctrlmd_Direction as sbit at TRISF5_bit    ' MD signal direction
' End T6963C module connections

...
' init display for 240 pixel width, 128 pixel height and 8 bits character
width
T6963C_init(240, 128, 8)
```

Notes	None.

T6963C_writeData

Prototype	`sub procedure T6963C_writeData(dim mydata as byte)`
Description	Writes data to T6963C controller.
Parameters	- `mydata`: data to be written
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_writeData(AddrL)`
Notes	None.

T6963C_writeCommand

Prototype	`sub procedure T6963C_writeCommand(dim mydata as byte)`
Description	Writes command to T6963C controller.
Parameters	- `mydata:` command to be written
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_writeCommand(T6963C_CURSOR_POINTER_SET)`
Notes	None.

T6963C_setPtr

Prototype	`sub procedure T6963C_setPtr(dim p as word, dim c as byte)`
Description	Sets the memory pointer `p` for command `p`.
Parameters	- `p`: address where command should be written - `c`: command to be written
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_setPtr(T6963C_grHomeAddr + start, T6963C_ADDRESS_POINTER_SET)`
Notes	None.

T6963C_waitReady

Prototype	`sub procedure T6963C_waitReady()`
Description	Pools the status byte, and loops until Toshiba Glcd module is ready.
Parameters	None.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_waitReady()`
Notes	None.

T6963C_fill

Prototype	`sub procedure T6963C_fill(dim v as byte, dim start, len as word)`
Description	Fills controller memory block with given byte.
Parameters	- `v`: byte to be written - `start`: starting address of the memory block - `len`: length of the memory block in bytes
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_fill(0x33,0x00FF,0x000F)`
Notes	None.

T6963C_dot

Prototype	`sub procedure T6963C_dot(dim x, y as integer, dim color as byte)`
Description	Draws a dot in the current graphic panel of Glcd at coordinates (x, y).
Parameters	- `x`: dot position on x-axis - `y`: dot position on y-axis - `color`: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_dot(x0, y0, pcolor)`
Notes	None.

T6963C_write_char

Prototype	`sub procedure T6963C_write_char(dim c, x, y, mode as byte)`
Description	Writes a char in the current text panel of Glcd at coordinates (x, y).
Parameters	- `c`: char to be written - `x`: char position on x-axis - `y`: char position on y-axis - `mode`: mode parameter. Valid values: T6963C_ROM_MODE_OR, T6963C_ROM_MODE_XOR, T6963C_ROM_MODE_AND and T6963C_ROM_MODE_TEXT Mode parameter explanation: - OR Mode: In the OR-Mode, text and graphics can be displayed and the data is logically “OR-ed”. This is the most common way of combining text and graphics for example labels on buttons. - XOR-Mode: In this mode, the text and graphics data are combined via the logical “exclusive OR”. This can be useful to display text in the negative mode, i.e. white text on black background. - AND-Mode: The text and graphic data shown on display are combined via the logical “AND function”. - TEXT-Mode: This option is only available when displaying just a text. The Text Attribute values are stored in the graphic area of display memory. For more details see the T6963C datasheet.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_write_char(“A”,22,23,T6963C_ROM_MODE_AND)`
Notes	None.

T6963C_write_text

Prototype	`sub procedure T6963C_write_char(dim byref str as byte[10], dim x, y, mode as byte)`
Description	Writes text in the current text panel of Glcd at coordinates (x, y).
Parameters	- `str`: text to be written - `x`: text position on x-axis - `y`: text position on y-axis - `mode`: mode parameter. Valid values: T6963C_ROM_MODE_OR, T6963C_ROM_MODE_XOR, T6963C_ROM_MODE_AND and T6963C_ROM_MODE_TEXT Mode parameter explanation: - OR Mode: In the OR-Mode, text and graphics can be displayed and the data is logically “OR-ed”. This is the most common way of combining text and graphics for example labels on buttons. - XOR-Mode: In this mode, the text and graphics data are combined via the logical “exclusive OR”. This can be useful to display text in the negative mode, i.e. white text on black background. - AND-Mode: The text and graphic data shown on display are combined via the logical “AND function”. - TEXT-Mode: This option is only available when displaying just a text. The Text Attribute values are stored in the graphic area of display memory. For more details see the T6963C datasheet.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_write_char(“GLCD LIBRARY DEMO, WELCOME !”, 0, 0, T6963C_ROM_MODE_XOR)`
Notes	None.

T6963C_line

Prototype	`sub procedure T6963C_line(dim x0, y0, x1, y1 as integer, dim pcolor as byte)`
Description	Draws a line from (x0, y0) to (x1, y1).
Parameters	- `x0`: x coordinate of the line start - `y0`: y coordinate of the line end - `x1`: x coordinate of the line start - `y1`: y coordinate of the line end - `pcolor`: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_line(0, 0, 239, 127, T6963C_WHITE)`
Notes	None.

T6963C_rectangle

Prototype	`sub procedure T6963C_rectangle(dim x0, y0, x1, y1 as integer, dim pcolor as byte)`
Description	Draws a rectangle on Glcd.
Parameters	- `x0`: x coordinate of the upper left rectangle corner - `y0`: y coordinate of the upper left rectangle corner - `x1`: x coordinate of the lower right rectangle corner - `y1`: y coordinate of the lower right rectangle corner - `pcolor`: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE)`
Notes	None.

T6963C_rectangle_round_edges

Prototype	`sub procedure T6963C_rectangle_round_edges(dim x0, y0, x1, y1, radius as integer, dim pcolor as byte)`
Description	Draws a rounded edge rectangle on Glcd.
Parameters	- `x0`: x coordinate of the upper left rectangle corner - `y0`: y coordinate of the upper left rectangle corner - `x1`: x coordinate of the lower right rectangle corner - `y1`: y coordinate of the lower right rectangle corner - `round_radius`: radius of the rounded edge. - `pcolor`: color parameter. Valid values: `T6963C_BLACK` and `T6963C_WHITE`
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_rectangle_round_edges(20, 20, 219, 107, 12, T6963C_WHITE)`
Notes	None.

T6963C_rectangle_round_edges_fill

Prototype	`sub procedure T6963C_rectangle_round_edges_fill(dim x0, y0, x1, y1, radius as integer, dim pcolor as byte)`
Description	Draws a filled rounded edge rectangle on Glcd.
Parameters	- `x0`: x coordinate of the upper left rectangle corner - `y0`: y coordinate of the upper left rectangle corner - `x1`: x coordinate of the lower right rectangle corner - `y1`: y coordinate of the lower right rectangle corner - `round_radius`: radius of the rounded edge - `pcolor`: color parameter. Valid values: `T6963C_BLACK` and `T6963C_WHITE`
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_rectangle_round_edges_fill(20, 20, 219, 107, 12, T6963C_WHITE)`
Notes	None.

T6963C_box

Prototype	`sub procedure T6963C_box(dim x0, y0, x1, y1 as integer, dim pcolor as byte)`
Description	Draws a box on Glcd
Parameters	- `x0`: x coordinate of the upper left box corner - `y0`: y coordinate of the upper left box corner - `x1`: x coordinate of the lower right box corner - `y1`: y coordinate of the lower right box corner - `pcolor`: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_box(0, 119, 239, 127, T6963C_WHITE)`
Notes	None.

T6963C_circle

Prototype	`sub procedure T6963C_circle(dim x, y as integer, dim r as longint, dim pcolor as word)`
Description	Draws a circle on Glcd.
Parameters	- `x`: x coordinate of the circle center - `y`: y coordinate of the circle center - `r`: radius size - `pcolor`: color parameter. Valid values: T6963C_BLACK and T6963C_WHITE
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_circle(120, 64, 110, T6963C_WHITE)`
Notes	None.

T6963C_circle_fill

Prototype	`sub procedure T6963C_circle_fill(dim x, y as integer, dim r as longint, dim pcolor as byte)`
Description	Draws a filled circle on Glcd.
Parameters	- `x`: x coordinate of the circle center - `y`: y coordinate of the circle center - `r`: radius size - `pcolor`: color parameter. Valid values: `T6963C_BLACK` and `T6963C_WHITE`
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_circle_fill(120, 64, 110, T6963C_WHITE)`
Notes	None.

T6963C_image

Prototype	`sub procedure T6963C_image(dim const pic as ^byte)`
Description	Displays bitmap on Glcd.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_image(my_image)`
Notes	Image dimension must match the display dimension. Use the integrated Glcd Bitmap Editor (menu option **Tools › Glcd Bitmap Editor**) to convert image to a constant array suitable for displaying on Glcd.

T6963C_PartialImage

Prototype	`sub procedure T6963C_PartialImage(dim x_left, y_top, width, height, picture_width, picture_height as word, const image as ^byte)`
Description	Displays a partial area of the image on a desired location.
Parameters	- `x_left`: x coordinate of the desired location (upper left coordinate). - `y_top`: y coordinate of the desired location (upper left coordinate). - `width`: desired image width. - `height`: desired image height. - `picture_width`: width of the original image. - `picture_height`: height of the original image. - `image`: image to be displayed. Bitmap array can be located in both code and RAM memory (due to the mikroBasic PRO for PIC pointer to const and pointer to RAM equivalency).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See T6963C_init routine.
Example	`' Draws a 10x15 part of the image starting from the upper left corner on the coordinate (10,12). Original image size is 16x32.` `T6963C_PartialImage(10, 12, 10, 15, 16, 32, @image)`
Notes	Image dimension must match the display dimension. Use the integrated Glcd Bitmap Editor (menu option **Tools › Glcd Bitmap Editor**) to convert image to a constant array suitable for displaying on Glcd.

T6963C_sprite

Prototype	`sub procedure T6963C_sprite(dim px, py as byte, dim const pic as ^byte, dim sx, sy as byte)`
Description	Fills graphic rectangle area (px, py) to (px+sx, py+sy) with custom size picture.
Parameters	- `px`: x coordinate of the upper left picture corner. Valid values: multiples of the font width - `py`: y coordinate of the upper left picture corner - `pic`: picture to be displayed - `sx`: picture width. Valid values: multiples of the font width - `sy`: picture height
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_sprite(76, 4, einstein, 88, 119)  ' draw a sprite`
Notes	If `px` and `sx` parameters are not multiples of the font width they will be scaled to the nearest lower number that is a multiple of the font width.

T6963C_set_cursor

Prototype	`sub procedure T6963C_set_cursor(dim x, y as byte)`
Description	Sets cursor to row x and column y.
Parameters	- `x`: cursor position row number - `y`: cursor position column number
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_set_cursor(cposx, cposy)`
Notes	None.

T6963C_displayGrPanel

Prototype	`sub procedure T6963C_displayGrPanel(dim n as word)`
Description	Display selected graphic panel.
Parameters	- `n`: graphic panel number. Valid values: `0` and `1`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' display graphic panel 1` `T6963C_displayGrPanel(1)`
Notes	None.

T6963C_displayTxtPanel

Prototype	`sub procedure T6963C_displayTxtPanel(dim n as word)`
Description	Display selected text panel.
Parameters	- `n`: text panel number. Valid values: `0` and `1`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' display text panel 1` `T6963C_displayTxtPanel(1)`
Notes	None.

T6963C_setGrPanel

Prototype	`sub procedure T6963C_setGrPanel(dim n as word)`
Description	Compute start address for selected graphic panel and set appropriate internal pointers. All subsequent graphic operations will be preformed at this graphic panel.
Parameters	- `n`: graphic panel number. Valid values: `0` and `1`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' set graphic panel 1 as current graphic panel.` `T6963C_setGrPanel(1)`
Notes	None.

T6963C_setTxtPanel

Prototype	`sub procedure T6963C_setTxtPanel(dim n as word)`
Description	Compute start address for selected text panel and set appropriate internal pointers. All subsequent text operations will be preformed at this text panel.
Parameters	- `n`: text panel number. Valid values: `0` and `1`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' set text panel 1 as current text panel.` `T6963C_setTxtPanel(1)`
Notes	None.

T6963C_panelFill

Prototype	`sub procedure T6963C_panelFill(dim v as word)`
Description	Fill current panel in full (graphic+text) with appropriate value (0 to clear).
Parameters	- `v`: value to fill panel with.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' clear current panel` `T6963C_panelFill(0)`
Notes	None.

T6963C_grFill

Prototype	`sub procedure T6963C_grFill(dim v as word)`
Description	Fill current graphic panel with appropriate value (0 to clear).
Parameters	- `v`: value to fill graphic panel with.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' clear current graphic panel` `T6963C_grFill(0)`
Notes	None.

T6963C_txtFill

Prototype	`sub procedure T6963C_txtFill(dim v as word)`
Description	Fill current text panel with appropriate value (0 to clear).
Parameters	- `v`: this value increased by 32 will be used to fill text panel.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' clear current text panel` `T6963C_txtFill(0)`
Notes	None.

T6963C_cursor_height

Prototype	`sub procedure T6963C_cursor_height(dim n as word)`
Description	Set cursor size.
Parameters	- `n`: cursor height. Valid values: `0..7`.
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`T6963C_Cursor_Height(7)`
Notes	None.

T6963C_graphics

Prototype	`sub procedure T6963C_graphics(dim n as word)`
Description	Enable/disable graphic displaying.
Parameters	- `n`: graphic enable/disable parameter. Valid values: `0` (disable graphic dispaying) and `1` (enable graphic displaying).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' enable graphic displaying` `T6963C_graphics(1)`
Notes	None.

T6963C_text

Prototype	`sub procedure T6963C_text(dim n as word)`
Description	Enable/disable text displaying.
Parameters	- `n`: on/off parameter. Valid values: `0` (disable text displaying) and `1` (enable text displaying).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' enable text displaying` `T6963C_text(1)`
Notes	None.

T6963C_cursor

Prototype	`sub procedure T6963C_cursor(dim n as word)`
Description	Set cursor on/off.
Parameters	- `n`: on/off parameter. Valid values: `0` (set cursor off) and `1` (set cursor on).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' set cursor on` `T6963C_cursor(1)`
Notes	None.

T6963C_cursor_blink

Prototype	`sub procedure T6963C_cursor_blink(dim n as word)`
Description	Enable/disable cursor blinking.
Parameters	- `n`: cursor blinking enable/disable parameter. Valid values: `0` (disable cursor blinking) and `1` (enable cursor blinking).
Returns	Nothing.
Requires	Toshiba Glcd module needs to be initialized. See the T6963C_init routine.
Example	`' enable cursor blinking` `T6963C_cursor_blink(1)`
Notes	None.

Library Example

The following drawing demo tests advanced routines of the T6963C Glcd library. Hardware configurations in this example are made for the EasydsPIC4A board and dsPIC30F4013.

Copy Code To Clipboard

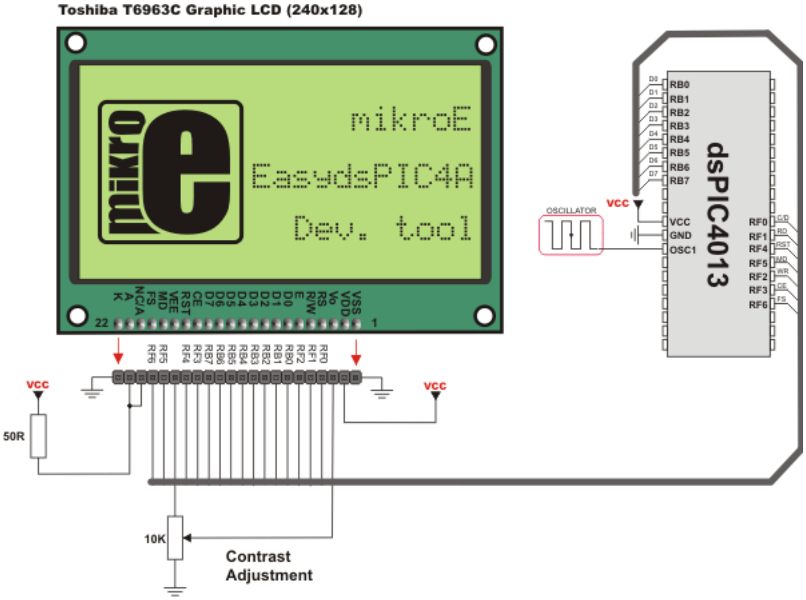
```
program T6963C_240x128

include __Lib_T6963C_Consts

' T6963C module connections
dim T6963C_dataPort as byte at PORTB                           ' DATA port

dim T6963C_ctrlwr  as sbit at LATF2_bit                         ' WR write signal
dim T6963C_ctrlrd  as sbit at LATF1_bit                         ' RD read signal
dim T6963C_ctrlcd  as sbit at LATF0_bit                         ' CD command/data signal
dim T6963C_ctrlrst as sbit at LATF4_bit                         ' RST reset signal
dim T6963C_ctrlwr_Direction  as sbit at TRISF2_bit              ' WR write signal direction
dim T6963C_ctrlrd_Direction  as sbit at TRISF1_bit              ' RD read signal direction
dim T6963C_ctrlcd_Direction  as sbit at TRISF0_bit   ' CD command/data signal direction
dim T6963C_ctrlrst_Direction as sbit at TRISF4_bit   ' RST reset signal direction

' Signals not used by library, they are set in main function
dim T6963C_ctrlce as sbit at LATF3_bit                          ' CE signal
dim T6963C_ctrlfs as sbit at LATF6_bit                          ' FS signal
dim T6963C_ctrlmd as sbit at LATF5_bit                          ' MD signal
dim T6963C_ctrlce_Direction as sbit at TRISF3_bit               ' CE signal direction
dim T6963C_ctrlfs_Direction as sbit at TRISF6_bit               ' FS signal direction
dim T6963C_ctrlmd_Direction as sbit at TRISF5_bit               ' MD signal direction
' End T6963C module connections

dim   panel as byte          ' current panel
          i as word          ' general purpose register
      curs  as byte          ' cursor visibility
      cposx,
      cposy as word          ' cursor x-y position
```

```
 txt, txt1 as string[29]

main:

  txt1 = " EINSTEIN WOULD HAVE LIKED mE"
  txt  = " GLCD LIBRARY DEMO, WELCOME !"

  #DEFINE COMPLETE_EXAMPLE      ' comment this line to make simpler/smaller example
  ADPCFG = 0xFFFF               ' initialize AN pins as digital

  TRISB8_bit = 1                ' Set RB8 as input
  TRISB9_bit = 1                ' Set RB9 as input
  TRISB10_bit = 1               ' Set RB10 as input
  TRISB11_bit = 1               ' Set RB11 as input
  TRISB12_bit = 1               ' Set RB12 as input

  T6963C_ctrlce_Direction = 0
  T6963C_ctrlce = 0              ' Enable T6963C
  T6963C_ctrlfs_Direction = 0
  T6963C_ctrlfs = 0              ' Font Select 8x8
  T6963C_ctrlmd_Direction = 0
  T6963C_ctrlmd = 0              ' Column number select

  panel = 0
  i = 0
  curs = 0
  cposx = 0
  cposy = 0

   ' Initialize T6369C
  T6963C_init(240, 128, 8)

'      * Enable both graphics and text display at the same time

  T6963C_graphics(1)
  T6963C_text(1)

'      * Text messages
  T6963C_write_text(txt, 0, 0, T6963C_ROM_MODE_XOR)
  T6963C_write_text(txt1, 0, 15, T6963C_ROM_MODE_XOR)

'      * Cursor
  T6963C_cursor_height(8)        ' 8 pixel height
  T6963C_set_cursor(0, 0)        ' Move cursor to top left
  T6963C_cursor(0)               ' Cursor off

'      * Draw rectangles
  T6963C_rectangle(0, 0, 239, 127, T6963C_WHITE)
  T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE)
  T6963C_rectangle(40, 40, 199, 87, T6963C_WHITE)
  T6963C_rectangle(60, 60, 179, 67, T6963C_WHITE)
```

```
'     * Draw a cross
  T6963C_line(0, 0, 239, 127, T6963C_WHITE)
  T6963C_line(0, 127, 239, 0, T6963C_WHITE)

'     * Draw solid boxes
  T6963C_box(0, 0, 239, 8, T6963C_WHITE)
  T6963C_box(0, 119, 239, 127, T6963C_WHITE)

'     * Draw circles
  #IFDEF COMPLETE_EXAMPLE
  T6963C_circle(120, 64, 10, T6963C_WHITE)
  T6963C_circle(120, 64, 30, T6963C_WHITE)
  T6963C_circle(120, 64, 50, T6963C_WHITE)
  T6963C_circle(120, 64, 70, T6963C_WHITE)
  T6963C_circle(120, 64, 90, T6963C_WHITE)
  T6963C_circle(120, 64, 110, T6963C_WHITE)
  T6963C_circle(120, 64, 130, T6963C_WHITE)

  T6963C_sprite(76, 4, @einstein, 88, 119)       ' Draw a sprite

  T6963C_setGrPanel(1)                           ' Select other graphic panel

  T6963C_image(@mikroE_240x128_bmp)       ' Fill the graphic screen with a picture
  #ENDIF

  while (TRUE)                            ' Endless loop

'     * If RB8 is pressed, toggle the display between graphic panel 0 and graphic 1
    if(RB8_bit <> 0) then
      T6963C_graphics(1)
      T6963C_text(0)
      Delay_ms(300)

'           * If RB9 is pressed, display only graphic panel
    #IFDEF COMPLETE_EXAMPLE
    else
      if (RB9_bit <> 0) then
        Inc(panel)
        panel = panel and 1
        T6963C_displayGrPanel(panel)
        Delay_ms(300)
    #ENDIF

'           * If RB10 is pressed, display only text panel
      else
        if (RB10_bit <> 0) then
          T6963C_graphics(0)
          T6963C_text(1)
          Delay_ms(300)

'           * If RB11 is pressed, display text and graphic panels
        else
```

```
      if (RB11_bit <> 0) then
            T6963C_graphics(1)
            T6963C_text(1)
            Delay_ms(300)

'           * If RB12 is pressed, change cursor
          else
            if(RB12_bit <> 0) then
              Inc(curs)
              if (curs = 3) then
                curs = 0
              end if

              select case curs
                case 0
                   ' no cursor
                   T6963C_cursor(0)

                case 1
                   ' blinking cursor
                   T6963C_cursor(1)
                   T6963C_cursor_blink(1)
               case 2
                   ' non blinking cursor
                   T6963C_cursor(1)
                   T6963C_cursor_blink(0)
              end select

              Delay_ms(300)
            end if
          end if
        end if
      #IFDEF COMPLETE_EXAMPLE
      end if
      #ENDIF
    end if

'     * Move cursor, even if not visible
    Inc(cposx)

    if (cposx = T6963C_txtCols) then
      cposx = 0
      Inc(cposy)
      if (cposy = T6963C_grHeight div T6963C_CHARACTER_HEIGHT) then
        cposy = 0
      end if
    end if
    T6963C_set_cursor(cposx, cposy)

    Delay_ms(100)
  wend
end.
```

HW Connection



T6963C Glcd HW connection

TFT Library

Thin film transistor liquid crystal display (TFT-LCD) is a variant of liquid crystal display (LCD) which uses thin-film transistor (TFT) technology to improve image quality (e.g., addressability, contrast).
TFT LCD is one type of active matrix LCD, though all LCD-screens are based on TFT active matrix addressing.
TFT LCDs are used in television sets, computer monitors, mobile phones, handheld video game systems, personal digital assistants, navigation systems, projectors, etc.

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for working with HX8347-D 320x240 TFT Lcd controller.The HX8347-D is designed to provide a single-chip solution that combines a gate driver, a source driver, power supply circuit for 262,144 colors to drive a TFT panel with 320x240 dots at maximum.

The HX8347-D is suitable for any small portable battery-driven and long-term driving products, such as small PDAs, digital cellular phones and bi-directional pagers.

External dependencies of TFT Library

The following variables must be defined in all projects using TFT library:	**Description:**	**Example:**
`dim TFT_DataPort as byte sfr external`	TFT Data Port.	`dim TFT_DataPort as byte at LATE`
`dim TFT_DataPort_Direction as byte sfr external`	Direction of the TFT Data Port.	`dim TFT_DataPort_Direction as byte at TRISE`
`dim TFT_WR as byte sfr external`	Write signal.	`dim TFT_WR as sbit at LATD13_bit`
`dim TFT_RD as byte sfr external`	Read signal.	`dim TFT_RD as sbit at LATD12_bit`
`dim TFT_CS as byte sfr external`	Chip Select signal.	`dim TFT_CS as sbit at LATC3_bit`
`dim TFT_RS as byte sfr external`	Command/Register Select signal.	`dim TFT_RS as sbit at LATB15_bit`
`dim TFT_RST as byte sfr external`	Reset signal.	`dim TFT_RST as sbit at LATC1_bit`
`dim TFT_WR_Direction as byte sfr external`	Direction of the Write pin.	`dim TFT_WR_Direction as sbit at TRISD13_bit`
`dim TFT_RD_Direction as byte sfr external`	Direction of the Read pin.	`dim TFT_WR_Direction as sbit at TRISD12_bit`
`dim TFT_CS_Direction as byte sfr external`	Direction of the Chip Select pin.	`dim TFT_CS_Direction as sbit at TRISC3_bit`
`dim TFT_RS_Direction as byte sfr external`	Direction of the Register Select pin.	`dim TFT_RS_Direction as sbit at TRISB13_bit`
`dim TFT_RST_Direction as byte sfr external`	Direction of the Reset pin.	`dim TFT_RST_Direction as sbit at TRISC1_bit`

Library Routines

- TFT_Init
- TFT_Set_Index
- TFT_Write_Command
- TFT_Write_Data
- TFT_Set_Active
- TFT_Set_Font
- TFT_Write_Char
- TFT_Write_Text
- TFT_Fill_Screen
- TFT_Set_Pen
- TFT_Set_Brush
- TFT_Dot
- TFT_Line
- TFT_H_Line
- TFT_V_Line
- TFT_Rectangle
- TFT_Rectangle_Round_Edges
- TFT_Circle
- TFT_Image
- TFT_PartialImage
- TFT_Image_Jpeg
- TFT_RGBToColor16bit
- TFT_Color16bitToRGB

TFT_Init

Prototype	`sub procedure TFT_Init(dim display_width, display_height as word)`
Returns	Nothing
Description	Initializes TFT display in the 8-bit working mode. Parameters: - `width`: width of the TFT panel - `height`: height of the TFT panel
Requires	Global variables: - `TFT_DataPort`: Data Port - `TFT_WR`: Write signal pin - `TFT_RD`: Read signal pin - `TFT_CS`: Chip Select signal pin - `TFT_RS`: Register Select signal pin - `TFT_RST`: Reset signal pin - `TFT_DataPort_Direction`: Direction of Data Port - `TFT_WR_Direction`: Direction of Write signal pin - `TFT_RD_Direction`: Direction of Read signal pin - `TFT_CS_Direction`: Direction of Chip Select signal pin - `TFT_RS_Direction`: Direction of Register Select signal pin - `TFT_RST_Direction`: Direction of Reset signal pin must be defined before using this function.

Example

```
' TFT display connections
dim TFT_DataPort as byte at LATE
dim TFT_WR as sbit at LATD13_bit
dim TFT_RD as sbit at LATD12_bit
dim TFT_CS as sbit at LATC3_bit
dim TFT_RS as sbit at LATB15_bit
dim TFT_RST as sbit at LATC1_bit

dim TFT_DataPort_Direction as byte at TRISE
dim TFT_WR_Direction as sbit at TRISD13_bit
dim TFT_RD_Direction as sbit at TRISD12_bit
dim TFT_CS_Direction as sbit at TRISC3_bit
dim TFT_RS_Direction as sbit at TRISB15_bit
dim TFT_RST_Direction as sbit at TRISC1_bit
' End of TFT display connections

' Initialize 240x320 TFT display
TFT_Init(240, 320)
```

TFT_Set_Index

Prototype	`sub procedure TFT_Set_Index(dim index as byte)`
Returns	Nothing
Description	Accesses register space of the controller and sets the desired register. Parameters: - `index`: desired register number.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`' Access register at the location 0x02` `TFT_Set_Index(0x02)`

TFT_Write_Command

Prototype	`sub procedure TFT_Write_Command(dim cmd as byte)`
Returns	Nothing
Description	Accesses data space and writes a command. Parameters: - `cmd`: command to be written.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`' Write a command` `TFT_Write_Command(0x02)`

TFT_Write_Data

Prototype	`sub procedure TFT_Write_Data(dim _data as word)`
Returns	Nothing
Description	Writes date into display memory. Parameters: - `_data`:data to be written.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`' Send data` `TFT_Write_Data(0x02)`

TFT_Set_Active

Prototype	`sub procedure TFT_Set_Active(dim Set_Index_Ptr as ^TTFT_Set_Index_Ptr, dim Write_Command_Ptr as ^TTFT_Write_Command_Ptr, dim Write_Data_Ptr as ^TTFT_Write_Data_Ptr)`
Returns	Nothing
Description	This function sets appropriate pointers to a user-defined basic routines in order to enable multiple working modes. Parameters: - `Set_Index_Ptr`: Set_Index handler. - `Write_Command_Ptr`: _Write_Command handler. - `Write_Data_Ptr`: Write_Data handler.
Requires	None.

Example

```
' Example of establishing 16-bit communication between TFT display and
PORTD, PORTE of MCU :

sub procedure Set_Index(dim index as byte)
  TFT_RS = 0
  Lo(LATD) = index
  TFT_WR = 0
  TFT_WR = 1
end sub

sub procedure Write_Command(dim cmd as byte)
  TFT_RS = 1
  Lo(LATD) = cmd
  TFT_WR = 0
  TFT_WR = 1
end sub

sub procedure Write_Data(dim _data as byte)
  TFT_RS = 1
  Lo(LATE) = Hi(_data)
  Lo(LATD) = Lo(_data)
  TFT_WR = 0
  TFT_WR = 1
end sub

main :
  TRISE = 0
  TRISD = 0

  TFT_Set_Active(Set_Index,Write_Command,Write_Data)
  TFT_Init(320, 240)

  .....
end.
```

TFT_Set_Font

Prototype	`sub procedure TFT_Set_Font(dim activeFont as ^far const byte, dim font_color as word, dim font_orientation as byte)`
Returns	Nothing
Description	Sets font, its color and font orientation. Parameters: - `activeFont`: desired font. Currently, only `TFT_defaultFont` (Tahoma14x16) is supported. - `font_color`: sets font color: (see table below) - `font_orientation`: sets font orientation: (see table below)
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Set_Font(@TFT_defaultFont, CL_BLACK, FO_HORIZONTAL)`

Value	Description
`CL_AQUA`	Aqua color
`CL_BLACK`	Black color
`CL_BLUE`	Blue color
`CL_FUCHSIA`	Fuchsia color
`CL_GRAY`	Gray color
`CL_GREEN`	Green color
`CL_LIME`	Lime color
`CL_MAROON`	Maroon color
`CL_NAVY`	Navy color
`CL_OLIVE`	Olive color
`CL_PURPLE`	Purple color
`CL_RED`	Red color
`CL_SILVER`	Silver color
`CL_TEAL`	Teal color
`CL_WHITE`	White color
`CL_YELLOW`	Yellow color

Value	Description
`FO_HORIZONTAL`	Horizontal orientation
`FO_VERTICAL`	Vertical orientation

TFT_Write_Char

Prototype	`sub procedure TFT_Write_Char(dim ch, x, y as word)`
Returns	Nothing.
Description	Writes a char on the TFT at coordinates (x, y). - `c`: char to be written. - `x`: char position on x-axis. - `y`: char position on y-axis.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Write_Char("A",22,23)`

TFT_Write_Text

Prototype	`sub procedure TFT_Write_Text(dim byref text as string, dim x, y as word)`
Returns	Nothing.
Description	Writes text on the TFT at coordinates (x, y). Parameters: - `text`: text to be written. - `x`: text position on x-axis. - `y`: text position on y-axis.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Write_Text("TFT LIBRARY DEMO, WELCOME !", 0, 0)`

TFT_Fill_Screen

<table>
<tr><td>Prototype</td><td><code>sub procedure TFT_Fill_Screen(dim color as word)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Fills screen memory block with given color.

Parameters :

- <code>color</code>: color to be filled:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>CL_AQUA</code></td><td>Aqua color</td></tr>
<tr><td><code>CL_BLACK</code></td><td>Black color</td></tr>
<tr><td><code>CL_BLUE</code></td><td>Blue color</td></tr>
<tr><td><code>CL_FUCHSIA</code></td><td>Fuchsia color</td></tr>
<tr><td><code>CL_GRAY</code></td><td>Gray color</td></tr>
<tr><td><code>CL_GREEN</code></td><td>Green color</td></tr>
<tr><td><code>CL_LIME</code></td><td>Lime color</td></tr>
<tr><td><code>CL_MAROON</code></td><td>Maroon color</td></tr>
<tr><td><code>CL_NAVY</code></td><td>Navy color</td></tr>
<tr><td><code>CL_OLIVE</code></td><td>Olive color</td></tr>
<tr><td><code>CL_PURPLE</code></td><td>Purple color</td></tr>
<tr><td><code>CL_RED</code></td><td>Red color</td></tr>
<tr><td><code>CL_SILVER</code></td><td>Silver color</td></tr>
<tr><td><code>CL_TEAL</code></td><td>Teal color</td></tr>
<tr><td><code>CL_WHITE</code></td><td>White color</td></tr>
<tr><td><code>CL_YELLOW</code></td><td>Yellow color</td></tr>
</table>
</td></tr>
<tr><td>Requires</td><td>TFT module needs to be initialized. See the TFT_Init routine.</td></tr>
<tr><td>Example</td><td><code>TFT_Fill_Screen(CL_BLACK)</code></td></tr>
</table>

TFT_Dot

Prototype	`sub procedure TFT_Dot(dim x, y as integer, dim color as word)`
Returns	Nothing.
Description	Draws a dot on the TFT at coordinates (x, y). Parameters: - `x`: dot position on x-axis. - `y`: dot position on y-axis. - `color`: color parameter. Valid values: (see table below)
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Dot(50, 50, CL_BLACK)`

Value	**Description**
`CL_AQUA`	Aqua color
`CL_BLACK`	Black color
`CL_BLUE`	Blue color
`CL_FUCHSIA`	Fuchsia color
`CL_GRAY`	Gray color
`CL_GREEN`	Green color
`CL_LIME`	Lime color
`CL_MAROON`	Maroon color
`CL_NAVY`	Navy color
`CL_OLIVE`	Olive color
`CL_PURPLE`	Purple color
`CL_RED`	Red color
`CL_SILVER`	Silver color
`CL_TEAL`	Teal color
`CL_WHITE`	White color
`CL_YELLOW`	Yellow color

TFT_Set_Pen

Prototype	`sub procedure TFT_Set_Pen(dim pen_color as word, dim pen_width as byte)`
Returns	Nothing.
Description	Sets color and thickness parameter for drawing line, circle and rectangle elements. Parameters: - `pen_color`: Sets color. **Value** – **Description** `CL_AQUA` – Aqua color `CL_BLACK` – Black color `CL_BLUE` – Blue color `CL_FUCHSIA` – Fuchsia color `CL_GRAY` – Gray color `CL_GREEN` – Green color `CL_LIME` – Lime color `CL_MAROON` – Maroon color `CL_NAVY` – Navy color `CL_OLIVE` – Olive color `CL_PURPLE` – Purple color `CL_RED` – Red color `CL_SILVER` – Silver color `CL_TEAL` – Teal color `CL_WHITE` – White color `CL_YELLOW` – Yellow color - `pen_width`: sets thickness.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Set_Pen(CL_BLACK, 10)`

TFT_Set_Brush

<table>
<tr><td>Prototype</td><td><code>sub procedure TFT_Set_Brush(dim brush_enabled as byte, dim brush_color as word, dim gradient_enabled, gradient_orientation as byte, dim gradient_color_from, gradient_color_to as word)</code></td></tr>
<tr><td>Returns</td><td>Nothing.</td></tr>
<tr><td>Description</td><td>Sets color and gradient which will be used to fill circles or rectangles.

Parameters:

- <code>brush_enabled</code>: enable brush fill.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>1</code></td><td>Enable brush fill.</td></tr>
<tr><td><code>0</code></td><td>Disable brush fill.</td></tr>
</table>

- <code>brush_color</code>: set brush fill color.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>CL_AQUA</code></td><td>Aqua color</td></tr>
<tr><td><code>CL_BLACK</code></td><td>Black color</td></tr>
<tr><td><code>CL_BLUE</code></td><td>Blue color</td></tr>
<tr><td><code>CL_FUCHSIA</code></td><td>Fuchsia color</td></tr>
<tr><td><code>CL_GRAY</code></td><td>Gray color</td></tr>
<tr><td><code>CL_GREEN</code></td><td>Green color</td></tr>
<tr><td><code>CL_LIME</code></td><td>Lime color</td></tr>
<tr><td><code>CL_MAROON</code></td><td>Maroon color</td></tr>
<tr><td><code>CL_NAVY</code></td><td>Navy color</td></tr>
<tr><td><code>CL_OLIVE</code></td><td>Olive color</td></tr>
<tr><td><code>CL_PURPLE</code></td><td>Purple color</td></tr>
<tr><td><code>CL_RED</code></td><td>Red color</td></tr>
<tr><td><code>CL_SILVER</code></td><td>Silver color</td></tr>
<tr><td><code>CL_TEAL</code></td><td>Teal color</td></tr>
<tr><td><code>CL_WHITE</code></td><td>White color</td></tr>
<tr><td><code>CL_YELLOW</code></td><td>Yellow color</td></tr>
</table>
</td></tr>
</table>

Description

- `gradient_enabled`: enable gradient

Value	Description
`1`	Enable gradient.
`0`	Disable gradient.

- `gradient_orientation`: sets gradient orientation:

Value	Description
`LEFT_TO_RIGHT`	Left to right gradient orientation
`TOP_TO_BOTTOM`	Top to bottom gradient orientation

- `gradient_color_from`: sets the starting gradient color.

Value	Description
`CL_AQUA`	Aqua color
`CL_BLACK`	Black color
`CL_BLUE`	Blue color
`CL_FUCHSIA`	Fuchsia color
`CL_GRAY`	Gray color
`CL_GREEN`	Green color
`CL_LIME`	Lime color
`CL_MAROON`	Maroon color
`CL_NAVY`	Navy color
`CL_OLIVE`	Olive color
`CL_PURPLE`	Purple color
`CL_RED`	Red color
`CL_SILVER`	Silver color
`CL_TEAL`	Teal color
`CL_WHITE`	White color
`CL_YELLOW`	Yellow color

Description	- `gradient_color_to`: sets the ending gradient color. Value / Description: `CL_AQUA` Aqua color `CL_BLACK` Black color `CL_BLUE` Blue color `CL_FUCHSIA` Fuchsia color `CL_GRAY` Gray color `CL_GREEN` Green color `CL_LIME` Lime color `CL_MAROON` Maroon color `CL_NAVY` Navy color `CL_OLIVE` Olive color `CL_PURPLE` Purple color `CL_RED` Red color `CL_SILVER` Silver color `CL_TEAL` Teal color `CL_WHITE` White color `CL_YELLOW` Yellow color
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`' Enable gradient from black to white color, left-right orientation` `TFT_Set_Brush(0, 0, 1, LEFT_TO_RIGHT, CL_BLACK, CL_WHITE)`

Value	Description
`CL_AQUA`	Aqua color
`CL_BLACK`	Black color
`CL_BLUE`	Blue color
`CL_FUCHSIA`	Fuchsia color
`CL_GRAY`	Gray color
`CL_GREEN`	Green color
`CL_LIME`	Lime color
`CL_MAROON`	Maroon color
`CL_NAVY`	Navy color
`CL_OLIVE`	Olive color
`CL_PURPLE`	Purple color
`CL_RED`	Red color
`CL_SILVER`	Silver color
`CL_TEAL`	Teal color
`CL_WHITE`	White color
`CL_YELLOW`	Yellow color

TFT_Line

Prototype	`sub procedure TFT_Line(dim x1, y1, x2, y2 as integer)`
Returns	Nothing.
Description	Draws a line from (x1, y1) to (x2, y2). Parameters: - `x1`: x coordinate of the line start. - `y1`: y coordinate of the line end. - `x2`: x coordinate of the line start. - `y2`: y coordinate of the line end.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Line(0, 0, 239, 127)`

TFT_H_Line

Prototype	`sub procedure TFT_H_Line(dim x_start, x_end, y_pos as integer)`
Returns	Nothing.
Description	Draws a horizontal line on TFT. Parameters: - `x_start`: x coordinate of the line start. - `x_end`: x coordinate of the line end. - `y_pos`: y coordinate of horizontal line.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`' Draw a horizontal line between dots (10,20) and (50,20)` `TFT_H_Line(10, 50, 20)`

TFT_V_Line

Prototype	`sub procedure TFT_V_Line(dim y_start, y_end, x_pos as integer)`
Returns	Nothing.
Description	Draws a vertical line on TFT. Parameters: - `y_start`: y coordinate of the line start. - `y_end`: y coordinate of the line end. - `x_pos`: x coordinate of vertical line.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`' Draw a vertical line between dots (10,5) and (10,25)` `TFT_V_Line(5, 25, 10)`

TFT_Rectangle

Prototype	`sub procedure TFT_Rectangle(dim x_upper_left, y_upper_left, x_bottom_right, y_bottom_right as integer)`
Returns	Nothing.
Description	Draws a rectangle on TFT. Parameters: - `x_upper_left:` x coordinate of the upper left rectangle corner. - `y_upper_left:` y coordinate of the upper left rectangle corner. - `x_bottom_right:` x coordinate of the lower right rectangle corner. - `y_bottom_right:` y coordinate of the lower right rectangle corner.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Rectangle(20, 20, 219, 107)`

TFT_Rectangle_Round_Edges

Prototype	`sub procedure TFT_Rectangle_Round_Edges(dim x_upper_left, y_upper_left, x_bottom_right, y_bottom_right, round_radius as word)`
Returns	Nothing.
Description	Draws a rounded edge rectangle on TFT. Parameters: - `x_upper_left`: x coordinate of the upper left rectangle corner. - `y_upper_left`: y coordinate of the upper left rectangle corner. - `x_bottom_right`: x coordinate of the lower right rectangle corner. - `y_bottom_right`: y coordinate of the lower right rectangle corner. - `round_radius`: radius of the rounded edge.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Rectangle_Round_Edges(20, 20, 219, 107, 12)`

TFT_Circle

Prototype	`sub procedure TFT_Circle(dim x_center, y_center, radius as integer)`
Returns	Nothing.
Description	Draws a circle on TFT. Parameters: - `x`: x coordinate of the circle center. - `y`: y coordinate of the circle center. - `r`: radius size.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Circle(120, 64, 110)`

TFT_Image

Prototype	`sub procedure TFT_Image(dim left, top as word, dim image as far const byte, dim stretch as byte)`
Returns	Nothing.
Description	Displays an image on a desired location. Parameters: - `left`: position of the image's left edge. - `top`:position of the image's top edge. - `image`: image to be displayed. Bitmap array is located in code memory. - `stretch`: stretches image by a given factor (if 2, it will double the image.).
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Image(0, 0, @image, 1)`

TFT_Partial_Image

Prototype	`sub procedure TFT_Partial_Image(dim left, top, width, height as word, dim image as far const byte, dim stretch as byte)`
Returns	Nothing.
Description	Displays a partial area of the image on a desired location. Parameters: - `left`: left coordinate of the image. - `top`: top coordinate of the image. - `width`: desired image width. - `height`: desired image height. - `image`: image to be displayed. Bitmap array is located in code memory. - `stretch`: stretches image by a given factor (if 2, it will double the image.).
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`' Draws a 10x15 part of the image starting from the upper left corner on the coordinate (10,12)` `TFT_PartialImage(10, 12, 10, 15, @image, 1)`

TFT_Image_Jpeg

Prototype	`sub function TFT_Image_Jpeg(dim left, top as word, dim image as far const byte) as byte`
Returns	- `0` - if image is loaded and displayed successfully. - `1` - if error occured.
Description	Displays a JPEG image on a desired location. Parameters: - `left`: left coordinate of the image. - `top`: top coordinate of the image. - `image`: image to be displayed. Bitmap array is located in code memory.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Image_Jpeg(0, 0, @image)`

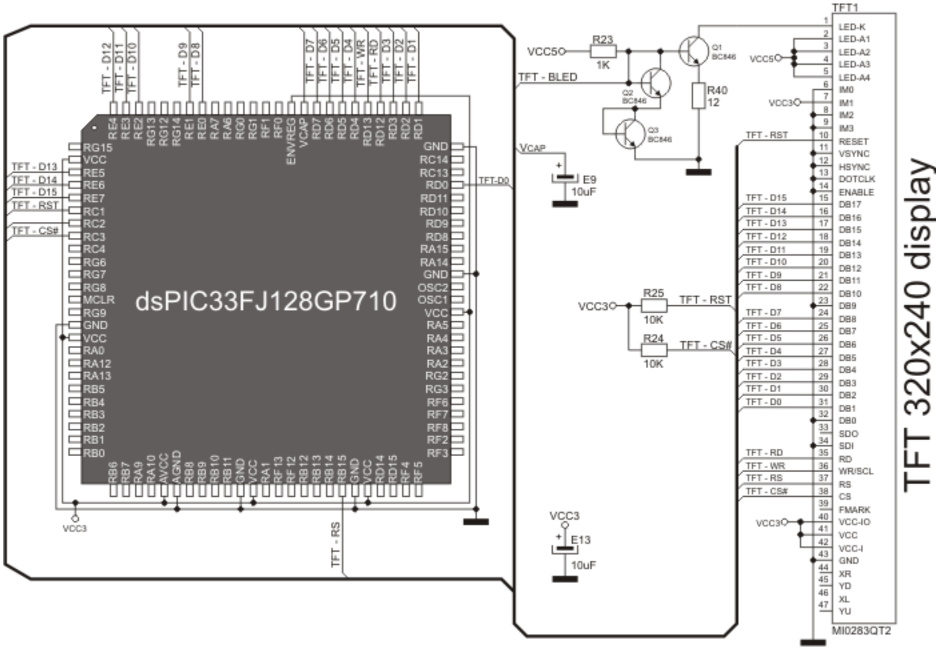
TFT_RGBToColor16bit

Prototype	`sub function TFT_RGBToColor16bit(dim rgb_red, rgb_green, rgb_blue as byte) as word`
Returns	Returns a color value in the following bit-order : 5 bits red, 6 bits green and 5 bits blue color.
Description	Converts 5:6:5 RGB format into true color format. Parameters: - `rgb_red`: red component of the image. - `rgb_green`: green component of the image. - `rgb_blue`: blue component of the image.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`color16 = TFT_Image_Jpeg(150, 193, 65)`

TFT_Color16bitToRGB

Prototype	`sub procedure TFT_Color16bitToRGB(dim color as word, dim rgb_red, rgb_green, rgb_blue as ^byte)`
Returns	Nothing.
Description	Converts true color into 5:6:5 RGB format. Parameters: - `color`: true color to be converted. - `rgb_red`: red component of the input color. - `rgb_green`: green component of the input color. - `rgb_blue`: blue component of the input color.
Requires	TFT module needs to be initialized. See the TFT_Init routine.
Example	`TFT_Color16bitToRGB(start_color, @red_start, @green_start, @blue_start)`

HW Connection



TFT HW connection

Touch Panel Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for working with Touch Panel.

Library Dependency Tree

External dependencies of Touch Panel Library

The following variables must be defined in all projects using Touch Panel Library:	Description:	Example:
`dim DriveA as sbit sfr external`	DriveA line.	`dim DriveA as sbit at LATC13_bit`
`dim DriveB as sbit sfr external`	DriveB line.	`dim DriveB as sbit at LATC14_bit`
`dim DriveA_Direction as sbit sfr external`	Direction of the DriveA pin.	`dim DriveA_Direction as sbit at TRISC13_bit`
`dim DriveB_Direction as sbit sfr external`	Direction of the DriveB pin.	`dim DriveB_Direction as sbit at TRISC14_bit`

Library Routines

- TP_Init
- TP_Set_ADC_Threshold
- TP_Press_Detect
- TP_Get_Coordinates
- TP_Calibrate_Bottom_Left
- TP_Calibrate_Upper_Right
- TP_Get_Calibration_Consts
- TP_Set_Calibration_Consts

TP_Init

Prototype	`sub procedure TP_Init(dim display_width as word, dim display_height as word, dim readX_ChNo as byte, dim readY_ChNo as byte)`
Description	Initialize touch panel display. Default touch panel ADC threshold value is set to 3900.
Parameters	- `display_width:` set display width. - `display_height:` set display height. - `readX_ChNo:` read X coordinate from desired ADC channel. - `readY_ChNo:` read Y coordinate from desired ADC channel.
Returns	Nothing.
Requires	Before calling this function initialize ADC module.
Example	`ADC1_Init()                ' Initalize ADC module` `TP_Init(128, 64, 6, 7)     ' Initialize touch panel, dimensions 128x64`
Notes	None.

TP_Set_ADC_Threshold

Prototype	`sub procedure TP_Set_ADC_Threshold(dim threshold as word)`
Description	Set custom ADC threshold value, call this function after TP_Init.
Parameters	- `threshold`: custom ADC threshold value.
Returns	Nothing.
Requires	TP_Init has to be called before using this routine.
Example	`TP_Set_ADC_Threshold(3900)     ' Set touch panel ADC threshold`
Notes	None.

TP_Press_Detect

Prototype	`sub function TP_Press_Detect() as byte`
Description	Detects if the touch panel has been pressed.
Parameters	None.
Returns	- `1` - if touch panel is pressed. - `0` - otherwise.
Requires	Global variables: - `DriveA:` DriveA. - `DriveB:` DriveB. - `DriveA_Direction:` Direction of DriveA pin. - `DriveB_Direction:` Direction of DriveB pin. must be defined before using this function.

Example

```
' Touch Panel module connections
dim DriveA as sbit at LATC13_bit
    DriveB as sbit at LATC14_bit
    DriveA_Direction as sbit at TRISC13_bit
    DriveB_Direction as sbit at TRISC14_bit
' End Touch Panel module connections

if (TP_Press_Detect() <> 0) then
  ...
end if
```

Notes	None.

TP_Get_Coordinates

Prototype	`sub function TP_Get_Coordinates(dim byref x_coordinate as word, dim byref y_coordinate as word) as byte`
Description	Get touch panel coordinates and store them in `x_coordinate` and `y_coordinate` parameters.
Parameters	- `x_coordinate:` x coordinate of the place of touch. - `y_coordinate:` y coordinate of the place of touch.
Returns	- `1` - if reading is within display dimension range. - `0` - if reading is out of display dimension range.
Requires	Nothing.

Example

```
if (TP_Get_Coordinates(@x_coord, @y_coord) = 0) then
  ...
end if
```

Notes	None.

TP_Calibrate_Bottom_Left

Prototype	`sub procedure TP_Calibrate_Bottom_Left()`
Description	Calibrate bottom left corner of the touch Panel.
Parameters	None.
Returns	Nothing.
Requires	Nothing.
Example	`TP_Calibrate_Bottom_Left()     ' Calibration of bottom left corner`
Notes	None.

TP_Calibrate_Upper_Right

Prototype	`sub procedure TP_Calibrate_Upper_Right()`
Description	Calibrate upper right corner of the touch panel.
Parameters	None.
Returns	Nothing.
Requires	Nothing.
Example	`TP_Calibrate_Upper_Right()     ' Calibration of upper right corner`
Notes	None.

TP_Get_Calibration_Consts

Prototype	`sub procedure TP_Get_Calibration_Consts(dim byref x_min as word, dim byref x_max as word, dim byref y_min as word, dim byref y_max as word)`
Description	Gets calibration constants after calibration is done and stores them in `x_min, x_max, y_min` and `y_max` parameters.
Parameters	- `x_min`: x coordinate of the bottom left corner of the working area. - `x_max`: x coordinate of the upper right corner of the working area. - `y_min`: y coordinate of the bottom left corner of the working area. - `y_max`: y coordinate of the upper right corner of the working area.
Returns	Nothing.
Requires	Nothing.
Example	`TP_Get_Calibration_Consts(@x_min, @y_min, @x_max, @y_max)    ' Get calibration constants`
Notes	None.

TP_Set_Calibration_Consts

Prototype	`sub procedure TP_Set_Calibration_Consts(dim x_min as word, dim x_max as word, dim y_min as word, dim y_max as word)`
Description	Sets calibration constants.
Parameters	- `x_min`: x coordinate of the bottom left corner of the working area. - `x_max`: x coordinate of the upper right corner of the working area. - `y_min`: y coordinate of the bottom left corner of the working area. - `y_max`: y coordinate of the upper right corner of the working area.
Returns	Nothing.
Requires	Nothing.
Example	`TP_Set_Calibration_Consts(148, 3590, 519, 3370)   ' Set calibration constants`
Notes	None.

Library Example

The following drawing demo tests routines of the Touch Panel library:

Copy Code To Clipboard

```
program TouchPanelCalibrationAndWrite

' Glcd module connections
dim GLCD_D7 as sbit at RD3_bit
    GLCD_D6 as sbit at RD2_bit
    GLCD_D5 as sbit at RD1_bit
    GLCD_D4 as sbit at RD0_bit
    GLCD_D3 as sbit at RB3_bit
    GLCD_D2 as sbit at RB2_bit
    GLCD_D1 as sbit at RB1_bit
    GLCD_D0 as sbit at RB0_bit
    GLCD_D7_Direction as sbit at TRISD3_bit
    GLCD_D6_Direction as sbit at TRISD2_bit
    GLCD_D5_Direction as sbit at TRISD1_bit
    GLCD_D4_Direction as sbit at TRISD0_bit
    GLCD_D3_Direction as sbit at TRISB3_bit
    GLCD_D2_Direction as sbit at TRISB2_bit
    GLCD_D1_Direction as sbit at TRISB1_bit
    GLCD_D0_Direction as sbit at TRISB0_bit

dim GLCD_CS1 as sbit at LATB4_bit
    GLCD_CS2 as sbit at LATB5_bit
    GLCD_RS  as sbit at LATF0_bit
    GLCD_RW  as sbit at LATF1_bit
    GLCD_EN  as sbit at LATF4_bit
    GLCD_RST as sbit at LATF5_bit
```

```
dim GLCD_CS1_Direction as sbit at TRISB4_bit
    GLCD_CS2_Direction as sbit at TRISB5_bit
    GLCD_RS_Direction  as sbit at TRISF0_bit
    GLCD_RW_Direction  as sbit at TRISF1_bit
    GLCD_EN_Direction  as sbit at TRISF4_bit
    GLCD_RST_Direction as sbit at TRISF5_bit
' eEnd Glcd module connections

' Touch Panel module connections
dim DriveA as sbit at LATC13_bit
    DriveB as sbit at LATC14_bit
    DriveA_Direction as sbit at TRISC13_bit
    DriveB_Direction as sbit at TRISC14_bit
' end Touch Panel module connections

dim write_erase as bit
    pen_size as byte
    x_coord, y_coord as word
    write_msg, clear_msg, erase_msg as char[5]           ' GLCD menu messages

sub procedure Initialize()
  ADPCFG = 0xFF3F                          ' set AN6 and AN7 channel pins as analog

  DriveA_Direction = 0                                   ' Set DriveA pin as output
  DriveB_Direction = 0                                   ' Set DriveB pin as output

  Glcd_Init()                                            ' Initialize GLCD
  Glcd_Fill(0)                                           ' Clear GLCD

  ADC1_Init()                                            ' Initialize ADC
  TP_Init(128, 64, 6, 7)                                 ' Initialize touch panel
  TP_Set_ADC_Threshold(3900)                      ' Set touch panel ADC threshold
end sub

sub procedure Calibrate()
    Glcd_Dot(0,63,1)                                     ' Draw bottom left dot
    Glcd_Write_Text(“TOUCH BOTTOM LEFT”,12,3,1)
    TP_Calibrate_Bottom_Left()                   ' Calibration of bottom left corner
    Delay_ms(1000)

    Glcd_Dot(0,63,0)                                     ' Clear bottom left dot
    Glcd_Dot(127,0,1)                                    ' Draw upper right dot
    Glcd_Write_Text(“                 ”,12,3,1)
    Glcd_Write_Text(“TOUCH UPPER RIGHT”,12,4,1)
    TP_Calibrate_Upper_Right()                  ' Calibration of upper right corner
    Delay_ms(1000)
end sub

main:
  write_msg = “WRITE”
  clear_msg = “CLEAR”
  erase_msg = “ERASE”
```

```
  Initialize()

  Glcd_Fill(0)                                            ' Clear GLCD
  Glcd_Write_Text(“CALIBRATION”,12,3,1)
  Delay_ms(1000)
  Glcd_Fill(0)                                            ' Clear GLCD
  Calibrate()
  Glcd_Fill(0)

  Glcd_Write_Text(“WRITE ON SCREEN”, 20, 5, 1)
  Delay_ms(1000)
  Glcd_Fill(0)

  Glcd_V_Line(0,7,0,1)
  Glcd_Write_Text(clear_msg,1,0,0)
  Glcd_V_Line(0,7,97,1)
  Glcd_Write_Text(erase_msg,98,0,0)

  ' Pen Menu:

Glcd_Rectangle(41,0,52,9,1)
  Glcd_Box(45,3,48,6,1)
  Glcd_Rectangle(63,0,70,7,1)
  Glcd_Box(66,3,67,4,1)
  Glcd_Rectangle(80,0,86,6,1)
  Glcd_Dot(83,3,1)

  write_erase = 1
  pen_size = 1

  while (TRUE)
      if (TP_Press_Detect() <> 0) then
        ' After a PRESS is detected read X-Y and convert it to 128x64 space
        if (TP_Get_Coordinates(@x_coord, @y_coord) = 0) then
          if ((x_coord < 31) and (y_coord < 8)) then
            Glcd_Fill(0)

            ' Pen Menu:
            Glcd_Rectangle(41,0,52,9,1)
            Glcd_Box(45,3,48,6,1)
            Glcd_Rectangle(63,0,70,7,1)
            Glcd_Box(66,3,67,4,1)
            Glcd_Rectangle(80,0,86,6,1)
            Glcd_Dot(83,3,1)

            Glcd_V_Line(0,7,0,1)
            Glcd_Write_Text(clear_msg,1,0,0)
            Glcd_V_Line(0,7,97,1)
            if (write_erase) then
              Glcd_Write_Text(erase_msg,98,0,0)
            else
```

```
              Glcd_Write_Text(write_msg,98,0,0)
            end if
          end if

          ' If write/erase is pressed
          if ((x_coord > 96) and (y_coord < 8)) then
            if (write_erase) then
              write_erase = 0
              Glcd_Write_Text(write_msg,98,0,0)
              Delay_ms(500)
            else
              write_erase = 1
              Glcd_Write_Text(erase_msg,98,0,0)
              Delay_ms(500)
            end if

' If pen size is selected
          if ((x_coord >= 41) and (x_coord <= 52) and (y_coord <= 9)) then
            pen_size = 3
          end if

if ((x_coord >= 63) and (x_coord <= 70) and (y_coord <= 7)) then
            pen_size = 2
          end if

          if ((x_coord >= 80) and (x_coord <= 86) and (y_coord <= 6)) then
            pen_size = 1
          end if

          if (y_coord < 11) then
            continue
          end if

          select case pen_size
            case 1
              if ( (x_coord >= 0) and (y_coord >= 0) and (x_coord <= 127) and (y_coord
<= 63) ) then
                 Glcd_Dot(x_coord, y_coord, write_erase)
              end if

            case 2
             if ( (x_coord >= 0) and (y_coord >= 0) and (x_coord <= 127-1) and (y_coord
<= 63-1) ) then
                 Glcd_Box(x_coord, y_coord, x_coord + 1, y_coord + 1, write_erase)
              end if

             case 3
             if ( (x_coord >= 1) and (y_coord >= 1) and (x_coord <= 127-2) and (y_coord
<= 63-2) ) then
                 Glcd_Box(x_coord-1, y_coord-1, x_coord + 2, y_coord + 2, write_erase)
              end if
          end select

        end if
      end if
    wend
end.
```

Touch Panel TFT Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a library for working with Touch Panel for TFT.

Library Dependency Tree

External dependencies of Touch Panel TFT Library

The following variables must be defined in all projects using Touch Panel TFT Library:	Description:	Example:
`dim DriveX_Left as sbit sfr external`	DriveX_Left line.	`dim DriveX_Left as sbit at LATB13_bit`
`dim DriveX_Right as sbit sfr external`	DriveX_Right line.	`dim DriveX_Right as sbit at LATB11_bit`
`dim DriveY_Up as sbit sfr external`	DriveY_Up line.	`dim DriveY_Up as sbit at LATB12_bit`
`dim DriveY_Down as sbit sfr external`	DriveY_Down line.	`dim DriveY_Down as sbit at LATB10_bit`
`dim DriveX_Left_Direction as sbit sfr external`	Direction of the DriveX_Left pin.	`dim DriveX_Left_Direction as sbit at TRISB13_bit`
`dim DriveX_Right_Direction as sbit sfr external`	Direction of the DriveX_Right pin.	`dim DriveX_Right_Direction as sbit at TRISB11_bit`
`dim DriveY_Up_Direction as sbit sfr external`	Direction of the DriveY_Up pin.	`dim DriveY_Up_Direction as sbit at TRISB12_bit`
`dim DriveY_Down_Direction as sbit sfr external`	Direction of the DriveY_Down pin.	`dim DriveY_Down_Direction as sbit at TRISB10_bit`

Library Routines

- TP_TFT_Init
- TP_TFT_Set_ADC_Threshold
- TP_TFT_Press_Detect
- TP_TFT_Get_Coordinates
- TP_TFT_Calibrate_Min
- TP_TFT_Calibrate_Max
- TP_TFT_Get_Calibration_Consts
- TP_TFT_Set_Calibration_Consts

TP_TFT_Init

Prototype	`sub procedure TP_TFT_Inits(dim display_width as word, dim display_height as word, dim readX_ChNo as byte, dim readY_ChNo as byte)`
Description	Initialize TFT touch panel display. Default touch panel ADC threshold value is set to 900.
Parameters	- `display_width:` set display width. - `display_height:` set display height. - `readX_ChNo:` read X coordinate from desired ADC channel. - `readY_ChNo:` read Y coordinate from desired ADC channel.
Returns	Nothing.
Requires	Before calling this function initialize ADC module.
Example	`ADC1_Init()                      ' Initalize ADC module` `TP_TFT_Init(320, 240, 13, 12)  ' Initialize touch panel, dimensions 320x240`
Notes	None.

TP_TFT_Set_ADC_Threshold

Prototype	`sub procedure TP_TFT_Set_ADC_Threshold(dim threshold as word)`
Description	Set custom ADC threshold value, call this function after TP_TFT_Init.
Parameters	- `threshold`: custom ADC threshold value.
Returns	Nothing.
Requires	TP_TFT_Init has to be called before using this routine.
Example	`TP_TFT_Set_ADC_Threshold(900)     ' Set touch panel ADC threshold`
Notes	None.

TP_TFT_Press_Detect

Prototype	`sub function TP_TFT_Press_Detect() as byte`
Description	Detects if the touch panel has been pressed.
Parameters	None.
Returns	- `1` - if touch panel is pressed. - `0` - otherwise.
Requires	Global variables: - `DriveX_Left`: DriveX_Left pin. - `DriveX_Right`: DriveX_Right pin. - `DriveY_Up`: DriveY_Up pin. - `DriveY_Down`: DriveY_Down pin. - `DriveX_Left_Direction`: Direction of DriveX_Left pin. - `DriveX_Right_Direction`: Direction of DriveX_Right pin. - `DriveY_Up_Direction`: Direction of DriveY_Up pin. - `DriveY_Down_Direction`: Direction of DriveY_Down pin. must be defined before using this function.

Example

```
' Touch Panel module connections
dim DriveX_Left as sbit at LATB13_bit
    DriveX_Right as sbit at LATB11_bit
    DriveY_Up as sbit at LATB12_bit
    DriveY_Down as sbit at LATB10_bit
    DriveX_Left_Direction as sbit at TRISB13_bit
    DriveX_Right_Direction as sbit at TRISB11_bit
    DriveY_Up_Direction as sbit at TRISB12_bit
    DriveY_Down_Direction as sbit at TRISB10_bit
' End Touch Panel module connections

if (TP_TFT_Press_Detect() <> 0) then
  ...
end if
```

Notes	None.

TP_TFT_Get_Coordinates

Prototype	`sub function TP_TFT_Get_Coordinates(dim byref x_coordinate as word, dim byref y_coordinate as word) as byte`
Description	Get touch panel coordinates and store them in `x_coordinate` and `y_coordinate` parameters.
Parameters	- `x_coordinate`: x coordinate of the place of touch. - `y_coordinate`: y coordinate of the place of touch.
Returns	- `1` - if reading is within display dimension range. - `0` - if reading is out of display dimension range.
Requires	Nothing.

Example	`if (TP_TFT_Get_Coordinates(@x_coord, @y_coord) = 0) then`
	`...`
	`end if`

Notes	None.

TP_TFT_Calibrate_Min

Prototype	`sub procedure TP_TFT_Calibrate_Min()`
Description	Calibrate bottom left corner of the touch Panel.
Parameters	None.
Returns	Nothing.
Requires	Nothing.
Example	`TP_TFT_Calibrate_Min()    ' Calibration of bottom left corner`
Notes	None.

TP_TFT_Calibrate_Max

Prototype	`sub procedure TP_TFT_Calibrate_Max()`
Description	Calibrate upper right corner of the touch panel.
Parameters	None.
Returns	Nothing.
Requires	Nothing.
Example	`TP_TFT_Calibrate_Max()    ' Calibration of upper right corner`
Notes	None.

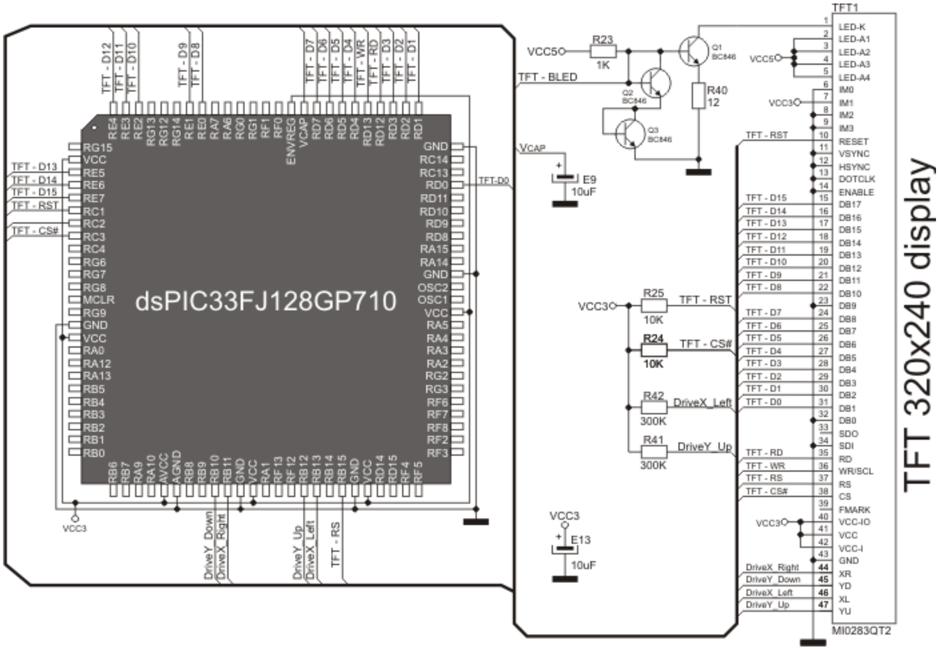
TP_TFT_Get_Calibration_Consts

Prototype	`sub procedure TP_TFT_Get_Calibration_Consts(dim byref x_min as word, dim byref x_max as word, dim byref y_min as word, dim byref y_max as word)`
Description	Gets calibration constants after calibration is done and stores them in `x_min`, `x_max`, `y_min` and `y_max` parameters.
Parameters	- `x_min`: x coordinate of the bottom left corner of the working area. - `x_max`: x coordinate of the upper right corner of the working area. - `y_min`: y coordinate of the bottom left corner of the working area. - `y_max`: y coordinate of the upper right corner of the working area.
Returns	Nothing.
Requires	Nothing.
Example	`TP_TFT_Get_Calibration_Consts(@x_min, @y_min, @x_max, @y_max)       ' Get calibration constants`
Notes	None.

TP_TFT_Set_Calibration_Consts

Prototype	`sub procedure TP_TFT_Set_Calibration_Consts(dim x_min as word, dim x_max as word, dim y_min as word, dim y_max as word)`
Description	Sets calibration constants.
Parameters	- `x_min`: x coordinate of the bottom left corner of the working area. - `x_max`: x coordinate of the upper right corner of the working area. - `y_min`: y coordinate of the bottom left corner of the working area. - `y_max`: y coordinate of the upper right corner of the working area.
Returns	Nothing.
Requires	Nothing.
Example	`TP_TFT_Set_Calibration_Consts(148, 3590, 519, 3370)     ' Set calibration constants`
Notes	None.

HW Connection



Touch Panel TFT HW connection

UART Library

The UART hardware module is available with a number of dsPIC30/33 and PIC24 MCUs. The mikroBasic PRO for dsPIC30/33 and PIC24 UART Library provides comfortable work with the Asynchronous (full duplex) mode.

Important:

- UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**.
- Switching between the UART modules in the UART library is done by the UART_Set_Active function (UART modules have to be previously initialized).
- Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

Library Routines

- UARTx_Init
- UARTx_Init_Advanced
- UARTx_Data_Ready
- UARTx_Tx_Idle
- UARTx_Read
- UARTx_Read_Text
- UARTx_Write
- UARTx_Write_Text
- UART_Set_Active

UARTx_Init

Prototype	`sub procedure UARTx_Init(dim baud_rate as longint)`
Description	Configures and initializes the UART module. The internal UART module module is set to: - continue operation in IDLE mode - default Tx and Rx pins - loopback mode disabled - 8-bit data, no parity - 1 STOP bit - transmitter enabled - generate interrupt on transmission end - interrupt on reception enabled - Address Detect mode disabled
Parameters	- `baud_rate`: requested baud rate
Returns	Nothing.
Requires	Routine requires the UART module.
Example	`' Initialize hardware UART1 module and establish communication at 2400 bps` `UART1_Init(2400)`
Notes	Refer to the device data sheet for baud rates allowed for specific Fosc. For the dsPIC33 and PIC24 MCUs, the compiler will choose for which speed the calculation is to be performed (high or low). This does not mean that it is the best choice for desired baud rate. If the baud rate error generated in this way is too big then UARTx_Init_Advanced routine, which allows speed select be used. UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**. Switching between the UART modules in the UART library is done by the UART_Set_Active function (UART modules have to be previously initialized). Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

UARTx_Init_Advanced

Prototype

```
' dsPIC30 prototype
sub procedure UARTx_Init_Advanced(dim baud_rate as longint, dim parity,
stop_bits as word)

' dsPIC33 and PIC24 prototype
sub procedure UARTx_Init_Advanced(dim baud_rate as longint, dim parity,
stop_bits as word, dim high_low_speed as word)
```

Description

Configures and initializes the UART module with user defined settings.

Parameters

- `baud_rate`: requested baud rate
- `parity`: parity and data selection parameter.

Valid values:

Data/Parity Mode	
Description	**Predefined library const**
`8-bit data, no parity`	`_UART_8BIT_NOPARITY`
`8-bit data, even parity`	`_UART_8BIT_EVENPARITY`
`8-bit data, odd parity`	`_UART_8BIT_ODDPARITY`
`9-bit data, no parity`	`_UART_9BIT_NOPARITY`

- `stop_bits`: stop bit selection parameter.

Valid values:

Stop bits	
Description	**Predefined library const**
`One stop bit`	`_UART_ONE_STOPBIT`
`Two stop bit`	`_UART_TWO_STOPBITS`

- `high_low_speed`: high/low speed selection parameter. Available only for dsPIC33 and PIC24 MCUs.

Valid values:

High/Low Speed	
Description	**Predefined library const**
`Low Speed UART`	`_UART_LOW_SPEED`
`Hi Speed UART`	`_UART_HI_SPEED`

Returns	Nothing.
Requires	Routine requires the UART module.

Example

```
' dsPIC30 family example
' Initialize hardware UART1 module and establish communication at 2400 bps,
8-bit data, even parity and 2 STOP bits
UART1_Init_Advanced(2400, 2, 1)

' dsPIC33 and PIC24 family example
' Initialize hardware UART2 module and establish communication at 2400 bps,
8-bit data, even parity, 2 STOP bits and high speed baud rate calculations
UART2_Init_Advanced(2400, 2, 1, 1)
```

Notes	Refer to the device data sheet for baud rates allowed for specific Fosc. UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**. Switching between the UART modules in the UART library is done by the UART_Set_Active function (UART modules have to be previously initialized). Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

UARTx_Data_Ready

Prototype	`sub function UARTx_Data_Ready() as word`
Description	The function tests if data in receive buffer is ready for reading.
Parameters	None.
Returns	- `1` if data is ready for reading - `0` if there is no data in the receive register
Requires	Routine requires at least one UART module. Used UART module must be initialized before using this routine. See UARTx_Init and UARTx_Init_Advanced routines.

Example

```
dim receive as word
...
' read data if ready
if (UART1_Data_Ready() = 1) then
  receive = UART1_Read()
end if
```

Notes	UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**. Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

UARTx_Tx_Idle

Prototype	`sub function UARTx_Tx_Idle() as word`
Description	Use the function to test if the transmit shift register is empty or not.
Parameters	None.
Returns	- `1` if the data has been transmitted - `0` otherwise
Requires	Routine requires at least one UART module. Used UART module must be initialized before using this routine. See UARTx_Init and UARTx_Init_Advanced routines.

Example

```
' If the previous data has been shifted out, send next data:
if (UART1_Tx_Idle() = 1) then
  UART1_Write(_data)
end if
```

Notes	UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**. Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

UARTx_Read

Prototype	`sub function UARTx_Read() as word`
Description	The function receives a byte via UART. Use the UARTx_Data_Ready function to test if data is ready first.
Parameters	None.
Returns	Received byte.
Requires	Routine requires at least one UART module. Used UART module must be initialized before using this routine. See UARTx_Init and UARTx_Init_Advanced routines.

Example

```
dim receive as word
...
' read data if ready
if (UART1_Data_Ready() = 1) then
  receive = UART1_Read()
end if
```

Notes	UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**. Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

UARTx_Read_Text

Prototype	`sub procedure UARTx_Read_Text(dim byref output, delimiter as string; dim Attempts as byte)`
Description	Reads characters received via UART until the delimiter sequence is detected. The read sequence is stored in the parameter `output`; delimiter sequence is stored in the parameter `delimiter`. This is a blocking call: the delimiter sequence is expected, otherwise the procedure exits (if the delimiter is not found).
Parameters	- `Output`: received text - `Delimiter`: sequence of characters that identifies the end of a received string - `Attempts`: defines number of received characters in which `Delimiter` sequence is expected. If `Attempts` is set to 255, this routine will continuously try to detect the `Delimiter` sequence.
Returns	Nothing.
Requires	Routine requires at least one UART module. Used UART module must be initialized before using this routine. See UARTx_Init and UARTx_Init_Advanced routines.

Example

Read text until the sequence “OK” is received, and send back what’s been received:

```
UART1_Init(4800)                        ' initialize UART module
Delay_ms(100)

 while TRUE
   if (UART1_Data_Ready() = 1)          ' if data is received
     UART1_Read_Text(output, “OK”, 10)   ' reads text until ‘OK’ is found
     UART1_Write_Text(output)            ' sends back text
   end if
 wend.
```

Notes	UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**. Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

UARTx_Write

Prototype	`sub procedure UARTx_Write(dim data_ as word)`
Description	The function transmits a byte via the UART module.
Parameters	- `data_`: data to be sent
Returns	Nothing.
Requires	Routine requires at least one UART module. Used UART module must be initialized before using this routine. See UARTx_Init and UARTx_Init_Advanced routines.

Example

```
dim data_ as byte
...
data_ = 0x1E
UART1_Write(data_)
```

Notes	UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**. Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

UARTx_Write_Text

Prototype	`sub procedure UARTx_Write_Text(dim byref uart_text as string[20])`
Description	Sends text via UART. Text should be zero terminated.
Parameters	- `UART_text`: text to be sent
Returns	Nothing.
Requires	Routine requires at least one UART module. Used UART module must be initialized before using this routine. See UARTx_Init and UARTx_Init_Advanced routines.

Example

Read text until the sequence “OK” is received, and send back what’s been received:

```
UART1_Init(4800)                            ' initialize UART module
Delay_ms(100)

 while TRUE
   if (UART1_Data_Ready() = 1)              ' if data is received
     UART1_Read_Text(output, 'OK', 10)   ' reads text until 'OK' is found
     UART1_Write_Text(output)               ' sends back text
   end if
 wend.
```

Notes	UART library routines require you to specify the module you want to use. To select the desired UART module, simply change the letter **x** in the routine prototype for a number from **1** to **4**. Number of UART modules per MCU differs from chip to chip. Please, read the appropriate datasheet before utilizing this library.

UART_Set_Active

Prototype	`sub procedure UART_Set_Active (dim read_ptr as ^TUART_Rd_Ptr, dim write_ptr as ^TUART_Wr_Ptr, dim ready_ptr as ^TUART_Rdy_Ptr, dim tx_idle_ptr as ^TUART_TX_Idle_Ptr)`
Description	Sets active UART module which will be used by UARTx_Data_Ready, UARTx_Read and UARTx_Write routines.
Parameters	Parameters: - `read_ptr`: UARTx_Read handler - `write_ptr`: UARTx_Write handler - `ready_ptr`: UARTx_Data_Ready handler - `tx_idle_ptr`: UARTx_Tx_Idle handler
Returns	Nothing.
Requires	Routine is available only for MCUs with multiple UART modules. Used UART module must be initialized before using this routine. See UARTx_Init and UARTx_Init_Advanced routines.

Example

```
UART1_Init(9600);                      ' initialize UART1 module
UART2_Init(9600);                      ' initialize UART2 module

RS485Master_Init();                    ' initialize MCU as Master

UART_Set_Active(@UART1_Read,  @UART1_Write,  @UART1_Data_Ready,  @UART1_Tx_
Idle);  ' set UART1 active
RS485Master_Send(dat,1,160);          ' send message through UART1

UART_Set_Active(@UART2_Read,  @UART2_Write,  @UART2_Data_Ready,  @UART2_Tx_
Idle);  ' set UART2 active
RS485Master_Send(dat,1,160);          ' send through UART2
```

Notes	None.

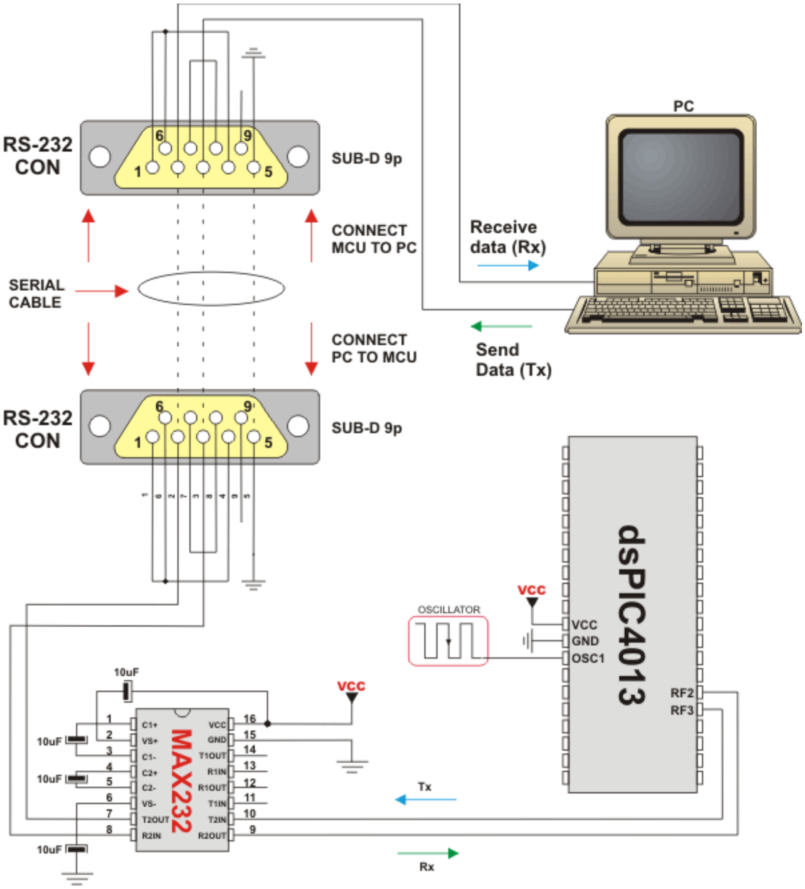
Library Example

This example demonstrates simple data exchange via UART. If MCU is connected to the PC, you can test the example from the mikroBasic PRO for dsPIC30/33 and PIC24 USART communication terminal, launch it from the drop-down menu **Tools › USART Terminal** or simply click the USART Terminal Icon .

Copy Code To Clipboard

```
program UART1
dim uart_rd as byte

main:
  ADPCFG = 0xFFFF                             ' Configure AN pins as digital

  UART1_Init(9600)                            ' Initialize UART module at 9600 bps
  Delay_ms(100)                               ' Wait for UART module to stabilize

'   U1MODE.ALTIO = 1       ' un-comment this line to have Rx and Tx pins on their
alternate
      ' locations. This is used to free the pins for other module, namely the SPI.

  UART1_Write_Text(“Start”)
  UART1_Write(10)
  UART1_Write(13)

  while TRUE                                  ' Endless loop
    if (UART1_Data_Ready() <> 0) then         ' If data is received,
      uart_rd = UART1_Read()                  '   read the received data,
      UART1_Write(uart_rd)                    '   and send data via UART
    end if
  wend
end.
```

HW Connection



RS232 HW connection

USB Library

Universal Serial Bus (USB) provides a serial bus standard for connecting a wide variety of devices, including computers, cell phones, game consoles, PDA's, etc.

USB Library contains HID routines that support HID class devices, and also the generic routines that can be used with vendor specified drivers.

USB HID Class

The HID class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of HID class devices include :

- Keyboards and pointing devices, for example: standard mouse devices, trackballs, and joysticks.
- Front-panel controls, for example: knobs, switches, buttons, and sliders.
- Controls that might be found on devices such as telephones, VCR remote controls, games or simulation devices, for example: data gloves, throttles, steering wheels, and rudder pedals.
- Devices that may not require human interaction but provide data in a similar format to HID class devices, for example, bar-code readers, thermometers, or voltmeters.

Many typical HID class devices include indicators, specialized displays, audio feedback, and force or tactile feedback. Therefore, the HID class definition includes support for various types of output directed to the end user.

Descriptor File

Each project based on the USB library should include a descriptor source file which contains vendor id and name, product id and name, report length, and other relevant information. To create a descriptor file, use the integrated USB HID terminal of mikroBasic PRO for dsPIC30/33 and PIC24 (**Tools › USB HID Terminal**). The default name for descriptor file is `USBdsc.mbas`, but you may rename it.

Library Routines

- HID_Enable
- HID_Read
- HID_Write
- HID_Disable
- USB_Interrupt_Proc
- USB_Polling_Proc
- Gen_Enable
- Gen_Read
- Gen_Write

HID_Enable

Prototype	`sub procedure HID_Enable(dim readbuff as ^byte, dim writebuff as ^byte)`
Description	Enables USB HID communication.
Parameters	- `readbuff`: Read Buffer. - `writebuff`: Write Buffer. These parameters are used for HID communication.
Returns	Nothing.
Requires	Nothing.
Example	`HID_Enable(@readbuff,@writebuff)`
Notes	This function needs to be called before using other routines of USB HID Library.

HID_Read

Prototype	`sub function HID_Read() as byte`
Description	Receives message from host and stores it in the Read Buffer.
Parameters	None.
Returns	If the data reading has failed, the function returns 0. Otherwise, it returns number of characters received from the host.
Requires	USB HID needs to be enabled before using this function. See HID_Enable.
Example	`while(HID_Read() = 0)` `wend`
Notes	None.

HID_Write

Prototype	`sub function HID_Write(dim writebuff as ^byte, dim len as byte) as byte`
Description	Function sends data from Write Buffer `writebuff` to host.
Parameters	- `writebuff`: Write Buffer, same parameter as used in initialization; see HID_Enable. - `len`: specifies a length of the data to be transmitted.
Returns	If the data transmitting has failed, the function returns 0. Otherwise, it returns number of transmitted bytes.
Requires	USB HID needs to be enabled before using this function. See HID_Enable.
Example	`while(HID_Write(@writebuff,64) = 0)` `wend`
Notes	Function call needs to be repeated as long as data is not successfuly sent.

HID_Disable

Prototype	`sub procedure HID_Disable()`
Description	Disables USB HID communication.
Parameters	None.
Returns	Nothing.
Requires	USB HID needs to be enabled before using this function. See HID_Enable.
Example	`HID_Disable()`
Notes	None.

USB_Interrupt_Proc

Prototype	`sub procedure USB_Interrupt_Proc()`
Description	This routine is used for servicing various USB bus events. Should be called inside USB interrupt routine.
Parameters	None.
Returns	Nothing.
Requires	Nothing.

Example

```
sub procedure USB1Interrupt() iv IVT_ADDR_USB1INTERRUPT
  USB_Interrupt_Proc()
end sub
```

Notes	Do not use this function with USB_Polling_Proc, only one should be used. To enable servicing through interrupt, `USB_INTERRUPT` constant should be set (it is set by default in descriptor file).

USB_Polling_Proc

Prototype	`sub procedure USB_Polling_Proc()`
Description	This routine is used for servicing various USB bus events. It should be periodically, preferably every 100 microseconds.
Parameters	None.
Returns	Nothing.
Requires	Nothing.

Example

```
while TRUE
  USB_Polling_Proc()
  kk = HID_Read()
  if (kk <> 0) then
    for cnt = 0 to 63
      writebuff[cnt] = readbuff[cnt]
    next cnt
    HID_Write(@writebuff,64)
  end if
wend
```

Notes	Do not use this functions with USB_Interrupt_Proc. To enable servicing by polling, `USB_INTERRUPT` constant should be set to 0 (it is located in descriptor file).

Gen_Enable

Prototype	`sub procedure Gen_Enable(dim readbuff as ^byte, dim writebuff as ^byte)`
Description	Initialize the USB module of the MCU.
Parameters	- `readbuff`: Read Buffer. - `writebuff`: Write Buffer.
Returns	Nothing.
Requires	USB needs to be enabled before using this function. See HID_Enable.
Example	`Gen_Enable(@readbuff,@writebuff)`
Notes	None.

Gen_Read

Prototype	`sub function Gen_Read(dim readbuff as ^byte, dim length as byte, dim ep as byte) as byte`
Description	Generic routine that receives the specified data from the specified endpoint.
Parameters	- `readbuff`: Received data. - `length`: The length of the data that you wish to receive. - `ep`: Endpoint number you want to receive the data into.
Returns	Returns the number of received bytes, otherwise 0.
Requires	USB needs to be enabled before using this function. See HID_Enable.
Example	`while(Gen_Read(@readbuff,64,1) = 0)` `wend`
Notes	None.

Gen_Write

Prototype	`sub function Gen_Write(dim writebuff as ^byte, dim as length as byte, dim ep as byte) as byte`
Description	Sends the specified data to the specified endpoint.
Parameters	- `writebuff`: The data that you want to send. - `length`: the length of the data that you wish to send. - `ep`: Endpoint number you want to send the data into.
Returns	Returns the number of transmitted bytes, otherwise 0.
Requires	USB needs to be enabled before using this function. See HID_Enable.
Example	`while(Gen_Write(@writebuff,64,1) = 0)` `wend`
Notes	None.

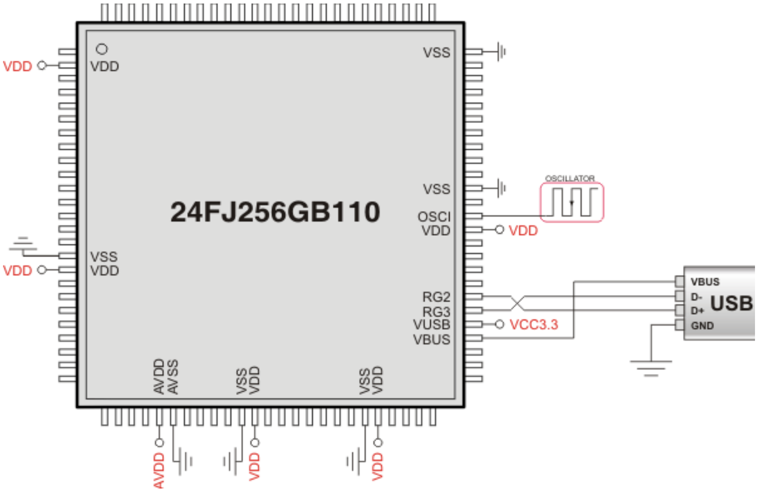
Library Example

This example establishes connection with the HID terminal that is active on the PC. Upon connection establishment, the HID Device Name will appear in the respective window. After that software will wait for data and it will return received data back. Examples uses `USBdsc.mbas` descriptor file, which is in the same folder, and can be created by the HID Terminal.

Copy Code To Clipboard

```
program HID_Write

dim cnt as char
dim readbuff as char[64]
dim writebuff as char[64]

sub procedure USB1Interrupt() iv IVT_ADDR_USB1INTERRUPT
   USB_Interrupt_Proc()
end sub

main:
  AD1PCFGL = 0xFFFF
  USB_Init_desc()
  HID_Enable(@readbuff,@writebuff)
  while TRUE
    while(HID_Read() = 0)
    wend

    for cnt=0 to 63
      writebuff[cnt] = readbuff[cnt]
    next cnt

    while(HID_Write(@writebuff,64) = 0)
    wend
  wend
end.
```

HW Connection



USB connection scheme

Digital Signal Processing Libraries

mikroBasic PRO for dsPIC30/33 and PIC24 includes various libraries for DSP engine. All DSP routines work with fractional Q15 format.

Digital Signal Processing Libraries

- FIR Filter Library
- IIR Filter Library
- FFT Library
- Bit Reverse Complex Library
- Vectors Library
- Matrices Library

FIR Filter Library

mikroBasic PRO for dsPIC30/33 and PIC24 includes a library for finite impulse response (FIR) filter. All routines work with fractional Q15 format.

A finite impulse response (FIR) filter is a type of a digital filter, whose impulse response (the filter's response to a delta function) is finite because it settles to zero in a finite number of sample intervals.

Library Routines

- FIR_Radix

FIR_Radix

Prototype	`sub function FIR_Radix(dim FilterOrder as word, dim ptrCoeffs as longint,dim BuffLength as word, dim ptrInput as word, dim Index as word) as word`
Description	This function applies FIR filter to `ptrInput`.
Parameters	- `FilterOrder:` order of the filter + 1 - `ptrCoeffs:` pointer to filter coefficients in program memory - `BuffLength` number of input samples - `ptrInput:` pointer to input samples - `Index:` index of current sample
Returns	$\sum_{k=0}^{N-1} coef[k]*input[N-k]$ with: `N` - buffer length `k` - current index
Requires	Nothing.
Example	

```
const BUFFFER_SIZE = 32
const FILTER_ORDER = 20
const COEFF_B as integer[FILTER_ORDER+1]  = (
      0x0000, 0x0048, 0x0133, 0x02D3, 0x052B, 0x0826,
      0x0BA0, 0x0F62, 0x1329, 0x16AA, 0x199A, 0x16AA,
      0x1329, 0x0F62, 0x0BA0, 0x0826, 0x052B, 0x02D3,
      0x0133, 0x0048, 0x0000
      )

dim input as word[BUFFFER_SIZE] ydata       ' Input buffer
    inext as word                           ' Input buffer index
...
dim CurrentValue as word
CurrentValue = FIR_Radix(FILTER_ORDER+1,    ' Filter order
                          word(@COEFF_B),   ' B coefficients of the filter
                          BUFFER_SIZE,      ' Input buffer length
                          word(@input),     ' Input buffer
                          inext)            ' Current sample
```

Notes	Input samples must be in Y data space.

IIR Filter Library

mikroBasic PRO for dsPIC30/33 and PIC24 includes a library for Infinite Impulse Response (IIR) filter. All routines work with fractional Q15 format.

A infinite impulse response (IIR) filter is a type of a digital filter, whose impulse response (the filter's response to a delta function) is non-zero over an infinite length of time.

Library Routines

IIR_Radix

IIR_Radix

Prototype	`sub function IIR_Radix(dim BScale, AScale as integer, dim ptrB, ptrA, FilterOrder, ptrInput, InputLen, ptrOutput, Index as word) as word`
Description	This function applies IIR filter to `ptrInput`.
Parameters	- `BScale:` B scale factor. - `AScale:` A scale factor. - `ptrB:` pointer to B coefficients (in program memory). - `ptrA:` pointer to A coefficients (in program memory). - `FilterOrder:` order of the filter + 1. - `ptrInput:` address of input samples. - `InputLen:` number of samples. - `ptrOutput:` pointer to output samples. Output length is equal to Input length. - `Index:` index of current sample.
Returns	$y[n]=\sum_{k=0}^{N}(Acoeff[n]*x[n-k])-\sum_{k=1}^{M}(Bcoef[k]*y[n-k])$
Requires	Nothing.

Example

```
const BUFFER_SIZE = 8
const FILTER_ORDER = 6
const COEFF_B as word[FILTER_ORDER+1] = (0x0548, 0x1FAE, 0x4F34, 0x699B,
0x4F34, 0x1FAE, 0x0548)
const COEFF_A as word[FILTER_ORDER+1] = (0x4000, 0xB3FE, 0x5389, 0xD4D8,
0x10DD, 0xFCB0, 0x0052)
const SCALE_B = 2
const SCALE_A = -1
dim inext as word                       ' Input buffer index
    input as word[BUFFER_SIZE] ydata    ' Input buffer
    output as word[BUFFER_SIZE] ydata   ' Output buffer
...
dim CurrentValue as word
CurrentValue = IIR_Radix(SCALE_B,
                         SCALE_A,
                         word(@COEFF_B),   ' b coefficients of the filter
                         word(@COEFF_A),   ' a coefficients of the filter
                         FILTER_ORDER+1,   ' Filter order + 1
                         word(@input),     ' Input buffer
                         BUFFER_SIZE,      ' Input buffer length
                         word(@output),    ' Input buffer
                         inext)            ' Current sample
```

Notes	Input and output samples must be in Y data space.

FFT Library

mikroBasic PRO for dsPIC30/33 and PIC24 includes a library for FFT calculation. All routines work with fractional Q15 format.

Library Dependency Tree

Library Routines

- FFT

FFT

Prototype	`sub procedure FFT(dim log2N as word, dim TwiddleFactorsAddress as longint, dim byref Samples as word[1024])`
Description	Function applies FFT transformation to input samples, input samples must be in Y data space. $F(k)=\frac{1}{N}*\sum_{(n,k)=0}^{N-1}(f(n)*WN(kn)),\ WN(kn)=e^{\frac{-j*2*\pi*k*n}{N}}$ - `f(n):` array of complex input samples - `WN:` TwiddleFactors - `N =` 2^{m}, m□Z The amplitude of current FFT sample is calculated as: $F[k]=\sqrt{(Re^{2}[k]+Im^{2}[k])}$
Parameters	- `log2N:` buffer length (must be the power of 2). - T`widdleFactorsAddress:` address of constant array which contains complex twiddle factors. The array is expected to be in program memory. See Twiddle Factors for adequate array values. - `Samples:` array of input samples. Upon completion, complex array of FFT samples is placed in the `Samples:` parameter.
Returns	Nothing.
Requires	Nothing.

Example	`dim InputSamples as word[512] ydata`
	`...`
	`' Perform FFT (DFT), 7 stages, 128 samples of complex pairs`
	`FFT(8, TwiddleCoeff_256, InputSamples)`

Notes	Complex array of FFT samples is placed in Samples parameter. Input Samples are arranged in manner Re,Im,Re,Im... (where Im is always zero). Output samples are arranged in the same manner but Im parts are different from zero. Output samples are symmetrical (First half of output samples (index from 0 to N/2) is identical as second half of output samples(index from N/2 to N). Input data is a complex vector such that the magnitude of the real and imaginary parts of each of its elements is less than 0.5. If greater or equal to this value the results could produce saturation. Note that the output values are scaled by a factor of 1/N, with N the length of the FFT. input is expected in natural ordering, while output is produced in bit reverse ordering.

Twiddle Factors:

TwiddleCoeff_64

```
const TwiddleCoeff_64 as word[64] = (
   0x7FFF, 0x0000, 0x7F62, 0xF374, 0x7D8A, 0xE707, 0x7A7D, 0xDAD8,
   0x7642, 0xCF04, 0x70E3, 0xC3A9, 0x6A6E, 0xB8E3, 0x62F2, 0xAECC,
   0x5A82, 0xA57E, 0x5134, 0x9D0E, 0x471D, 0x9592, 0x3C57, 0x8F1D,
   0x30FC, 0x89BE, 0x2528, 0x8583, 0x18F9, 0x8276, 0x0C8C, 0x809E,
   0x0000, 0x8000, 0xF374, 0x809E, 0xE707, 0x8276, 0xDAD8, 0x8583,
   0xCF04, 0x89BE, 0xC3A9, 0x8F1D, 0xB8E3, 0x9592, 0xAECC, 0x9D0E,
   0xA57E, 0xA57E, 0x9D0E, 0xAECC, 0x9592, 0xB8E3, 0x8F1D, 0xC3A9,
   0x89BE, 0xCF04, 0x8583, 0xDAD8, 0x8276, 0xE707, 0x809E, 0xF374)
```

TwiddleCoeff_128

```
const TwiddleCoeff_128 as word[128] = (
   0x7FFF, 0x0000, 0x7FD9, 0xF9B8, 0x7F62, 0xF374, 0x7E9D, 0xED38,
   0x7D8A, 0xE707, 0x7C2A, 0xE0E6, 0x7A7D, 0xDAD8, 0x7885, 0xD4E1,
   0x7642, 0xCF04, 0x73B6, 0xC946, 0x70E3, 0xC3A9, 0x6DCA, 0xBE32,
   0x6A6E, 0xB8E3, 0x66D0, 0xB3C0, 0x62F2, 0xAECC, 0x5ED7, 0xAA0A,
   0x5A82, 0xA57E, 0x55F6, 0xA129, 0x5134, 0x9D0E, 0x4C40, 0x9930,
   0x471D, 0x9592, 0x41CE, 0x9236, 0x3C57, 0x8F1D, 0x36BA, 0x8C4A,
   0x30FC, 0x89BE, 0x2B1F, 0x877B, 0x2528, 0x8583, 0x1F1A, 0x83D6,
   0x18F9, 0x8276, 0x12C8, 0x8163, 0x0C8C, 0x809E, 0x0648, 0x8027,
   0x0000, 0x8000, 0xF9B8, 0x8027, 0xF374, 0x809E, 0xED38, 0x8163,
   0xE707, 0x8276, 0xE0E6, 0x83D6, 0xDAD8, 0x8583, 0xD4E1, 0x877B,
   0xCF04, 0x89BE, 0xC946, 0x8C4A, 0xC3A9, 0x8F1D, 0xBE32, 0x9236,
   0xB8E3, 0x9592, 0xB3C0, 0x9930, 0xAECC, 0x9D0E, 0xAA0A, 0xA129,
   0xA57E, 0xA57E, 0xA129, 0xAA0A, 0x9D0E, 0xAECC, 0x9930, 0xB3C0,
   0x9592, 0xB8E3, 0x9236, 0xBE32, 0x8F1D, 0xC3A9, 0x8C4A, 0xC946,
   0x89BE, 0xCF04, 0x877B, 0xD4E1, 0x8583, 0xDAD8, 0x83D6, 0xE0E6,
   0x8276, 0xE707, 0x8163, 0xED38, 0x809E, 0xF374, 0x8027, 0xF9B8)
```

TwiddleCoeff_256

```
const TwiddleCoeff_256 as word[256] = (
   0x7FFF, 0x0000, 0x7FF6, 0xFCDC, 0x7FD9, 0xF9B8, 0x7FA7, 0xF695,
   0x7F62, 0xF374, 0x7F0A, 0xF055, 0x7E9D, 0xED38, 0x7E1E, 0xEA1E,
   0x7D8A, 0xE707, 0x7CE4, 0xE3F4, 0x7C2A, 0xE0E6, 0x7B5D, 0xDDDC,
   0x7A7D, 0xDAD8, 0x798A, 0xD7D9, 0x7885, 0xD4E1, 0x776C, 0xD1EF,
   0x7642, 0xCF04, 0x7505, 0xCC21, 0x73B6, 0xC946, 0x7255, 0xC673,
   0x70E3, 0xC3A9, 0x6F5F, 0xC0E9, 0x6DCA, 0xBE32, 0x6C24, 0xBB85,
   0x6A6E, 0xB8E3, 0x68A7, 0xB64C, 0x66D0, 0xB3C0, 0x64E9, 0xB140,
   0x62F2, 0xAECC, 0x60EC, 0xAC65, 0x5ED7, 0xAA0A, 0x5CB4, 0xA7BD,
   0x5A82, 0xA57E, 0x5843, 0xA34C, 0x55F6, 0xA129, 0x539B, 0x9F14,
   0x5134, 0x9D0E, 0x4EC0, 0x9B17, 0x4C40, 0x9930, 0x49B4, 0x9759,
   0x471D, 0x9592, 0x447B, 0x93DC, 0x41CE, 0x9236, 0x3F17, 0x90A1,
   0x3C57, 0x8F1D, 0x398D, 0x8DAB, 0x36BA, 0x8C4A, 0x33DF, 0x8AFB,
   0x30FC, 0x89BE, 0x2E11, 0x8894, 0x2B1F, 0x877B, 0x2827, 0x8676,
   0x2528, 0x8583, 0x2224, 0x84A3, 0x1F1A, 0x83D6, 0x1C0C, 0x831C,
```

```
  0x18F9, 0x8276, 0x15E2, 0x81E2, 0x12C8, 0x8163, 0x0FAB, 0x80F6,
  0x0C8C, 0x809E, 0x096B, 0x8059, 0x0648, 0x8027, 0x0324, 0x800A,
  0x0000, 0x8000, 0xFCDC, 0x800A, 0xF9B8, 0x8027, 0xF695, 0x8059,
  0xF374, 0x809E, 0xF055, 0x80F6, 0xED38, 0x8163, 0xEA1E, 0x81E2,
  0xE707, 0x8276, 0xE3F4, 0x831C, 0xE0E6, 0x83D6, 0xDDDC, 0x84A3,
  0xDAD8, 0x8583, 0xD7D9, 0x8676, 0xD4E1, 0x877B, 0xD1EF, 0x8894,
  0xCF04, 0x89BE, 0xCC21, 0x8AFB, 0xC946, 0x8C4A, 0xC673, 0x8DAB,
  0xC3A9, 0x8F1D, 0xC0E9, 0x90A1, 0xBE32, 0x9236, 0xBB85, 0x93DC,
  0xB8E3, 0x9592, 0xB64C, 0x9759, 0xB3C0, 0x9930, 0xB140, 0x9B17,
  0xAECC, 0x9D0E, 0xAC65, 0x9F14, 0xAA0A, 0xA129, 0xA7BD, 0xA34C,
  0xA57E, 0xA57E, 0xA34C, 0xA7BD, 0xA129, 0xAA0A, 0x9F14, 0xAC65,
  0x9D0E, 0xAECC, 0x9B17, 0xB140, 0x9930, 0xB3C0, 0x9759, 0xB64C,
  0x9592, 0xB8E3, 0x93DC, 0xBB85, 0x9236, 0xBE32, 0x90A1, 0xC0E9,
  0x8F1D, 0xC3A9, 0x8DAB, 0xC673, 0x8C4A, 0xC946, 0x8AFB, 0xCC21,
  0x89BE, 0xCF04, 0x8894, 0xD1EF, 0x877B, 0xD4E1, 0x8676, 0xD7D9,
  0x8583, 0xDAD8, 0x84A3, 0xDDDC, 0x83D6, 0xE0E6, 0x831C, 0xE3F4,
  0x8276, 0xE707, 0x81E2, 0xEA1E, 0x8163, 0xED38, 0x80F6, 0xF055,
  0x809E, 0xF374, 0x8059, 0xF695, 0x8027, 0xF9B8, 0x800A, 0xFCDC)
```

TwiddleCoeff_512

```
const TwiddleCoeff_512 as word[512] = (
  0x7FFF, 0x0000, 0x7FFE, 0xFE6E, 0x7FF6, 0xFCDC, 0x7FEA, 0xFB4A,
  0x7FD9, 0xF9B8, 0x7FC2, 0xF827, 0x7FA7, 0xF695, 0x7F87, 0xF505,
  0x7F62, 0xF374, 0x7F38, 0xF1E4, 0x7F0A, 0xF055, 0x7ED6, 0xEEC6,
  0x7E9D, 0xED38, 0x7E60, 0xEBAB, 0x7E1E, 0xEA1E, 0x7DD6, 0xE892,
  0x7D8A, 0xE707, 0x7D3A, 0xE57D, 0x7CE4, 0xE3F4, 0x7C89, 0xE26D,
  0x7C2A, 0xE0E6, 0x7BC6, 0xDF61, 0x7B5D, 0xDDDC, 0x7AEF, 0xDC59,
  0x7A7D, 0xDAD8, 0x7A06, 0xD958, 0x798A, 0xD7D9, 0x790A, 0xD65C,
  0x7885, 0xD4E1, 0x77FB, 0xD367, 0x776C, 0xD1EF, 0x76D9, 0xD079,
  0x7642, 0xCF04, 0x75A6, 0xCD92, 0x7505, 0xCC21, 0x7460, 0xCAB2,
  0x73B6, 0xC946, 0x7308, 0xC7DB, 0x7255, 0xC673, 0x719E, 0xC50D,
  0x70E3, 0xC3A9, 0x7023, 0xC248, 0x6F5F, 0xC0E9, 0x6E97, 0xBF8C,
  0x6DCA, 0xBE32, 0x6CF9, 0xBCDA, 0x6C24, 0xBB85, 0x6B4B, 0xBA33,
  0x6A6E, 0xB8E3, 0x698C, 0xB796, 0x68A7, 0xB64C, 0x67BD, 0xB505,
  0x66D0, 0xB3C0, 0x65DE, 0xB27F, 0x64E9, 0xB140, 0x63EF, 0xB005,
  0x62F2, 0xAECC, 0x61F1, 0xAD97, 0x60EC, 0xAC65, 0x5FE4, 0xAB36,
  0x5ED7, 0xAA0A, 0x5DC8, 0xA8E2, 0x5CB4, 0xA7BD, 0x5B9D, 0xA69C,
  0x5A82, 0xA57E, 0x5964, 0xA463, 0x5843, 0xA34C, 0x571E, 0xA238,
  0x55F6, 0xA129, 0x54CA, 0xA01C, 0x539B, 0x9F14, 0x5269, 0x9E0F,
  0x5134, 0x9D0E, 0x4FFB, 0x9C11, 0x4EC0, 0x9B17, 0x4D81, 0x9A22,
  0x4C40, 0x9930, 0x4AFB, 0x9843, 0x49B4, 0x9759, 0x486A, 0x9674,
  0x471D, 0x9592, 0x45CD, 0x94B5, 0x447B, 0x93DC, 0x4326, 0x9307,
  0x41CE, 0x9236, 0x4074, 0x9169, 0x3F17, 0x90A1, 0x3DB8, 0x8FDD,
  0x3C57, 0x8F1D, 0x3AF3, 0x8E62, 0x398D, 0x8DAB, 0x3825, 0x8CF8,
  0x36BA, 0x8C4A, 0x354E, 0x8BA0, 0x33DF, 0x8AFB, 0x326E, 0x8A5A,
  0x30FC, 0x89BE, 0x2F87, 0x8927, 0x2E11, 0x8894, 0x2C99, 0x8805,
  0x2B1F, 0x877B, 0x29A4, 0x86F6, 0x2827, 0x8676, 0x26A8, 0x85FA,
  0x2528, 0x8583, 0x23A7, 0x8511, 0x2224, 0x84A3, 0x209F, 0x843A,
  0x1F1A, 0x83D6, 0x1D93, 0x8377, 0x1C0C, 0x831C, 0x1A83, 0x82C6,
  0x18F9, 0x8276, 0x176E, 0x822A, 0x15E2, 0x81E2, 0x1455, 0x81A0,
```

```
0x12C8, 0x8163, 0x113A, 0x812A, 0x0FAB, 0x80F6, 0x0E1C, 0x80C8,
0x0C8C, 0x809E, 0x0AFB, 0x8079, 0x096B, 0x8059, 0x07D9, 0x803E,
0x0648, 0x8027, 0x04B6, 0x8016, 0x0324, 0x800A, 0x0192, 0x8002,
0x0000, 0x8000, 0xFE6E, 0x8002, 0xFCDC, 0x800A, 0xFB4A, 0x8016,
0xF9B8, 0x8027, 0xF827, 0x803E, 0xF695, 0x8059, 0xF505, 0x8079,
0xF374, 0x809E, 0xF1E4, 0x80C8, 0xF055, 0x80F6, 0xEEC6, 0x812A,
0xED38, 0x8163, 0xEBAB, 0x81A0, 0xEA1E, 0x81E2, 0xE892, 0x822A,
0xE707, 0x8276, 0xE57D, 0x82C6, 0xE3F4, 0x831C, 0xE26D, 0x8377,
0xE0E6, 0x83D6, 0xDF61, 0x843A, 0xDDDC, 0x84A3, 0xDC59, 0x8511,
0xDAD8, 0x8583, 0xD958, 0x85FA, 0xD7D9, 0x8676, 0xD65C, 0x86F6,
0xD4E1, 0x877B, 0xD367, 0x8805, 0xD1EF, 0x8894, 0xD079, 0x8927,
0xCF04, 0x89BE, 0xCD92, 0x8A5A, 0xCC21, 0x8AFB, 0xCAB2, 0x8BA0,
0xC946, 0x8C4A, 0xC7DB, 0x8CF8, 0xC673, 0x8DAB, 0xC50D, 0x8E62,
0xC3A9, 0x8F1D, 0xC248, 0x8FDD, 0xC0E9, 0x90A1, 0xBF8C, 0x9169,
0xBE32, 0x9236, 0xBCDA, 0x9307, 0xBB85, 0x93DC, 0xBA33, 0x94B5,
0xB8E3, 0x9592, 0xB796, 0x9674, 0xB64C, 0x9759, 0xB505, 0x9843,
0xB3C0, 0x9930, 0xB27F, 0x9A22, 0xB140, 0x9B17, 0xB005, 0x9C11,
0xAECC, 0x9D0E, 0xAD97, 0x9E0F, 0xAC65, 0x9F14, 0xAB36, 0xA01C,
0xAA0A, 0xA129, 0xA8E2, 0xA238, 0xA7BD, 0xA34C, 0xA69C, 0xA463,
0xA57E, 0xA57E, 0xA463, 0xA69C, 0xA34C, 0xA7BD, 0xA238, 0xA8E2,
0xA129, 0xAA0A, 0xA01C, 0xAB36, 0x9F14, 0xAC65, 0x9E0F, 0xAD97,
0x9D0E, 0xAECC, 0x9C11, 0xB005, 0x9B17, 0xB140, 0x9A22, 0xB27F,
0x9930, 0xB3C0, 0x9843, 0xB505, 0x9759, 0xB64C, 0x9674, 0xB796,
0x9592, 0xB8E3, 0x94B5, 0xBA33, 0x93DC, 0xBB85, 0x9307, 0xBCDA,
0x9236, 0xBE32, 0x9169, 0xBF8C, 0x90A1, 0xC0E9, 0x8FDD, 0xC248,
0x8F1D, 0xC3A9, 0x8E62, 0xC50D, 0x8DAB, 0xC673, 0x8CF8, 0xC7DB,
0x8C4A, 0xC946, 0x8BA0, 0xCAB2, 0x8AFB, 0xCC21, 0x8A5A, 0xCD92,
0x89BE, 0xCF04, 0x8927, 0xD079, 0x8894, 0xD1EF, 0x8805, 0xD367,
0x877B, 0xD4E1, 0x86F6, 0xD65C, 0x8676, 0xD7D9, 0x85FA, 0xD958,
0x8583, 0xDAD8, 0x8511, 0xDC59, 0x84A3, 0xDDDC, 0x843A, 0xDF61,
0x83D6, 0xE0E6, 0x8377, 0xE26D, 0x831C, 0xE3F4, 0x82C6, 0xE57D,
0x8276, 0xE707, 0x822A, 0xE892, 0x81E2, 0xEA1E, 0x81A0, 0xEBAB,
0x8163, 0xED38, 0x812A, 0xEEC6, 0x80F6, 0xF055, 0x80C8, 0xF1E4,
0x809E, 0xF374, 0x8079, 0xF505, 0x8059, 0xF695, 0x803E, 0xF827,
0x8027, 0xF9B8, 0x8016, 0xFB4A, 0x800A, 0xFCDC, 0x8002, 0xFE6E)
```

Bit Reverse Complex Library

mikroBasic PRO for dsPIC30/33 and PIC24 includes a Bit Reverse Complex Library for DSP engine. All routines work with fractional Q15 format.

Library Routines

- BitReverseComplex

BitReverseComplex

Prototype	`sub procedure BitReverseComplex(dim log2N as word, dim byref ReIm as word[1024])`
Description	This function does Complex (in-place) Bit Reverse re-organization.
Parameters	`- N:` buffer length (must be the power of 2). `- ReIm:` output sample(from FFT).
Returns	Nothing.
Requires	Nothing.

Example

```
dim InputSamples as word[512] ydata     ' Y data is required by FFT routine
                ' See datasheet for your dsPIC to see Y data space limits.
...

' Perform FFT (DFT), 7 stages, 128 samples of complex pairs
' Twiddle factors are taken from help
FFT(8, word(@TwiddleCoeff_256), InputSamples)

' DFT butterfly algorythm bit-reverses output samples.
' We have to restore them in natural order.
BitReverseComplex(8, InputSamples)
```

Notes	Input samples must be in Y data space.

Vectors Library

mikroBasic PRO for dsPIC30/33 and PIC24 includes a library for working and using vectors. All routines work with fractional Q15 format.

Library Routines

- Vector_Set
- Vector_Power
- Vector_Subtract
- Vector_Scale
- Vector_Negate
- Vector_Multiply
- Vector_Min
- Vector_Max
- Vector_Dot
- Vector_Correlate
- Vector_Convolve
- Vector_Add

Vector_Set

Prototype	`sub procedure Vector_Set(dim byref input as word[1024], dim size, value as word)`
Description	Sets `size` elements of `input` to `value`, starting from the first element.
Parameters	- `input:` pointer to original vector - `size:` number of vector elements - `value:` value written to the elements
Returns	Nothing.
Requires	Nothing.

Example	`dim vec2 as word[3]`
	
	`Vector_Set(vec2, 3, 0x4000)`

Notes	- `size` must be > 0 - Length of `input` is limited by available RAM

Vector_Power

Prototype	`sub function Vector_Power(dim N as word, dim byref srcV as word[1024]) as word`
Description	Function returns result of power value (powVal) in radix point 1.15
Parameters	- `N:` number elements in vector(s) - `srcV:` pointer to source vector
Returns	$powVal = \sum_{n=0}^{numElems-1} (srcV[n]*srcV[n])$
Requires	Nothing.

Example

```
dim vec1 as word[3]

Vector_Power(3, vec1)
```

Notes	- [W0..W2] used, not restored - [W4] used, not restored - AccuA used, not restored - CORCON saved, used, restored

Vector_Subtract

Prototype	`sub procedure Vector_Subtract(dim byref dest, v1, v2 as word[1024], dim numElems as word)`
Description	This function does subtraction of two vectors. **dstV[n] = v1[n] - v2[n]**, n є [0, numElems-1]
Parameters	- `numElems:` must be less or equal to minimum size of two vectors. - `v1:` first vector - `v2:` second vector - `dest:` result vector
Returns	Nothing.
Requires	Nothing.

Example

```
dim vec1    as word[3]
    vec2    as word[3]
    vecDest as word[3]

Vector_Subtract(vecDest, vec1, vec2, 3)
```

Notes	- AccuA used, not restored. - CORCON saved, used, restored.

Vector_Scale

Prototype	`sub procedure Vector_Scale(dim N as word, dim ScaleValue as integer, dim byref SrcVector, DestVector as word[1024])`
Description	This function does vector scaling with scale value. **dstV[n] = sclVal * srcV[n]**, n є [0, numElems-1]
Parameters	- `N:` buffer length - `SrcVector:` original vector - `DestVector:` scaled vector - `ScaleValue:` scale value
Returns	Nothing.
Requires	Nothing.

Example

```
dim vec1    as word[3]
    vecDest as word[3]

Vector_Scale(3, 2, vec1, vecDest)
```

Notes	- [W0..W5] used, not restored - AccuA used, not restored - CORCON saved, used, restored

Vector_Negate

Prototype	`sub procedure Vector_Negate(dim byref srcVector, DestVector as word[1024], dim numElems as word)`
Description	This function does negation of vector. **dstV[n] = (-1)*srcV1[n] + 0**, nє [0, numElems]
Parameters	- `srcVector:` original vector - `destVector:` result vector - `numElems:` number of elements in vector(s)
Returns	Nothing.
Requires	Nothing.

Example

```
dim vec1    as word[3]
    vecDest as word[3]

Vector_Negate(vec1, vecDest, 3)
```

Notes	- Negate of 0x8000 is 0x7FFF - [W0]..[W5] used, not restored - AccuA used, not restored - CORCON saved, used, restored

Vector_Multiply

Prototype	`sub procedure Vector_Multiply(dim byref v1, v2, dest as word[1024], dim numElems as word)`
Description	This function does multiplication of two vectors. **dstV[n] = srcV1[n] * srcV2[n]**, n є [0, numElems-1]
Parameters	- `numElems:` number elements in vector(s) (must be less or equal to minimum size of two vectors) - `v1:` first vector - `v2:` second vector - `dest:` result vector
Returns	Nothing.
Requires	Nothing.

Example

```
dim vec1     as word[3]
    vec2     as word[3]
    vConDest as word[10]

Vector_Multiply(vec1, vConDest, vec2, 3)
```

Notes	- [W0..W5] used, not restored - AccuA used, not restored - CORCON saved, used, restored

Vector_Min

Prototype	`sub function Vector_Min(dim byref Vector as word[1024], dim numElems as word, dim byref MinIndex as word) as word`
Description	This function finds minimal value in vector. **minVal = min (srcV[n])**, n є [0, numElems-1] If `srcV[i] = srcV[j] = minVal`, and `i < j`, then `MinIndex = j`.
Parameters	- `Vector:` original vector - `numElems:` number of elements in vector - `MinIndex:` index of minimum value
Returns	Minimum value (`minVal`).
Requires	Nothing.

Example

```
dim vec1 as word[3]
    index, rslt as word

rslt = Vector_Min(vec1, 3, index)
```

Notes	- [W0..W5] used, not restored

Vector_Max

Prototype	`sub function Vector_Max(dim byref Vector as word[1024], dim numElems as word, dim byref MaxIndex as word) as word`
Description	This function find maximal value in vector. **maxVal = max (srcV[n])**, n є [0, numElems-1] If `srcV[i] = srcV[j] = maxVal,` and `i < j`, then `maxIndex = j.`
Parameters	- `Vector:` original vector - `numElems:` number of elements in vector(s) - `MaxIndex:` index of maximum value
Returns	Minimum value (`maxVal`).
Requires	Nothing.

Example

```
dim vec1 as word[3]
    index, rslt as word

rslt = Vector_Max(vec1, 3, index)
```

Notes

- [W0..W5] used, not restored

Vector_Dot

Prototype	`sub function Vector_Dot(dim byref v1, v2 as word[1024], dim numElems as word) as word`
Description	Function calculates vector dot product.
Parameters	- `v1:` first vector - `v2:` second vector - `numElems:` number of elements in vector(s)
Returns	Dot product value: $dotVal = \sum_{n=0}^{numElems-1} (srcV1[n] * srcV2[n])$
Requires	Nothing.

Example

```
dim vec1 as word[3]

rslt = Vector_Dot(vec1,vec1,3)
```

Notes

- [W0..W2] used, not restored
- [W4..W5] used, not restored
- AccuA used, not restored
- CORCON saved, used, restored

Vector_Correlate

Prototype	`sub procedure Vector_Correlate(dim byref v1, v2, dest as word[1024], dim numElemsV1, dim numElemsV2 as word)`
Description	Function calculates Vector correlation (using convolution). $r[n]=\sum_{k=0}^{N-1}(x[k]*y[k+n])$ where: `x[n]` defined for n є [0, N) `y[n]` defined for n є [0, M), M ≤ N `r[n]` defined for n є [0, N+M-1)
Parameters	`- v1:` first vector `- v2:` second vector `- numElemsV1:` number of the first vector elements `- numElemsV2:` number of the second vector elements `- dest:` result vector
Returns	Nothing.
Requires	Nothing.

Example

```
dim vec1 as word[3]
    vConDest as word[10]

Vector_Correlate(vec1,vec1,vConDest,3,3)
```

Notes	[W0..W7] used, not restored

Vector_Convolve

Prototype	`sub procedure Vector_Convolve(dim byref v1, v2, dest as word[1024], dim numElemsV1, numElemsV2 as word)`
Description	Function calculates Vector using convolution. $y[n]=\sum_{k=0}^{n}(x[k]*h[n-k])$, n є [0, M) $y[n]=\sum_{(k=n-M+1)}^{n}(x[k]*h[n-k])$, n є [M, N) $y[n]=\sum_{k=n-M+1}^{N-1}x[k]*h[n-k]$, n є [N, N+M-1)
Parameters	- `v1:` first vector - `v2:` second vector - `numElemsV1:` number of the first vector elements - `numElemsV2:` number of the second vector elements - `dest:` result vector
Returns	Nothing.
Requires	Nothing.

Example

```
dim vec1 as word[3]
    vConDest2 as word[10]

Vector_Convolve(vec1,vec1,vConDest2,3,3)
```

Notes	- [W0..W7] used, not restored - [W8..W10] saved, used, restored - AccuA used, not restored - CORCON saved, used, restored

Vector_Add

Prototype	`sub procedure Vector_Add(dim byref dest, v1, v2 as word[256], dim numElems as word)`
Description	Function calculates vector addition. **dstV[n] = srcV1[n] + srcV2[n]** , n є [0, numElems-1)
Parameters	- `v1:` first vector - `v2:` second vector - `numElemsV1:` number of vector(s) elements - `dest:` result vector
Returns	Nothing.
Requires	Nothing.

Example

```
dim vec1    as word[3]
    vec2    as word[3]
    vecDest as word[3]

Vector_Add(vecDest, vec1, vec2, 3)
```

Notes	- [W0..W4] used, not restored - AccuA used, not restored - CORCON saved, used, restored

Matrices Library

mikroBasic PRO for dsPIC30/33 and PIC24 includes a library for operating and working with matrices. All routines work with fractional Q15 format.

Library Routines

Matrix_Transpose
Matrix_Subtract
Matrix_Scale
Matrix_Multiply
Matrix_Add

Matrix_Transpose

Prototype	`sub procedure Matrix_Transpose(dim byref src, dest as word[1024], dim numRows, numCols as word)`
Description	Function does matrix transposition. **dstM[i][j] = srcM[j][i]**
Parameters	- `src:` original matrix - `dest:` result matrix - `numRows:` number of rows in the source matrix - `numCols:` number of cols in the source matrix
Returns	Nothing.
Requires	Nothing.

Example

```
dim
  mx1 as word[6]
  mxDest as word[9]
...
Matrix_Transpose(mx1, mxDest, 2, 3)
```

Notes	[W0..W5] used, not restored

Matrix_Subtract

Prototype	`sub procedure Matrix_Subtract(dim byref src1, src2, dest as word[1024], dim numRows, numCols as word)`
Description	Function does matrix subtraction. **dstM[i][j] = srcM1[i][j] - srcM2[i][j]**
Parameters	- `src1:` first matrix - `src2:` second matrix - `dest:` result matrix - `numRows:` number of rows in the source matrix - `numCols:` number of cols in the source matrix
Returns	Nothing.
Requires	Nothing.

Example

```
dim
  mx1 as word[6]
  mx2 as word[6]
  mxDest as word[9]
...
Matrix_Subtract(mx1, mx2, mxDest, 2, 3)
```

Notes	- [W0..W4] used, not restored - AccuA used, not restored - AccuB used, not restored - CORCON saved, used, restored

Matrix_Scale

Prototype	`sub procedure Matrix_Scale(dim ScaleValue as word, dim byref src1, dest as word[1024], dim numRows, numCols as word)`
Description	Function does matrix scale. **dstM[i][j] = sclVal * srcM[i][j]**
Parameters	- `ScaleValue:` scale value - `src1:` original matrix - `dest:` result matrix - `numRows:` number of rows in the source matrix - `numCols:` number of cols in the source matrix
Returns	Nothing.
Requires	Nothing.

Example

```
dim
  mx1 as word[6]
  mxDest as word[9]
...
Matrix_Scale(0x4000, mx1, mxDest, 2,3)
```

Notes	- [W0..W5] used, not restored - AccuA used, not restored - CORCON saved, used, restored - `numRows*numCols <` 2^{14}

Matrix_Multiply

Prototype

```
sub procedure Matrix_Multiply(dim byref src1, src2, dest as word[256], dim
numRows1, numCols2, numCols1Rows2 as word)
```

Description

Function does matrix multiplication.

$$dstM[i][j] = \sum_{(i,j,k)} srcM1[i][k] * srcM2[k][j]$$

with:
i є [0, numRows1-1]
j є [0, numCols2-1]
k є [0, numCols1Rows2-1]

Parameters

- `src1:` first matrix
- `src2:` second matrix
- `dest:` result matrix
- `numRows1:` number of rows in the first matrix
- `numCols2:` number of columns in the second matrix
- `numCols1Rows2:` number of columns in the first matrix and rows in the second matrix

Returns

Nothing.

Requires

Nothing.

Example

```
dim
  mx1 as word[6]
  mx2 as word[6]
  mxDest as word[9]
...
Matrix_Multiply(mx1,mx2,mxDest,2,2,3)
```

Notes

- [W0..W7] used, not restored
- [W8..W13] used, and restored
- AccuA used, not restored
- CORCON saved, used, restored

Matrix_Add

Prototype	`sub procedure Matrix_Add(dim byref src1, src2, dest as word[1024], dim numRows, numCols as word)`
Description	Function does matrix addition. **dstM[i][j] = srcM1[i][j] + srcM2[i][j]**
Parameters	- `src1:` first matrix - `src2:` second matrix - `dest:` result matrix - `numRows1:` number of rows in the first matrix - `numCols2:` number of columns in the second matrix
Returns	Nothing.
Requires	Nothing.

Example

```
dim
  mx1 as word[6]
  mx2 as word[6]
  mx3 as word[6]
...
Matrix_Add(mx1,mx2,mxDest,2,3)
```

Notes	- [W0..W4] used, not restored - AccuA used, not restored. - CORCON saved, used, restored. - `numRows1*numCols2 <` 2^{14}

Miscellaneous Libraries

- Button Library
- Conversions Library
- C Type Library
- Setjmp Library
- String Library
- Time Library
- Trigon Library
- Trigonometry Library

Button Library

The Button Library provides routines for detecting button presses and debouncing (eliminating the influence of contact flickering upon pressing a button)

Library Routines

- Button

Button

Prototype	`sub function Button(dim byref port as word, dim pin as byte, dim time as word, dim ActiveState as byte) as word`
Description	The function eliminates the influence of contact flickering upon pressing a button (debouncing). The Button pin is tested just after the function call and then again after the debouncing period has expired. If the pin was in the active state in both cases then the function returns 255 (true).
Parameters	- `port:` button port address - `pin:` button pin - `time:` debouncing period in milliseconds - `active_state:` determines what is considered as active state. Valid values: `0` (logical zero) and `1` (logical one)
Returns	-`255` if the pin was in the active state for given period. - `0` otherwise
Requires	Nothing.

Example

```
program Button_Test

dim oldstate as bit

main:
  oldstate = 0
  ADPCFG = 0xFFFF                         ' initialize AN pins as digital
  TRISD = 0xFFFF                              ' initialize portd as input
  TRISB = 0x0000                              ' initialize portb as output

  while TRUE
    if (Button(PORTD, 0, 1, 1)) then     ' detect logical one on RB0 pin
      oldstate = 1
    end if
    if (oldstate and Button(PORTD, 0, 1, 0)) then
      LATB = not LATB                    ' invert value of PORTB
      oldstate = 0
    end if
  wend                                   ' endless loop
end.
```

Notes	None.

C Type Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a set of library functions for testing and mapping characters.

Library Functions

- isalnum
- isalpha
- iscntrl
- isdigit
- isgraph
- islower
- ispunct
- isspace
- isupper
- isxdigit
- toupper
- tolower

isalnum

Prototype	`sub function isalnum(dim character as byte) as byte`
Description	Function returns 0xFF if the `character` is alphanumeric (A-Z, a-z, 0-9), otherwise returns zero.
Example	`res = isalnum("o")    ' returns 0xFF` `res = isalnum("\r")   ' returns 0`

isalpha

Prototype	`sub function isalpha(dim character as byte) as word`
Description	Function returns 0xFF if the `character` is alphabetic (A-Z, a-z), otherwise returns zero.
Example	`res = isalpha("A")   ' returns 0xFF` `res = isalpha("1")   ' returns 0`

iscntrl

Prototype	`sub function iscntrl(dim character as byte) as word`
Description	Function returns 0xFF if the `character` is a control or delete character(decimal 0-31 and 127), otherwise returns zero.
Example	`res = iscntrl("\r")   ' returns 0xFF` `res = iscntrl("o")    ' returns 0`

isdigit

Prototype	`sub function isdigit(dim character as byte) as word`
Description	Function returns 0xFF if the `character` is a digit (0-9), otherwise returns zero.
Example	`res = isdigit(“o”)   ‘ returns 0xFF` `res = isdigit(“1”)   ‘ returns 0`

isgraph

Prototype	`sub function isgraph(dim character as byte) as word`
Description	Function returns 0xFF if the `character` is a printable, excluding the space (decimal 32), otherwise returns zero.
Example	`res = isgraph(“o”)   ‘ returns 0xFF` `res = isgraph(“ ”)   ‘ returns 0`

islower

Prototype	`sub function islower(dim character as byte) as word`
Description	Function returns 0xFF if the `character` is a lowercase letter (a-z), otherwise returns zero.
Example	`res = islower(“0”)   ‘ returns 0xFF` `res = islower(“A”)   ‘ returns 0`

ispunct

Prototype	`sub function ispunct(dim character as byte) as word`
Description	Function returns 0xFF if the `character` is a punctuation (decimal 32-47, 58-63, 91-96, 123-126), otherwise returns zero.
Example	`res = ispunct(“.”)   ‘ returns 0xFF` `res = ispunct(“1”)   ‘ returns 0`

isspace

Prototype	`sub function isspace(dim character as byte) as word`
Description	Function returns 0xFF if the `character` is a white space (space, tab, CR, HT, VT, NL, FF), otherwise returns zero.
Example	`res = isspace(“ ”)   ‘ returns 0xFF` `res = isspace(“1”)   ‘ returns 0`

isupper

Prototype	`sub function isupper(dim character as byte) as word`
Description	Function returns 0xFF if the `character` is an uppercase letter (A-Z), otherwise returns zero.
Example	`res = isupper(“A”)   ‘ returns 0xFF` `res = isupper(“a”)   ‘ returns 0`

isxdigit

Prototype	`sub function isxdigit(dim character as byte) as word/p>`
Description	Function returns 0xFF if the `character` is a hex digit (0-9, A-F, a-f), otherwise returns zero.
Example	`res = isxdigit(“A”)   ‘ returns 0xFF` `res = isxdigit(“P”)   ‘ returns 0`

toupper

Prototype	`sub function toupper(dim character as byte) as byte`
Description	If the `character` is a lowercase letter (a-z), the function returns an uppercase letter. Otherwise, the function returns an unchanged input parameter.
Example	`res = toupper(“a”)   ‘ returns A` `res = toupper(“B”)   ‘ returns B`

tolower

Prototype	`sub function tolower(dim character as byte) as byte`
Description	If the `character` is an uppercase letter (A-Z), function returns a lowercase letter. Otherwise, function returns an unchanged input parameter.
Example	`res = tolower(“A”)   ‘ returns a` `res = tolower(“b”)   ‘ returns b`

Conversions Library

mikroBasic PRO for dsPIC30/33 and PIC24 Conversions Library provides routines for numerals to strings and BCD/decimal conversions.

Library Dependency Tree

Library Routines

You can get text representation of numerical value by passing it to one of the following routines:

- ByteToStr
- ShortToStr
- WordToStr
- IntToStr
- LongIntToStr
- LongWordToStr
- FloatToStr

- WordToStrWithZeros
- IntToStrWithZeros
- LongWordToStrWithZeros
- LongIntToStrWithZeros

- ByteToHex
- ShortToHex
- WordToHex
- IntToHex
- LongWordToHex
- LongIntToHex

- StrToInt
- StrToWord

The following functions convert decimal values to BCD and vice versa:

- Bcd2Dec
- Dec2Bcd
- Bcd2Dec16
- Dec2Bcd16

ByteToStr

Prototype	`sub procedure ByteToStr(dim input as byte, dim byref output as string[3])`
Description	Converts input byte to a string. The output string is right justified and remaining positions on the left (if any) are filled with blanks.
Parameters	- `input:` byte to be converted - `output:` destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as byte
    txt as string[3]
...
t = 24
ByteToStr(t, txt)   ' txt is “ 24” (one blank here)
```

Notes	None.

ShortToStr

Prototype	`sub procedure ShortToStr(dim input as short, dim byref output as string[4])`
Description	Converts input short (signed byte) number to a string. The output string is right justified and remaining positions on the left (if any) are filled with blanks.
Parameters	- `input:` short number to be converted - `output:` destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as short
    txt as string[4]
...
t = -24
ByteToStr(t, txt)   ' txt is “ -24” (one blank here)
```

Notes	None.

WordToStr

Prototype	`sub procedure WordToStr(dim input as word, dim byref output as string[5])`
Description	Converts input word to a string. The output string is right justified and the remaining positions on the left (if any) are filled with blanks.
Parameters	- `input:` word to be converted - `output:` destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as word
    txt as string[5]
...
t = 437
WordToStr(t, txt)   ' txt is "  437" (two blanks here)
```

Notes	None.

WordToStrWithZeros

Prototype	`sub procedure WordToStrWithZeros(dim input as word, dim byref output as string[5])`
Description	Converts input word to a string. The output string is right justified and the remaining positions on the left (if any) are filled with zeros.
Parameters	- `input:` signed integer number to be converted - `output:` destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as word
    txt as string[5]
...
t = 437
WordToStrWithZeros(t, txt)   ' txt is '00437'
```

Notes	None.

IntToStr

Prototype	`sub procedure IntToStr(dim input as integer, dim byref output as string[6])`
Description	Converts input integer number to a string. The output string is right justified and the remaining positions on the left (if any) are filled with blanks.
Parameters	- `input:` integer number to be converted - `output:` destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim input as integer
    txt as string[6]
...
input = -4220
IntToStr(input, txt)    ' txt is “ -4220”
```

Notes	None.

LongintToStr

Prototype	`sub procedure LongintToStr(dim input as longint, dim byref output as string[11])`
Description	Converts input longint number to a string. The output string is right justified and the remaining positions on the left (if any) are filled with blanks.
Parameters	- `input:` longint number to be converted - `output:` destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim input as longint
    txt as string[11]
...
input = -12345678
IntToStr(input, txt)     ' txt is “  -12345678”
```

Notes	None.

LongWordToStr

Prototype	`sub procedure LongWordToStr(dim input as longword, dim byref output as string[10])`
Description	Converts input double word number to a string. The output string is right justified and the remaining positions on the left (if any) are filled with blanks.
Parameters	- `input:` double word number to be converted - `output:` destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim input as longint
    txt as string[10]
...
input = 12345678
IntToStr(input, txt)     ' txt is “  12345678”
```

Notes	None.

FloatToStr

Prototype	`sub procedure FloatToStr(dim fnum as float, dim byref str as string[20]) as byte`
Description	Converts a floating point number to a string. The output string is left justified and null terminated after the last digit.
Parameters	- `fnum`: floating point number to be converted - `str`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim ff1, ff2, ff3 as float
    txt as string[20]
  ...
  ff1 = -374.2
  ff2 = 123.456789
  ff3 = 0.000001234

  FloatToStr(ff1, txt)   ' txt is “-374.2”
  FloatToStr(ff2, txt)   ' txt is “123.4567”
  FloatToStr(ff3, txt)   ' txt is “1.234e-6”
```

Notes	Given floating point number will be truncated to 7 most significant digits before conversion.

WordToStrWithZeros

Prototype	`sub procedure WordToStrWithZeros(dim input as word, dim byref output as string[5])`
Description	Converts input word to a string. The output string is right justified and the remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: word to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as word
    txt as string[5]

t = 437
WordToStrWithZeros(t, txt)   ' txt is "00437"
```

Notes	None.

IntToStrWithZeros

Prototype	`sub procedure IntToStrWithZeros(dim input as integer, dim byref output as string[6])`
Description	Converts input integer to a string. The output string is right justified and the remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: word to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim  t as integer
    txt as string[6]

t = -3276
IntToStrWithZeros(t, txt)   ' txt is "-03276"
```

Notes	None.

LongWordToStrWithZeros

Prototype	`sub procedure LongWordToStrWithZeros(dim input as longword, dim byref output as string[10])`
Description	Converts input dword to a string. The output string is right justified and the remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: word to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as longword
    txt as string[10]

t = 12345678
LongWordToStrWithZeros(t, txt)   ' txt is "0012345678"
```

Notes	None.

LongIntToStrWithZeros

Prototype	`sub procedure LongIntToStrWithZeros(dim input as longint, dim byref output as string[11])`
Description	Converts input longint to a string. The output string is right justified and the remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: word to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as longint
    txt as string[11]

t = -12345678
LongIntToStrWithZeros(t, txt)   ' txt is "-0012345678"
```

Notes	None.

ByteToHex

Prototype	`dim procedure ByteToHex(dim input as byte, dim byref output as string[2])`
Description	Converts input number to a string containing the number’s hexadecimal representation. The output string is right justified and remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: byte to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as byte
    txt as string[2]

t = 2
ByteToHex(t, txt)   ' txt is "02"
```

Notes	None.

ShortToHex

Prototype	`sub procedure ShortToHex(dim input as short, dim byref output as string[2])`
Description	Converts input number to a string containing the number’s hexadecimal representation. The output string is right justified and remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: short number to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as short
    txt as string[2]
  ...
t = -100
ShortToHex(t, txt)   ' txt is "9C"
```

Notes	None.

WordToHex

Prototype	`sub procedure WordToHex(dim input as word, dim byref output as string[4])`
Description	Converts input number to a string containing the number's hexadecimal representation. The output string is right justified and remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: word to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim t as word
    txt as string[4]

t = 1111
WordToHex(t, txt)   ' txt is "0457"
```

Notes	None.

IntToHex

Prototype	`sub procedure IntToHex(dim input as integer, dim byref output as string[4])`
Description	Converts input number to a string containing the number's hexadecimal representation. The output string is right justified and remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: integer number to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim input as integer
    txt as string[4]

input = -32768
IntToHex(input, txt)    ' txt is "8000"
```

Notes	None.

LongWordToHex

Prototype	`sub procedure LongWordToHex(dim input as longword, dim byref output as string[8])`
Description	Converts input number to a string containing the number's hexadecimal representation. The output string is right justified and remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: double word number to be converted - `output`: destination string
Returns	Nothing.

Example

```
dim input as longword
    txt as string[8]

input = 65535
LongWordToHex(input, txt)     ' txt is "0000FFFF"
```

Notes	None.

LongIntToHex

Prototype	`sub procedure LongIntToHex(dim input as longint, dim byref output as string[8])`
Description	Converts input number to a string containing the number's hexadecimal representation. The output string is right justified and remaining positions on the left (if any) are filled with zeros.
Parameters	- `input`: longint number to be converted - `output`: destination string
Returns	Nothing.
Requires	Nothing.

Example

```
dim input as longint
    txt as string[8]

input = -2147483648
LongIntToHex(input, txt)     ' txt is "80000000"
```

Notes	None.

StrToInt

Prototype	`sub function StrToInt(dim byref input as string[6]) as integer`
Description	Converts a string to an integer.
Parameters	- `input`: string to be converted
Returns	Integer variable.
Requires	Input string is assumed to be the correct representation of a number. The conversion will end with the first character which is not a decimal digit.

Example

```
dim ii as integer

main:
  ...
  ii = StrToInt(“-1234”)
end.
```

Notes	None.

StrToWord

Prototype	`sub function StrToWord(dim byref input as string[5]) as word`
Description	Converts a string to word.
Parameters	- `input`: string to be converted
Returns	Word variable.
Requires	Input string is assumed to be the correct representation of a number. The conversion will end with the first character which is not a decimal digit.

Example

```
dim ww as word

main:
  ...
  ww = StrToword(“65432”)
end.
```

Notes	None.

Bcd2Dec

Prototype	`sub function Bcd2Dec(dim bcdnum as byte) as byte`
Description	Converts input BCD number to its appropriate decimal representation.
Parameters	- `bcdnum`: number to be converted
Returns	Converted decimal value.
Requires	Nothing.

Example

```
dim a, b as byte
...
a = 22
b = Bcd2Dec(a)  ' b equals 34
```

Notes	None.

Dec2Bcd

Prototype	`sub function Dec2Bcd(dim decnum as byte) as byte`
Description	Converts input number to its appropriate BCD representation.
Parameters	- `decnum`: number to be converted
Returns	Converted BCD value.
Requires	Nothing.

Example	`dim a, b as byte`
	`...`
	`a = 22`
	`b = Dec2Bcd(a)  ' b equals 34`

Notes	None.

Bcd2Dec16

Prototype	`sub function Bcd2Dec16(dim bcdnum as word) as word`
Description	Converts 16-bit BCD numeral to its decimal equivalent.
Parameters	- `bcdnum` 16-bit BCD numeral to be converted
Returns	Converted decimal value.
Requires	Nothing.

Example	`dim a, b as word`
	`...`
	`a = 0x1234         ' a equals 4660`
	`b = Bcd2Dec16(a)   ' b equals 1234`

Notes	None.

Dec2Bcd16

Prototype	`sub function Dec2Bcd16(dim decnum as word) as word`
Description	Converts decimal value to its BCD equivalent.
Parameters	- `decnum` decimal number to be converted
Returns	Converted BCD value.
Requires	Nothing.

Example	`dim a, b as word`
	`...`
	`a = 2345`
	`b = Dec2Bcd16(a)   ' b equals 9029`

Notes	None.

Setjmp Library

The Setjmp library contains functions and types definitions for bypassing the normal function call and return discipline.

Library Routines

- Setjmp
- Longjmp

Setjmp

Prototype	`sub function setjmp(dim byref env as word[4]) as integer`
Returns	- `0` if the return is from direct invocation - `nonzero value` if the return is from a call to `Longjmp` (this value will be set by the `Longjmp` routine)
Description	This function saves calling position for a later use by longjmp. Parameters: - `env`: buffer suitable for holding information needed for restoring calling environment
Requires	Nothing.

Example	`dim buf as word[4]`
	`...`
	`Setjmp(buf)`

Notes	None.

Longjmp

Prototype	`sub procedure longjmp(dim byref env as word[4], dim val as integer)`
Returns	Nothing.
Description	Restores calling environment saved in the `env` buffer by the most recent invocation of `setjmp`. If there has been no such invocation, or the function containing the invocation of setjmp has terminated in the interim, the behavior is undefined. Parameters: - `env`: buffer holding the information saved by the corresponding `setjmp` invocation - `val`: value to be returned by the corresponding `setjmp` function
Requires	Invocation of `longjmp` must occur before return from the function in which `setjmp` was called encounters.

Example	`dim buf as word[4]`
	`...`
	`Longjmp(buf, 2)`

Library Example

Example demonstrates function cross calling using setjmp and longjmp functions. When called, Setjmp() saves its calling environment in its **buf** argument for later use by the Longjmp(). Longjmp(), on the other hand, restores the environment saved by the most recent invocation of the Setjmp() with the corresponding **buf** argument.

Copy Code To Clipboard

```
program Setjmp

dim buf as word[4]             ' Note: Program flow diagrams are indexed according
                               '       to the sequence of execution

sub procedure func33()         '  2<------------|
  Delay_ms(1000)               '                |
                               '                |
  nop                          '                |
  longjmp(buf, 2)              '  3---------------->|
  nop                          '                |   |
                               '                |   |
end sub                        '                |   |
                               '                |   |
sub procedure func()           '  1<--------|   |   |
  PORTB = 3                    '            |   |   |
  if (setjmp(buf) = 2) then    '  3<----------------|
    PORTB = 1                  '  4-->|     |   |
  else                         '      |     |   |
    func33()                   '  2------------>|
  end if                       '      |     |
                               '  4<--|     |
end sub                        '  5----->|  |
                               '         |  |
main:                          '         |  |
  ADPCFG = 0xFFFF              '         |  |
                               '         |  |
  PORTB = 0                    '         |  |
  TRISB = 0                    '         |  |
                               '         |  |
  nop                          '         |  |
                               '         |  |
  func()                       '  1-------->|
                               '         |
  nop                          '  5<-----|
  Delay_ms(1000)
  PORTB = 0xFFFF
end.
```

String Library

mikroBasic PRO for dsPIC30/33 and PIC24 includes a library which automatizes string related tasks.

Library Functions

- memchr
- memcmp
- memcpy
- memmove
- memset
- strcat
- strcat2
- strchr
- strcmp
- strcpy
- strlen
- strncat
- strncpy
- strspn
- strncmp
- strstr
- strcspn
- strpbrk
- strrchr
- ltrim
- rtrim
- strappendpre
- strappendsuf
- length

memchr

Prototype	`sub function memchr(dim p as ^byte, dim ch as byte, dim n as word) as word`
Description	The function locates the first occurrence of the byte `ch` in the initial `n` words of memory area starting at the address p. The function returns the offset of this occurrence from the memory address `p` or `0xFFFF` if `ch` was not found. For the parameter `p` you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example `@mystring` or `@PORTB`.
Example	`txt = “mikroElektronika”` `res = memchr(@txt, “e”, 16)  ‘ example locates first occurrence of the letter ‘e’ in the string ‘txt’ in the first 16 characters of the string`

memcmp

Prototype	`sub function memcmp(dim p1, p2 as ^byte, dim n as word) as integer`

Description	The function returns a positive, negative, or zero value indicating the relationship of first `n` words of memory areas starting at addresses `p1` and `p2`.
	
	This function compares two memory areas starting at addresses `p1` and `p2` for `n` words and returns a value indicating their relationship as follows:
	
	`Value     Meaning`
	`< 0       p1 “less than” p2`
	`= 0       p1 “equal to” p2`
	`> 0       p1 “greater than” p2`
	
	The value returned by the function is determined by the difference between the values of the first pair of words that differ in the strings being compared.
	
	For parameters `p1` and `p2` you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example `@mystring` or `@PORTB`.

Example	`txt = “mikroElektronika”`
	`txt_sub = “mikro”`
	
	`res = memcmp(@txt, @txt_sub, 16)  ‘ returns 69, which is ASCII code of the first differing character - letter ‘E’`

memcpy

Prototype	`sub procedure memcpy(dim p1, p2 as ^byte, dim nn as word)`
Description	The function copies `nn` words from the memory area starting at the address `p2` to the memory area starting at `p1`. If these memory buffers overlap, the `memcpy` function cannot guarantee that words are copied before being overwritten. If these buffers do overlap, use the memmove function. For parameters `p1` and `p2` you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example `@mystring` or `@PORTB`.

Example	`txt = “mikroElektronika”`
	`txt_sub = “mikr”`
	
	`memcpy(@txt+4, @txt_sub, 4)  ‘ string ‘txt’ will be populated with the first 4 characters of the ‘txt_sub’ string, beginning from the 4th character`

memmove

Prototype	`sub procedure memmove(dim p1, p2, as ^byte, dim nn as word)`
Description	The function copies `nn` words from the memory area starting at the address `p2` to the memory area starting at `p1`. If these memory buffers overlap, the `Memmove` function ensures that the words in `p2` are copied to `p1` before being overwritten. For parameters `p1` and `p2` you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example `@mystring` or `@PORTB`.

Example

```
txt = “mikroElektronika”
txt_sub = “mikr”

memmove(@txt+7, @txt_sub, 4)   ‘ string ‘txt’ will be populated with first 4
characters of the ‘txt_sub’ string, beginning from the 7th character
```

memset

Prototype	`sub procedure memset(dim p as ^byte, dim character as byte, dim n as word)`
Description	The function fills the first `n` words in the memory area starting at the address `p` with the value of word `character`. For parameter `p` you can use either a numerical value (literal/variable/constant) indicating memory address or a dereferenced value of an object, for example `@mystring` or `@PORTB`.

Example

```
txt = “mikroElektronika”

memset(@txt, “a”, 2)  ‘ routine will copy the character ‘a’ into each of the
first ‘n’ characters of the string ‘txt”,
```

strcat

Prototype	`sub procedure strcat(dim byref s1, s2 as string)`
Description	The function appends the value of string `s2` to string `s1` and terminates `s1` with a null character.

Example

```
txt = “mikroElektronika”
txt_sub = “mikr”

txt[3] = 0
strcat(txt, “_test”)  ‘ routine will append the ‘_test’ at the place of the
first null character, adding terminating null character to the result
```

strcat2

Prototype

```
sub procedure strcat2(dim byref l, s1, s2 as string)
```

Description

The procedure adjoins string `s2` at the end of the string `s1`, or at the first null character of the `s1`, and places the result string into `l` string.

Example

```
dim txt as string[16]
    txt_sub as string[5]
    l as string[21]

strcat2(l, txt, txt_sub)  ‘ routine will adjoin strings txt and txt_sub and
place the result into l
```

strchr

Prototype

```
sub function strchr(dim byref s as string, dim ch as byte) as word
```

Description

The function searches the string `s` for the first occurrence of the character `ch`. The null character terminating `s` is not included in the search.

The function returns the position (index) of the first character `ch` found in `s`; if no matching character was found, the function returns `0xFFFF`.

Example

```
txt = “mikroElektronika”

res = strchr(txt, “E”)  ‘ routine will locate the character ‘E’ in the ‘txt’
string, and return the position of the character
```

strcmp

Prototype

```
sub function strcmp(dim byref s1, s2 as string) as integer
```

Description

The function lexicographically compares the contents of the strings `s1` and `s2` and returns a value indicating their relationship:

Value	**Meaning**
`< 0`	`s1 “less than” s2`
`= 0`	`s1 “equal to” s2`
`> 0`	`s1 “greater than” s2`

The value returned by the function is determined by the difference between the values of the first pair of words that differ in the strings being compared.

Example

```
txt = “mikroElektronika”
txt_sub = “mikr”

res = strcmp(txt,txt_sub)  ‘ compares strings ‘txt’ and ‘txt_sub’ and
returns returns a difference between the first differing characters, in this
case 69
```

strcpy

Prototype	`sub procedure strcpy(dim byref s1, s2 as string)`
Description	The function copies the value of the string `s2` to the string `s1` and appends a null character to the end of `s1`.

Example	`txt = “mikroElektronika”`
	`txt_sub = “mikr”`
	
	`strcpy(txt,txt_sub)   ‘ copies string ‘txt_sub’ to ‘txt’`

strlen

Prototype	`sub function strlen(dim byref s as string) as word`
Description	The function returns the length, in words, of the string `s`. The length does not include the null terminating character.

Example	`txt = “mikroElektronika”`
	
	`res = strlen(txt)  ‘ calculates the length of the ‘txt’ string, result = 16`

strncat

Prototype	`sub procedure strncat(dim byref s1, s2 as string, dim size as word)`
Description	The function appends at most `size` characters from the string `s2` to the string `s1` and terminates `s1` with a null character. If `s2` is shorter than the `size` characters, `s2` is copied up to and including the null terminating character.

Example	`txt = “mikroElektronika”`
	`txt_sub = “mikr”`
	`txt[5] = 0`
	
	`strncat(txt,txt_sub,4)  ‘ routine appends first 4 characters from the string ‘txt_sub’ at the place of first null character in the ‘txt’ string`

strncpy

Prototype	`sub procedure strncpy(dim byref s1, s2 as string, dim size as word)`
Description	The function copies at most `size` characters from the string `s2` to the string `s1`. If `s2` contains fewer characters than `size`, `s1` is padded out with null characters up to the total length of the `size` characters.

Example	`txt = “mikroElektronika”`
	`txt_sub = “mikr”`
	
	`strncpy(txt,txt_sub,4)  ‘ copies first 4 characters form the string ‘txt_sub’ to ‘txt’`

strspn

Prototype

```
sub function strspn(dim byref s1, s2 as string) as word
```

Description

The function searches the string `s1` for characters *not* found in the `s2` string.

The function returns the index of first character located in `s1` that does not match a character in `s2`. If the first character in `s1` does not match a character in `s2`, a value of 0 is returned. If all characters in `s1` are found in `s2`, the length of `s1` is returned (not including the terminating null character).

Example

```
txt = "mikroElektronika"
txt_sub = "mikr"

res = strspn(txt,txt_sub)  ' routne returns 4
```

strncmp

Prototype

```
sub function strncmp(dim byref s1, s2 as string, dim len as byte) as integer
```

Description

The function lexicographically compares the first `len` characters of the strings `s1` and `s2` and returns a value indicating their relationship:

Value	Meaning
`< 0`	`s1 "less than" s2`
`= 0`	`s1 "equal to" s2`
`> 0`	`s1 "greater than" s2`

The value returned by the function is determined by the difference between the values of the first pair of words that differ in the strings being compared (within first `len` words).

Example

```
txt = "mikroElektronika"
txt_sub = "mikr"

res = strncmp(txt_sub,txt,3)  ' compares the first 3 characters from the string 'txt' with the sting 'txt_sub' and returns a difference
```

strstr

Prototype

```
sub function strstr(dim byref s1, s2 as string) as word
```

Description

The function locates the first occurrence of the string `s2` in the string `s1` (excluding the terminating null character).

The function returns a number indicating the position of the first occurrence of `s2` in `s1`; if no string was found, the function returns 0xFFFF. If `s2` is a null string, the function returns `0`.

Example

```
txt = "mikroElektronika"
txt_sub = "mikr"

res = strstr(txt_sub,txt)
```

strcspn

Prototype	`sub function strcspn(dim byref s1, s2 as string) as word`
Description	The function searches the string `s1` for any of the characters in the string `s2`. The function returns the index of the first character located in `s1` that matches any character in `s2`. If the first character in `s1` matches a character in `s2`, a value of 0 is returned. If there are no matching characters in `s1`, the length of the string is returned (not including the terminating null character).

Example	`txt = “mikroElektronika”`
	`txt_sub = “mikr”`
	
	`res = strcspn(txt_sub,txt)`

strpbrk

Prototype	`sub function strpbrk(dim byref s1, s2 as string) as word`
Description	The function searches `s1` for the first occurrence of any character from the string `s2`. The null terminator is not included in the search. The function returns an index of the matching character in `s1`. If `s1` contains no characters from `s2`, the function returns `0xFFFF`.

Example	`txt = “mikroElektronika”`
	`txt_sub = “mikr”`
	
	`res = strpbrk(txt_sub,txt)`

strrchr

Prototype	`sub function strrchr(dim byref s as string, dim ch as byte) as word`
Description	The function searches the string `s` for the last occurrence of the character `ch`. The null character terminating `s` is not included in the search. The function returns an index of the last `ch` found in `s`; if no matching character was found, the function returns `0xFFFF`.

Example	`txt = “mikroElektronika”`
	
	`res = strrchr(txt,”k”)   ‘ returns the index of the ‘k’ character of the ‘txt’ string`

ltrim

Prototype	`sub procedure ltrim(dim byref astring as string)`
Description	The procedure trims the leading spaces of the string.
Example	`txt = “  mikroE”` `ltrim(txt)   ‘ trims the leading 2 spaces of the ‘txt’ string`

rtrim

Prototype	`sub procedure rtrim(dim byref astring as string)`
Description	The procedure trims the trailing spaces of the string.

Example	`txt = “mikroE  “`
	
	`rtrim(txt)  ‘ trims the trailing 2 spaces of the ‘txt’ string and adds terminating null character to the result`

strappendpre

Prototype	`sub procedure strappendpre(dim letter as char, dim byref s1 as string)`
Description	The procedure appends character at the beginning of the string.

Example	`txt = “ikroE”`
	
	`strappendpre(“m”,txt)  ‘ adds letter ‘m’ at the beginning of the ‘txt’ string`

strappendsuf

Prototype	`sub procedure strappendsuf(dim byref s1 as string, dim letter as char)`
Description	The procedure appends character at the end of the string.

Example	`txt = “mikro”`
	
	`strappendsuf(“E”,txt)  ‘ adds letter ‘E’ at the end of the ‘txt’ string`

length

Prototype	`sub function length(dim byref s as string) as word`
Description	The function returns length of passed string.

Example	`txt = “mikroE”`
	
	`res = length(txt)  ‘ calculates and returns the length of the ‘txt’ string`

Time Library

The Time Library contains functions and type definitions for time calculations in the UNIX time format which counts the number of seconds since the “epoch”. This is very convenient for programs that work with time intervals: the difference between two UNIX time values is a real-time difference measured in seconds.

What is the epoch?
Originally it was defined as the beginning of 1970 GMT. (January 1, 1970 Julian day) GMT, Greenwich Mean Time, is a traditional term for the time zone in England.

The TimeStruct type is a structure type suitable for time and date storage.

Library Routines

- Time_dateToEpoch
- Time_epochToDate
- Time_dateDiff

Time_dateToEpoch

Prototype	`sub function Time_dateToEpoch(dim byref ts as TimeStruct) as longint`
Description	This function returns the UNIX time : number of seconds since January 1, 1970 0h00mn00s.
Parameters	- `ts`: time and date value for calculating UNIX time.
Returns	Number of seconds since January 1, 1970 0h00mn00s.
Requires	Nothing.
Example	see code below
Notes	None.

Example

```
dim ts1 as TimeStruct
    Epoch as longint
...
' what is the epoch of the date in ts ?
epoch = Time_dateToEpoch(@ts1)
```

Time_epochToDate

Prototype	`sub procedure Time_epochToDate(dim e as longint, dim byref ts as TimeStruct)`
Description	Converts the UNIX time to time and date.
Parameters	- `e`: UNIX time (seconds since UNIX epoch) - `ts`: time and date structure for storing conversion output
Returns	Nothing.
Requires	Nothing.

Example

```
dim ts2 as TimeStruct
    epoch as longint
...
' what date is epoch 1234567890 ?
epoch = 1234567890
Time_epochToDate(epoch,ts2)
```

Notes	None.

Time_dateDiff

Prototype	`sub function Time_dateDiff(dim byref t1, t2 as TimeStruct) as longint`
Description	This function compares two dates and returns time difference in seconds as a signed long. Result is positive if `t1` is before `t2`, result is null if `t1` is the same as `t2` and result is negative if `t1` is after `t2`.
Parameters	- `t1`: time and date structure (the first comparison parameter) - `t2`: time and date structure (the second comparison parameter)
Parameters	None.
Returns	Time difference in seconds as a signed long.
Requires	Nothing.

Example

```
dim ts1, ts2 as TimeStruct
    diff as longint
...
' how many seconds between these two dates contained in ts1 and ts2
buffers?
diff = Time_dateDiff(ts1, ts2)
```

Notes	None.

Library Example

Demonstration of Time library routines usage for time calculations in UNIX time format.

Copy Code To Clipboard

```
program Time_Demo
   '*
   '* simple time structure
   '*
  structure TimeStruct
     dim ss as byte      ' seconds
     dim mn as byte       ' minutes
     dim hh as byte       ' hours
     dim md as byte       ' day in month, from 1 to 31
     dim wd as byte       ' day in week, monday=0, tuesday=1, .... sunday=6
     dim mo as byte       ' month number, from 1 to 12 (and not from 0 to 11 as with unix
C time !)
     dim yy as word      ' year Y2K compliant, from 1892 to 2038
  end structure

  dim  ts1, ts2    as TimeStruct
       epoch, diff as longint

  main:
    ts1.ss = 0
    ts1.mn = 7
    ts1.hh = 17
    ts1.md = 23
    ts1.mo = 5
    ts1.yy = 2006

    '*
    '* what is the epoch of the date in ts ?
    '*
    epoch = Time_dateToEpoch(@ts1)       ' epoch = 1148404020

    '*
    '* what date is epoch 1234567890 ?
    '*

    epoch = 1234567890
    Time_epochToDate(epoch, @ts2)       ' ts2.ss = 30
                                        ' ts2.mn = 31
                                        ' ts2.hh = 23
                                        ' ts2.md = 13
                                        ' ts2.wd = 4
                                        ' ts2.mo = 2
                                        ' ts2.yy = 2009

    '*
    '* how much seconds between this two dates ?
```

```
'*
    diff = Time_dateDiff(@ts1, @ts2)      ' diff = 86163870

  end.
```

TimeStruct type definition

```
structure TimeStruct
     dim ss as byte      ' seconds
     dim mn as byte      ' minutes
     dim hh as byte      ' hours
     dim md as byte      ' day in month, from 1 to 31
     dim wd as byte      ' day in week, monday=0, tuesday=1, .... sunday=6
     dim mo as byte      ' month number, from 1 to 12 (and not from 0 to 11 as with unix
C time !)
     dim yy as word      ' year Y2K compliant, from 1892 to 2038
  end structure
```

Trigon Library

The mikroBasic PRO for dsPIC30/33 and PIC24 provides a set of library functions for floating point math handling. See also Predefined Globals and Constants for the list of predefined math constants.

Library Routines

- acos
- asin
- atan
- atan2
- ceil
- cos
- cosh
- eval_poly
- exp
- fabs
- floor
- frexp
- ldexp
- log
- log10
- modf
- pow
- sin
- sinh
- sqrt
- tan
- tanh

acos

Prototype	`sub function acos(dim x as float) as float`
Description	Function returns the arc cosine of parameter `x`; that is, the value whose cosine is `x`. The input parameter `x` must be between -1 and 1 (inclusive). The return value is in radians, between 0 and Π (inclusive).
Example	`res = acos(0.5)   ' res = 1.047198`

asin

Prototype	`sub function asin(dim x as float) as float`
Description	Function returns the arc sine of parameter `x`; that is, the value whose sine is `x`. The input parameter `x` must be between -1 and 1 (inclusive). The return value is in radians, between -Π/2 and Π/2 (inclusive).
Example	`res = asin(0.5)   ' res = 5.235987e-1`

atan

Prototype	`sub function atan(dim arg as float) as float`
Description	Function computes the arc tangent of parameter `f`; that is, the value whose tangent is `f`. The return value is in radians, between -Π/2 and Π/2 (inclusive).
Example	`res = atan(1.0)   ' res = 7.853982e-1`

atan2

Prototype	`sub function atan2(dim y as float, dim x as float) as float`
Description	This is the two-argument arc tangent function. It is similar to computing the arc tangent of `y/x`, except that the signs of both arguments are used to determine the quadrant of the result and `x` is permitted to be zero. The return value is in radians, between -Π and Π (inclusive).
Example	`res = atan2(2., 1.)   ' res = 4.636475e-1`

ceil

Prototype	`sub function ceil(dim x as float) as float`
Description	Function returns value of parameter `x` rounded up to the next whole number.
Example	`res = ceil(0.5)   ' res = 1.000000`

cos

Prototype	`sub function cos(dim arg as float) as float`
Description	Function returns the cosine of `f` in radians. The return value is from -1 to 1.
Example	`res = cos(PI/3.)   ' res = 0.500008`

cosh

Prototype	`sub function cosh(dim x as float) as float`
Description	Function returns the hyperbolic cosine of `x`, defined mathematically as $(e^x+e^{-x})/2$. If the value of `x` is too large (if overflow occurs), the function fails.
Example	`res = cosh(PI/3.)   ' res = 1.600286`

eval_poly

Prototype	`sub function eval_poly(dim x as float, dim byref d as array[10] of float, dim n as integer) as float`
Description	Function Calculates polynom for number `x`, with coefficients stored in `d[]`, for degree `n`.

exp

Prototype	`sub function exp(dim x as float) as float`
Description	Function returns the value of e — the base of natural logarithms — raised to the power `x` (i.e. e^x).
Example	`res = exp(0.5)    ' res = 1.648721`

fabs

Prototype	`sub function fabs(dim d as float) as float`
Description	Function returns the absolute (i.e. positive) value of `d`.
Example	`res = fabs(-1.3)   ' res = 1.3`

floor

Prototype	`sub function floor(dim x as float) as float`
Description	Function returns the value of parameter x rounded down to the nearest integer.
Example	`res = floor(15.258)   ' res = 15.000000`

frexp

Prototype	`sub function frexp(dim value as float, dim byref eptr as integer) as float`
Description	The function splits a floating-point value `value` into a normalized fraction and an integral power of 2. The return value is a normalized fraction and the integer exponent is stored in the object pointed to by `eptr`.

ldexp

Prototype	`sub function ldexp(dim value as float, dim newexp as integer) as float`
Description	Function returns the result of multiplying the floating-point number `num` by 2 raised to the power `n` (i.e. returns $x * 2^n$).
Example	`res = ldexp(2.5, 2)   ' res = 10`

log

Prototype	`sub function log(dim x as float) as float`
Description	Function returns the natural logarithm of `x` (i.e. $log_e(x)$).
Example	`res = log(10)   ' res = 2.302585E`

log10

Prototype	`sub function log10(dim x as float) as float`
Description	Function returns the base-10 logarithm of `x` (i.e. $\log_{10}(x)$).
Example	`res = log10(100.)   ' res = 2.000000`

modf

Prototype	`sub function modf(dim val as float, dim byref iptr as float) as float`
Description	Returns argument `val` split to the fractional part (function return val) and integer part (in number iptr).
Example	`res = modf(6.25, iptr)   ' res = 0.25, iptr = 6.00`

pow

Prototype	`sub function pow(dim x as float, dim y as float) as float`
Description	Function returns the value of `x` raised to the power `y` (i.e. x^y). If `x` is negative, the function will automatically cast `y` into `unsigned long`.
Example	`res = pow(10.,5.)   ' res = 9.999984e+4`

sin

Prototype	`sub function sin(dim arg as float) as float`
Description	Function returns the sine of `f` in radians. The return value is from -1 to 1.
Example	`res = sin(PI/2.)   ' res = 1.000000`

sinh

Prototype	`sub function sinh(dim x as float) as float`
Description	Function returns the hyperbolic sine of `x`, defined mathematically as $(e^x-e^{-x})/2$. If the value of `x` is too large (if overflow occurs), the function fails.
Example	`res = sinh(PI/2.)   ' res = 2.301296`

sqrt

Prototype	`sub function sqrt(dim x as float) as float`
Description	Function returns the non negative square root of `x`.
Example	`res = sqrt(10000.)   ' res = 100.0000`

tan

Prototype	`sub function tan(dim x as float) as float`
Description	Function returns the tangent of `x` in radians. The return value spans the allowed range of floating point in the mikroBasic PRO for dsPIC30/33 and PIC24.
Example	`res = tan(PI/4.)   ' res = 0.999998`

tanh

Prototype	`sub function tanh(dim x as float) as float)`
Description	Function returns the hyperbolic tangent of `x`, defined mathematically as `sinh(x)/cosh(x)`.
Example	`res = tanh(-PI/4.)    ' res = -0.655793`

Trigonometry Library

The mikroBasic PRO for dsPIC30/33 and PIC24 implements fundamental trigonometry functions. These functions are implemented as look-up tables. Trigonometry functions are implemented in integer format in order to save memory.

Library Routines

- sinE3
- cosE3

sinE3

Prototype	`sub function sinE3(dim angle_deg as word) as integer`
Description	The function calculates sine multiplied by 1000 and rounded to the nearest integer: `result = round(sin(angle_deg)*1000)`
Parameters	- `angle_deg`: input angle in degrees
Returns	The function returns the sine of input parameter multiplied by 1000.
Requires	Nothing.

Example	`dim res as integer`
	`...`
	`res = sinE3(45)   ' result is 707`

Notes	Return value range: `-1000..1000.`

cosE3

Prototype	`sub function cosE3(dim angle_deg as word) as integer`
Description	The function calculates cosine multiplied by 1000 and rounded to the nearest integer: `result = round(cos(angle_deg)*1000)`
Parameters	- `angle_deg`: input angle in degrees
Returns	The function returns the sine of input parameter multiplied by 1000.
Requires	Nothing.

Example	`dim res as integer`
	`...`
	`res = cosE3(196)   ' result is -193`

Notes	Return value range: `-1000..1000.`

CHAPTER 10

Tutorials

Managing Project

Projects

The mikroBasic PRO for dsPIC30/33 and PIC24 organizes applications into projects, consisting of a single project file (extension `.mbpds`) and one or more source files (extension `.mbas`). mikroBasic PRO for dsPIC30/33 and PIC24 IDE allows you to manage multiple projects (see Project Manager). Source files can be compiled only if they are part of a project.

The project file contains the following information:

- project name and optional description,
- target device,
- device flags (config word),
- device clock,
- list of the project source files with paths,
- binary files (*.mcl),
- image files,
- other files.

Note that the project does not include files in the same way as preprocessor does, see Add/Remove Files from Project.

New Project

The easiest way to create a project is by means of the New Project Wizard, drop-down menu **Project › New Project** or by clicking the New Project Icon from Project Toolbar.

New Project Wizard Steps

Start creating your New project, by clicking Next button:

New Project Wizard

Welcome to the New Project Wizard

This wizard helps you:

- Create a new project
- Select the device for your project
- Setup device clock
- Add project files

Click Next to continue

Step One - Select the device from the device drop-down list:

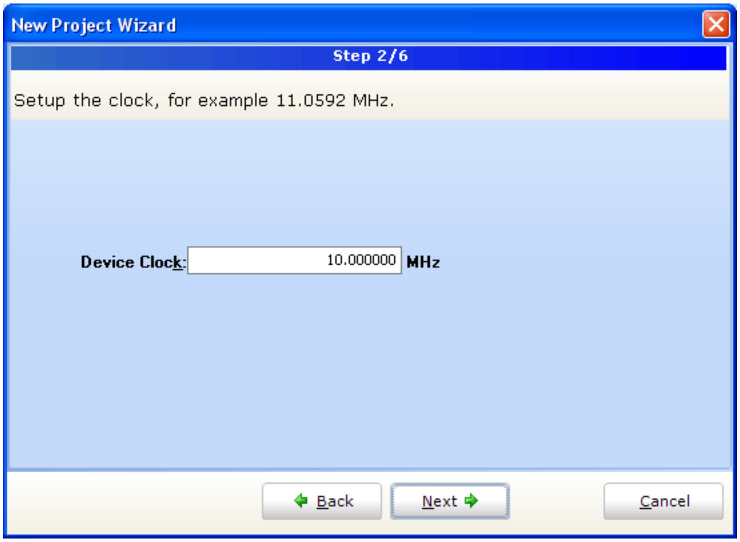
New Project Wizard

Step 1/6

Select the device you want to use.

Device Name: P30F4013

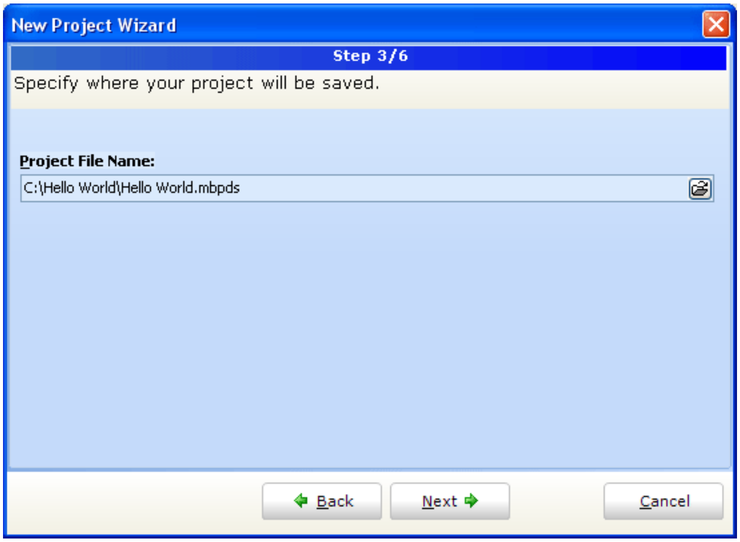
Step Two - Enter the oscillator frequency value:



Setup the clock, for example 11.0592 MHz.

Device Clock: 10.000000 MHz

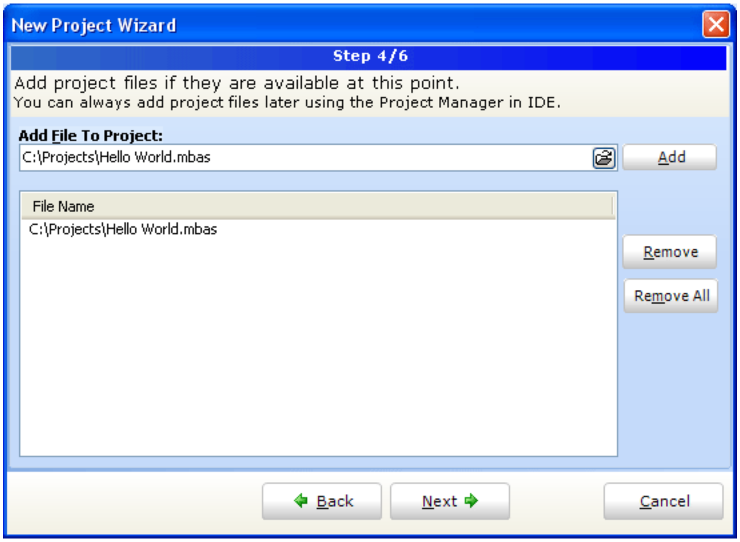
Step Three - Specify the location where your project will be saved:



Specify where your project will be saved.

Project File Name: C:\Hello World\Hello World.mbpds

Step Four - Add project file to the project if they are avaiable at this point. You can always add project files later using Project Manager:


New Project Wizard

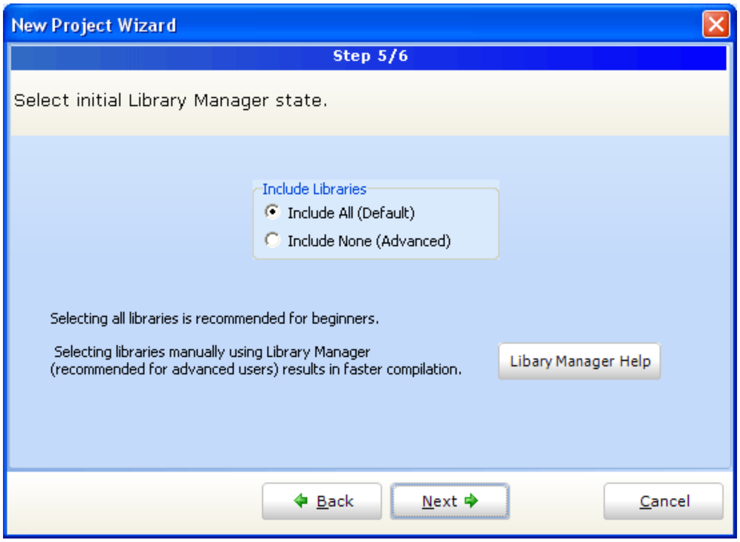
Step 4/6

Add project files if they are available at this point.
You can always add project files later using the Project Manager in IDE.

Add File To Project: C:\Projects\Hello World.mbas

File Name
C:\Projects\Hello World.mbas

Step Five - Select inital Library Manager state:


New Project Wizard

Step 5/6

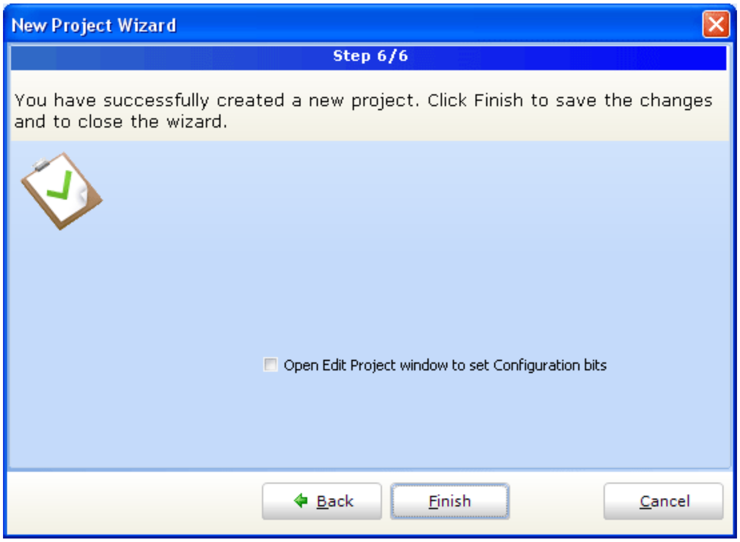
Select initial Library Manager state.

Include Libraries
● Include All (Default)
○ Include None (Advanced)

Selecting all libraries is recommended for beginners.

Selecting libraries manually using Library Manager
(recommended for advanced users) results in faster compilation.

Step Six - Click Finish button to create your New Project:



You have successfully created a new project. Click Finish to save the changes and to close the wizard.

☐ Open Edit Project window to set Configuration bits

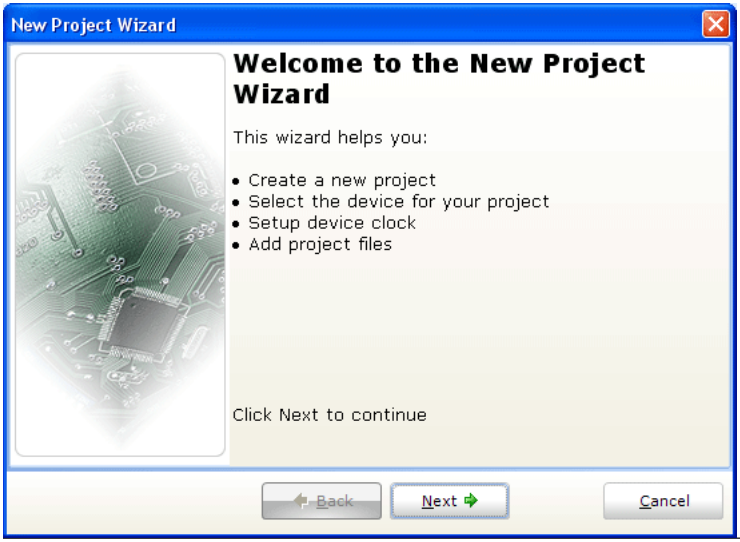
Related topics: Project Manager, Project Settings

New Project

The easiest way to create a project is by means of the New Project Wizard, drop-down menu **Project › New Project** or by clicking the New Project Icon from Project Toolbar.

New Project Wizard Steps

Start creating your New project, by clicking Next button:



Welcome to the New Project Wizard

This wizard helps you:

- Create a new project
- Select the device for your project
- Setup device clock
- Add project files

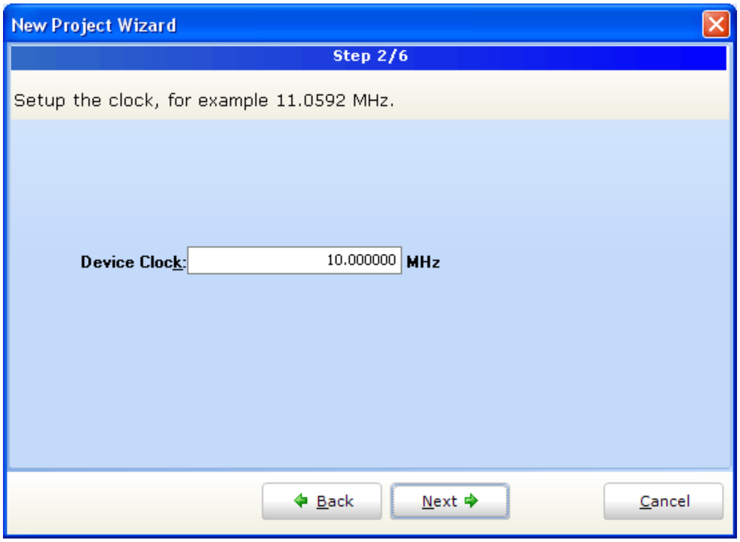
Click Next to continue

Step One - Select the device from the device drop-down list:

Select the device you want to use.

Device Name: P30F4013

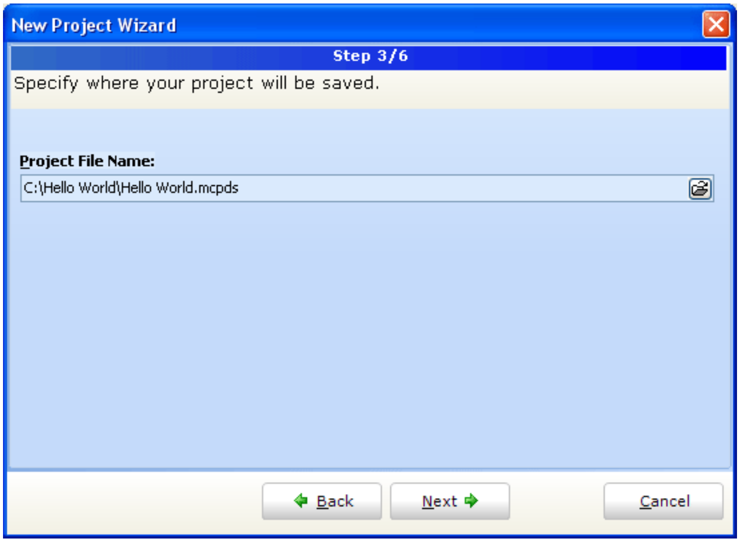
Step Two - Enter the oscillator frequency value:



Setup the clock, for example 11.0592 MHz.

Device Clock: 10.000000 MHz

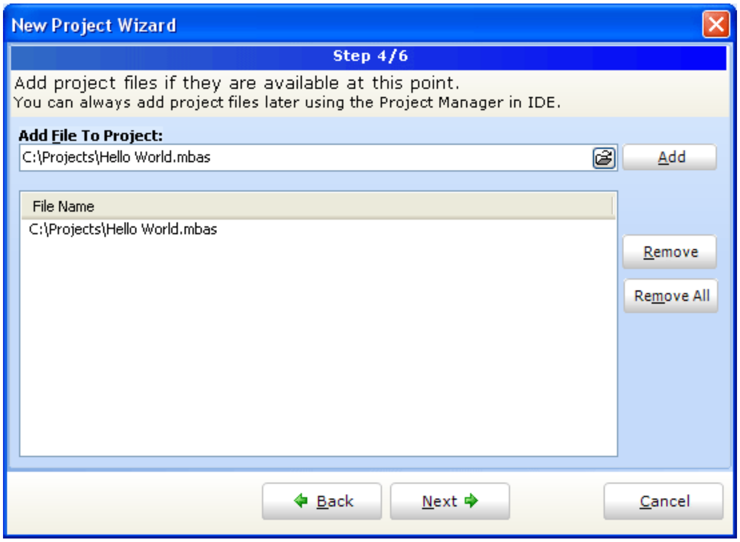
Step Three - Specify the location where your project will be saved:



Specify where your project will be saved.

Project File Name: C:\Hello World\Hello World.mcpds

Step Four - Add project file to the project if they are avaiable at this point. You can always add project files later using Project Manager:

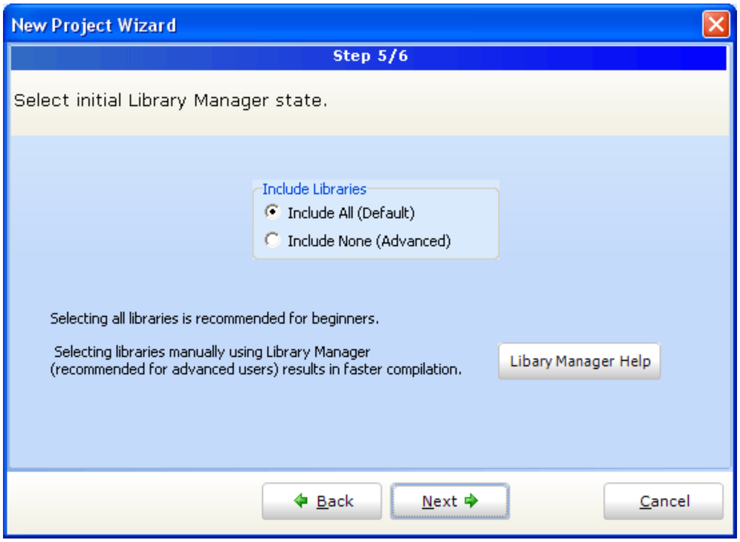


Add project files if they are available at this point.
You can always add project files later using the Project Manager in IDE.

Add File To Project: C:\Projects\Hello World.mbas

File Name
C:\Projects\Hello World.mbas

Step Five - Select inital Library Manager state:



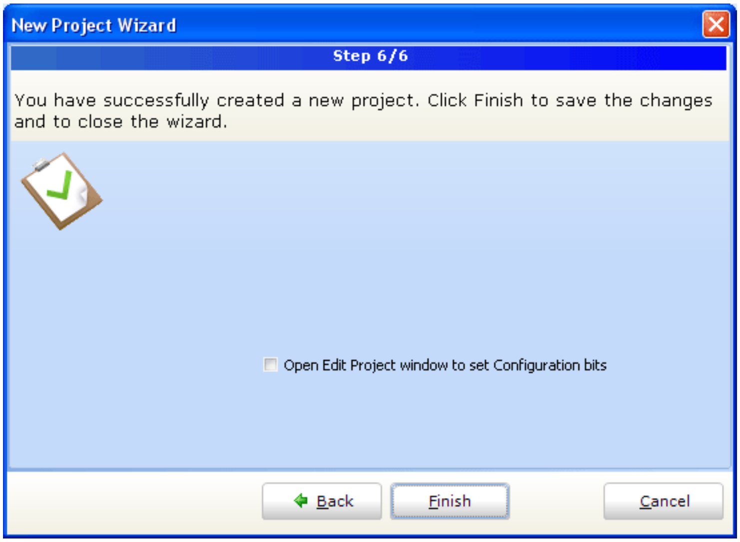
Select initial Library Manager state.

● Include All (Default)
○ Include None (Advanced)

Selecting all libraries is recommended for beginners.

Selecting libraries manually using Library Manager (recommended for advanced users) results in faster compilation.

Step Six - Click Finish button to create your New Project:



You have successfully created a new project. Click Finish to save the changes and to close the wizard.

☐ Open Edit Project window to set Configuration bits

Related topics: Project Manager, Project Settings

Customizing Projects

You can change basic project settings in the Project Settings window, like chip and oscillator frequency. Any change in the Project Setting Window affects currently active project only, so in case more than one project is open, you have to ensure that exactly the desired project is set as active one in the Project Manager. Also, you can change configuration bits of the selected chip in the Edit Project window.

Managing Project Group

mikroBasic PRO for dsPIC30/33 and PIC24 IDE provides convenient option which enables several projects to be open simultaneously. If you have several projects being connected in some way, you can create a project group.

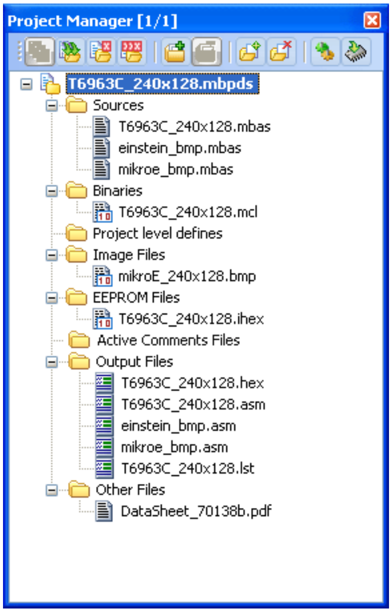
The project group may be saved by clicking the Save Project Group Icon from the Project Manager window.

The project group may be reopened by clicking the Open Project Group Icon . All relevant data about the project group is stored in the project group file (extension `.mbdsgroup`)

Add/Remove Files from Project

The project can contain the following file types:

- `.mbas` source files
- `.mcl` binary files
- `.pld` project level defines files
- image files
- `.ihex` EEPROM files
- `.hex`, `.asm` and `.lst` files, see output files. These files can not be added or removed from project.
- other files



Project Manager [1/1]

- T6963C_240x128.mbpds
  - Sources
    - T6963C_240x128.mbas
    - einstein_bmp.mbas
    - mikroe_bmp.mbas
  - Binaries
    - T6963C_240x128.mcl
  - Project level defines
  - Image Files
    - mikroE_240x128.bmp
  - EEPROM Files
    - T6963C_240x128.ihex
  - Active Comments Files
  - Output Files
    - T6963C_240x128.hex
    - T6963C_240x128.asm
    - einstein_bmp.asm
    - mikroe_bmp.asm
    - T6963C_240x128.lst
  - Other Files
    - DataSheet_70138b.pdf

The list of relevant files is stored in the project file (extension `.mbpds`).

To add a file to the project, click the Add File to Project Icon or press Insert button on your keyboard. Each added source file must be self-contained, i.e. it must have all necessary definitions after preprocessing.

To remove file(s) from the project, click the Remove File from Project Icon or press Delete button on your keyboard.

Note: For inclusion of the module files, use the `include` clause. See File Inclusion for more information.

Project Level Defines:

Project Level Defines (`.pld`) files can also be added to project. Project level define files enable you to have defines that are visible in all source files in the project. A file must contain one definition per line in the following form:

```
ANALOG
DEBUG
TEST
```

For example, lets make a project level define named `pld_test`. First of all, create a new file with the `.pld` extension, `pld_test_file.pld`.
Next, open it, and write something like this :

```
PLD_TEST
```

Once you have done this, save the file. In the Project Manager, add `pld_test_file.pld` file by right-clicking the Project Level Defines node.

In the source code write the following:

```
#IFDEF PLD_TEST
...
#ENDIF
```

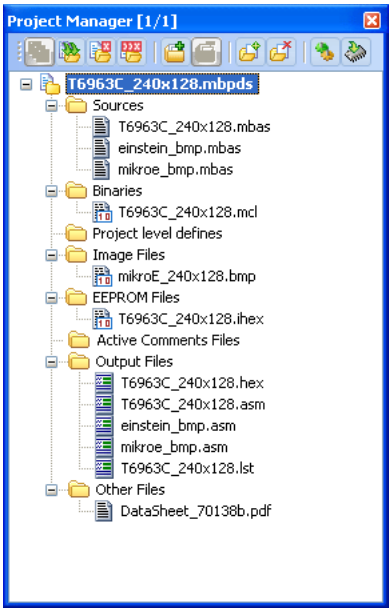
There are number of predefined project level defines. See predefined project level defines

Related topics: Project Manager, Project Settings, Edit Project

Add/Remove Files from Project

The project can contain the following file types:

- `.mbas` source files
- `.mcl` binary files
- `.pld` project level defines files
- image files
- `.ihex` EEPROM files
- `.hex`, `.asm` and `.lst` files, see output files. These files can not be added or removed from project.
- other files



- T6963C_240x128.mbpds
  - Sources
    - T6963C_240x128.mbas
    - einstein_bmp.mbas
    - mikroe_bmp.mbas
  - Binaries
    - T6963C_240x128.mcl
  - Project level defines
  - Image Files
    - mikroE_240x128.bmp
  - EEPROM Files
    - T6963C_240x128.ihex
  - Active Comments Files
  - Output Files
    - T6963C_240x128.hex
    - T6963C_240x128.asm
    - einstein_bmp.asm
    - mikroe_bmp.asm
    - T6963C_240x128.lst
  - Other Files
    - DataSheet_70138b.pdf

The list of relevant files is stored in the project file (extension `.mbpds`).

To add a file to the project, click the Add File to Project Icon or press Insert button on your keyboard. Each added source file must be self-contained, i.e. it must have all necessary definitions after preprocessing.

To remove file(s) from the project, click the Remove File from Project Icon or press Delete button on your keyboard.

Note: For inclusion of the module files, use the include clause. See File Inclusion for more information.

Project Level Defines:

Project Level Defines (`.pld`) files can also be added to project. Project level define files enable you to have defines that are visible in all source files in the project. A file must contain one definition per line in the following form:

```
ANALOG
DEBUG
TEST
```

For example, lets make a project level define named `pld_test`. First of all, create a new file with the `.pld` extension, `pld_test_file.pld`.
Next, open it, and write something like this:

```
PLD_TEST
```

Once you have done this, save the file. In the Project Manager, add `pld_test_file.pld` file by right-clicking the Project Level Defines node.

In the source code write the following:

```
#IFDEF PLD_TEST
...
#ENDIF
```

There are number of predefined project level defines. See predefined project level defines

Related topics: Project Manager, Project Settings, Edit Project

Source Files

Source files containing source code should have the extension `.mbas`. The list of source files relevant to the application is stored in project file with extension `.mbpds`, along with other project information. You can compile source files only if they are part of the project.

Managing Source Files

Creating new source file

To create a new source file, do the following:

1. Select **File › New Unit** from the drop-down menu, or press Ctrl+N, or click the New File Icon from the File Toolbar.
2. A new tab will be opened. This is a new source file. Select **File › Save** from the drop-down menu, or press Ctrl+S, or click the Save File Icon from the File Toolbar and name it as you want.

If you use the New Project Wizard, an empty source file, named after the project with extension `.mbas`, will be created automatically. The mikroBasic PRO for dsPIC30/33 and PIC24 does not require you to have a source file named the same as the project, it's just a matter of convenience.

Opening an existing file

1. Select **File › Open** from the drop-down menu, or press Ctrl+O, or click the Open File Icon from the File Toolbar. In Open Dialog browse to the location of the file that you want to open, select it and click the Open button.
2. The selected file is displayed in its own tab. If the selected file is already open, its current Editor tab will become active.

Printing an open file

1. Make sure that the window containing the file that you want to print is the active window.
2. Select **File › Print** from the drop-down menu, or press Ctrl+P.
3. In the Print Preview Window, set a desired layout of the document and click the OK button. The file will be printed on the selected printer.

Saving file

1. Make sure that the window containing the file that you want to save is the active window.
2. Select **File › Save** from the drop-down menu, or press Ctrl+S, or click the Save File Icon from the File Toolbar.

Saving file under a different name

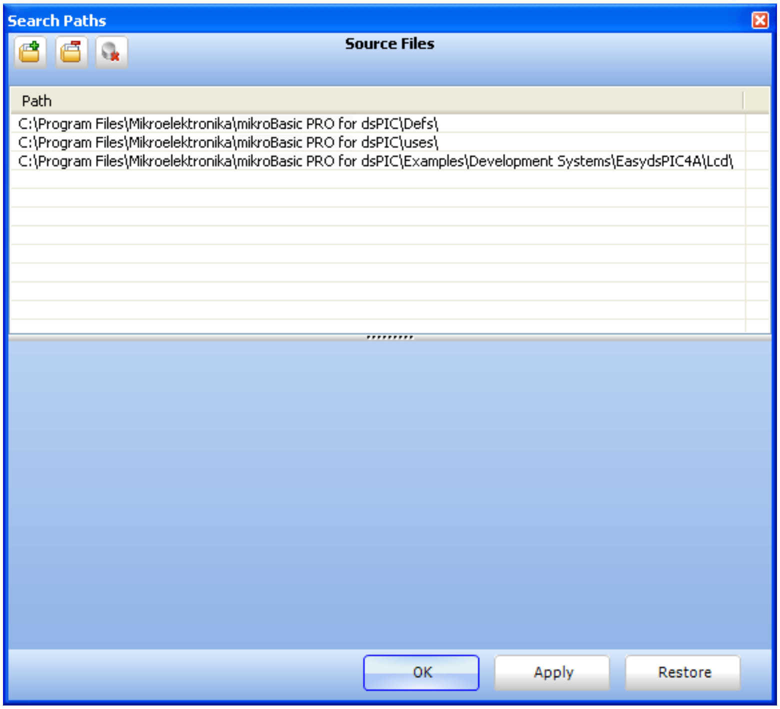
1. Make sure that the window containing the file that you want to save is the active window.
2. Select **File › Save As** from the drop-down menu. The New File Name dialog will be displayed.
3. In the dialog, browse to the folder where you want to save the file.
4. In the File Name field, modify the name of the file you want to save.
5. Click the Save button.

Closing file

1. Make sure that the tab containing the file that you want to close is the active tab.
2. Select **File › Close** from the drop-down menu, or right click the tab of the file that you want to close and select **Close** option from the context menu.
3. If the file has been changed since it was last saved, you will be prompted to save your changes.

Search Paths

You can specify your own custom search paths: select **Project › Edit Search Paths...** option from the drop-down menu:



C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Defs\
C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\uses\
C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Examples\Development Systems\EasydsPIC4A\Lcd\

Following options are available:

Icon	Description
	Add Search Path.
	Remove Search Path.
	Purge Invalid Paths.

Paths for Source Files (`.mbas`)

You can specify either absolute or relative path to the source file. If you specify a relative path, mikroBasic PRO for dsPIC30/33 and PIC24 will look for the file in following locations, in this particular order:

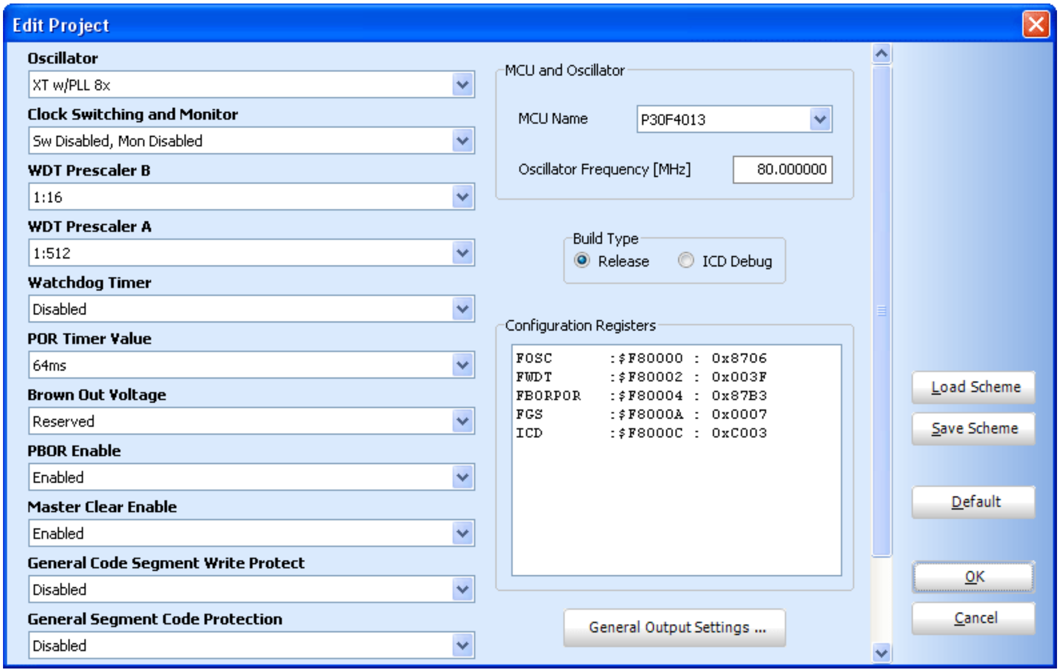
1. the project folder (folder which contains the project file `.mbpds`),
2. your custom search paths,
3. mikroBasic PRO for dsPIC30/33 and PIC24 installation folder > `Uses` folder.

Related topics:File Menu, File Toolbar, Project Manager, Project Settings,

Edit Project

Edit Project gives you option to change MCU you wish to use, change its oscillator frequency and build type. Also, Edit Project enables you to alter specific configuration bits of the selected device.

As you alter these bits, appropriate register values will be updated also. This can be viewed in the **Configuration Registers** pane.
When you have finished configuring your device, you can save bit configuration as a scheme,
using Save Scheme button.
In case you need this scheme in another project, you can load it using Load Scheme button.
There is also a Default button which lets you select default configuration bit settings for the selected device.



Oscillator: XT w/PLL 8x
Clock Switching and Monitor: Sw Disabled, Mon Disabled
WDT Prescaler B: 1:16
WDT Prescaler A: 1:512
Watchdog Timer: Disabled
POR Timer Value: 64ms
Brown Out Voltage: Reserved
PBOR Enable: Enabled
Master Clear Enable: Enabled
General Code Segment Write Protect: Disabled
General Segment Code Protection: Disabled

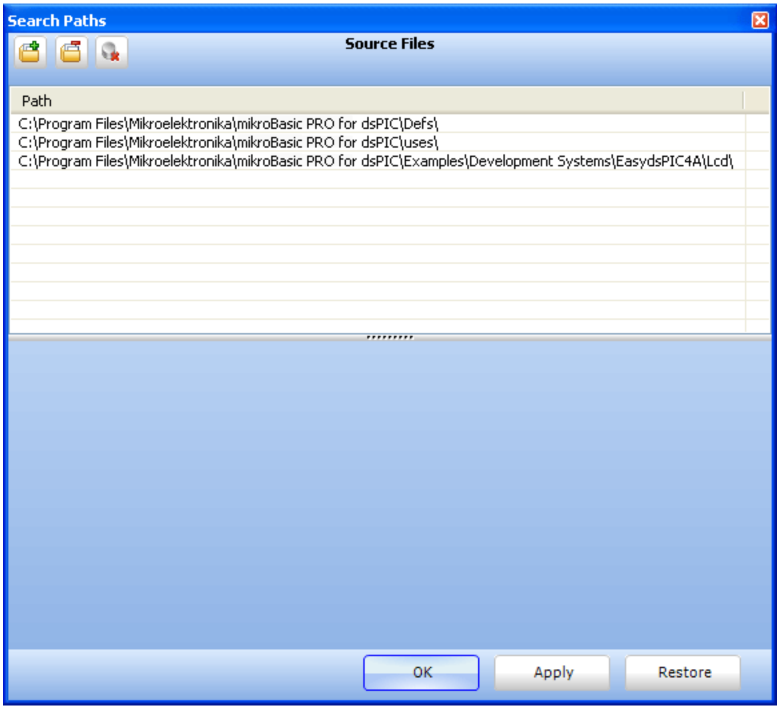
MCU Name: P30F4013
Oscillator Frequency [MHz]: 80.000000
● Release ○ ICD Debug

```
FOSC      :$F80000 : 0x8706
FWDT      :$F80002 : 0x003F
FBORPOR   :$F80004 : 0x87B3
FGS       :$F8000A : 0x0007
ICD       :$F8000C : 0xC003
```

Related topics: Project Settings, Customizing Projects

Search Paths

You can specify your own custom search paths: select **Project › Edit Search Paths...** option from the drop-down menu:



Path
C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Defs\
C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\uses\
C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Examples\Development Systems\EasydsPIC4A\Lcd\

Following options are available:

Icon	Description
	Add Search Path.
	Remove Search Path.
	Purge Invalid Paths.

Paths for Source Files (`.mbas`)

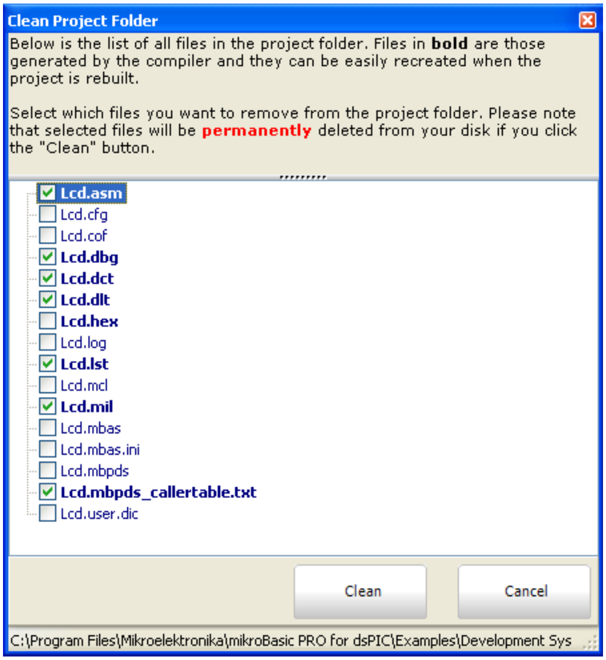
You can specify either absolute or relative path to the source file. If you specify a relative path, mikroBasic PRO for dsPIC30/33 and PIC24 will look for the file in following locations, in this particular order:

1. the project folder (folder which contains the project file `.mbpds`),
2. your custom search paths,
3. mikroBasic PRO for dsPIC30/33 and PIC24 installation folder > `Uses` folder.

Related topics:File Menu, File Toolbar, Project Manager, Project Settings,

Clean Project Folder

This menu gives you option to choose which files from your current project you want to delete.
Files marked in bold can be easily recreated by building a project. Other files should be marked for deletion only with a great care, because IDE cannot recover them.



Below is the list of all files in the project folder. Files in **bold** are those generated by the compiler and they can be easily recreated when the project is rebuilt.

Select which files you want to remove from the project folder. Please note that selected files will be **permanently** deleted from your disk if you click the "Clean" button.

- ☑ **Lcd.asm**
- ☐ Lcd.cfg
- ☐ Lcd.cof
- ☑ **Lcd.dbg**
- ☑ **Lcd.dct**
- ☑ **Lcd.dlt**
- ☐ **Lcd.hex**
- ☐ Lcd.log
- ☑ **Lcd.lst**
- ☐ Lcd.mcl
- ☑ **Lcd.mil**
- ☐ Lcd.mbas
- ☐ Lcd.mbas.ini
- ☐ Lcd.mbpds
- ☑ **Lcd.mbpds_callertable.txt**
- ☐ Lcd.user.dic

C:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Examples\Development Sys

Related topics: Customizing Projects

Compilation

When you have created the project and written the source code, it’s time to compile it. Select **Project › Build** from the drop-down menu, or click the Build Icon from the Build Toolbar. If more more than one project is open you

can compile all open projects by selecting **Project › Build All Projects** from the drop-down menu, or click the Build All Projects Icon from the Build Toolbar.

Progress bar will appear to inform you about the status of compiling. If there are some errors, you will be notified in the Messages Window. If no errors are encountered, the mikroBasic PRO for dsPIC30/33 and PIC24 will generate output files.

Output Files

Upon successful compilation, mikroBasic PRO for dsPIC30/33 and PIC24 will generate output files in the project folder (folder which contains the project file `.mbpds`). Output files are summarized in the table below:

Format	Description	File Type
Intel HEX	Intel style hex records. Use this file to program MCU.	`.hex`
Binary	mikro Compiled Library. Binary distribution of application that can be included in other projects.	`.mcl`
List File	Overview of MCU memory allotment: instruction addresses, registers, routines and labels.	`.lst`
Assembler File	Human readable assembly with symbolic names, extracted from the List File.	`.asm`

Assembly View

After compiling the program in the mikroBasic PRO for dsPIC30/33 and PIC24, you can click the View Assembly icon or select **View› View Assembly** from the drop-down menu to review the generated assembly code (`.asm` file) in a new tab window.
Assembly is human-readable with symbolic names.

Related topics: Build Menu, Build Toolbar, Messages Window, Project Manager, Project Settings

Creating New Library

mikroBasic PRO for dsPIC30/33 and PIC24 allows you to create your own libraries. In order to create a library in mikroBasic PRO for dsPIC30/33 and PIC24 follow the steps bellow:

1. Create a new source file, see Managing Source Files
2. Save the file in one of the subfolders of the compiler's Uses folder:
   `DriveName:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Uses\`
3. Write a code for your library and save it.
4. Add `__Lib_Example` file in some project, see Project Manager. Recompile the project.
   If you wish to use this library for all MCUs, then you should go to **Tools › Options › Output settings**, and check **Build all files as library** box.
   This will build libraries in a common form which will work with all MCUs. If this box is not checked, then library will be built for selected MCU.
   Bear in mind that compiler will report an error if a library built for specific MCU is used for another one.
5. Compiled file `__Lib_Example.mcl` should appear in `...\mikroBasic PRO for dsPIC\Uses\` folder.
6. Open the definition file for the MCU that you want to use. This file is placed in the compiler's Defs folder:
   `DriveName:\Program Files\Mikroelektronika\mikroBasic PRO for dsPIC\Defs\`
   and it is named `MCU_NAME.mlk`, for example `30F4013.mlk`
7. Add the the following segment of code to `<LIBRARIES>` node of the definition file (definition file is in XML format):

```
<LIB>
      <ALIAS>Example_Library</ALIAS>
      <FILE>__Lib_Example</FILE>
      <TYPE>REGULAR</TYPE>
</LIB>
```

8. Add Library to mlk file for each MCU that you want to use with your library.
9. Click Refresh button in Library Manager
10. `Example_Library` should appear in the Library manager window.

Multiple Library Versions

Library Alias represents unique name that is linked to corresponding Library `.mcl` file. For example UART library for 30F4013 is different from UART library for 30F6014 MCU. Therefore, two different UART Library versions were made, see `mlk` files for these two MCUs. Note that these two libraries have the same Library Alias (UART) in both `mlk` files. This approach enables you to have identical representation of UART library for both MCUs in Library Manager.

Related topics: Library Manager, Project Manager, Managing Source Files

Using Microchip MPLAB® IDE with mikroElektronika compilers

This new feature will boost your productivity by enabling you to import your code in a non-mikroElektronika eironment - Microchip's MPLAB®.
With the introduction of COFF File in mikroElektronika compiler, it is possible to debug and analyze your code through a software or hardware simulator.

Debugging Your Code

If your program has been built correctly, the compiler should generate a `.hex` file and a `.cof` file. The `cof` file contains all the information necessary for high-level debugging in MPLAB®, and it should be loaded by selecting the **File › Import...** menu in the MPLAB®.
Once you have done this, you have two choices: either to use MPLAB® ICD 2 Debugger, if you have the appropriate hardware, or MPLAB® Simulator.

Trademarks:
The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

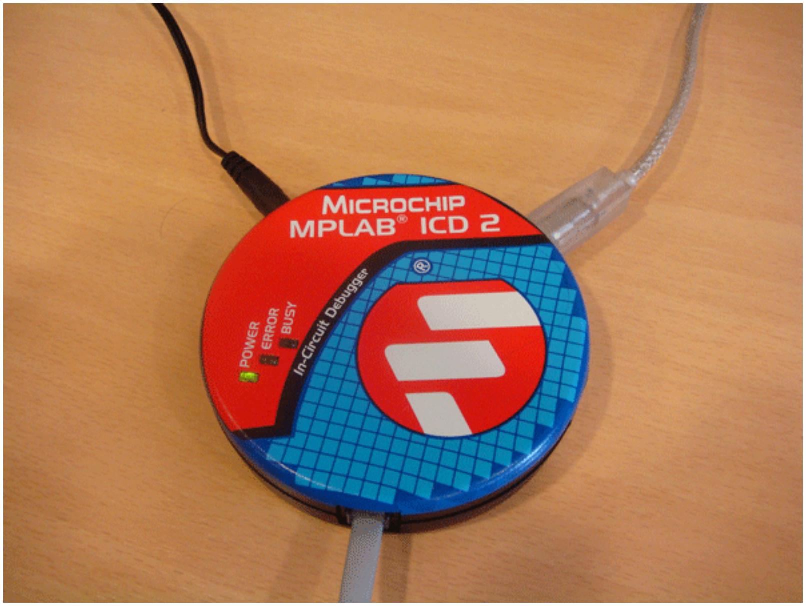
Related topics: COFF File, Using MPLAB® ICD 2 Debugger, Using MPLAB® Simulator

Using MPLAB® ICD 2 Debugger

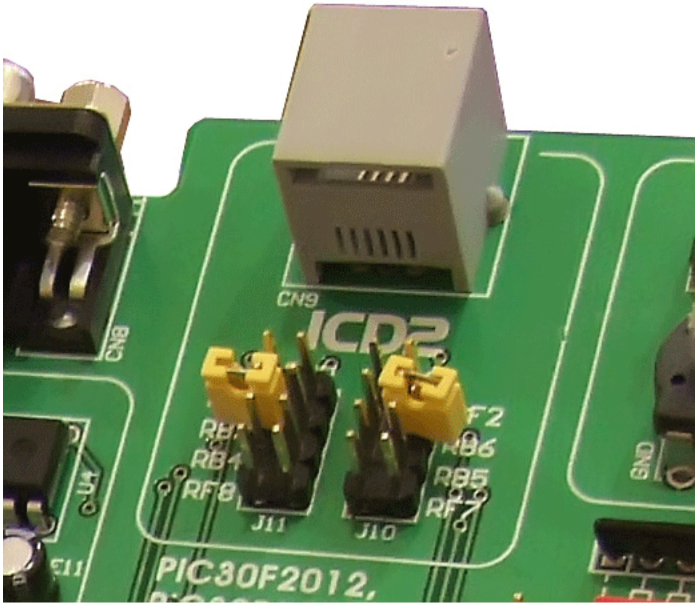
Important:
- It is assumed that MPLAB® and USB drivers for MPLAB® ICD 2 Debugger are previously installed.
- Procedure described below is also relevant for MPLAB® ICD 3 Debugger.
- Be sure to import compiled `.hex` file prior to importing `.cof` file, because it contains configuration bit settings which are essential for the proper functioning of the user code.

To successfully use MPLAB® ICD 2 Debugger with generated `.cof` file, follow the steps below:

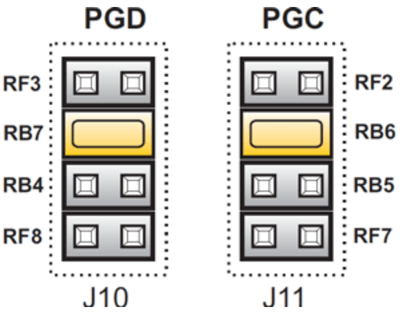
1. First of all, start mikroBasic PRO for dsPIC30/33 and PIC24 and open the desired project. In this example, UART project for EasydsPIC4A board and dsPIC30F4013 will be opened.
2. Open **Tools › Options › Output settings**, and check the "**Generate COFF file**" option, and click the OK button.
3. After that, compile the project by pressing Ctrl + F9.
4. Connect USB cable and turn on power supply on EasydsPIC4A.
5. Program the MCU by pressing F11.
6. Connect external power supply, USB cable from PC and modular interface cable to the MPLAB® ICD 2 Debugger's appropiate sockets, like on the picture below:



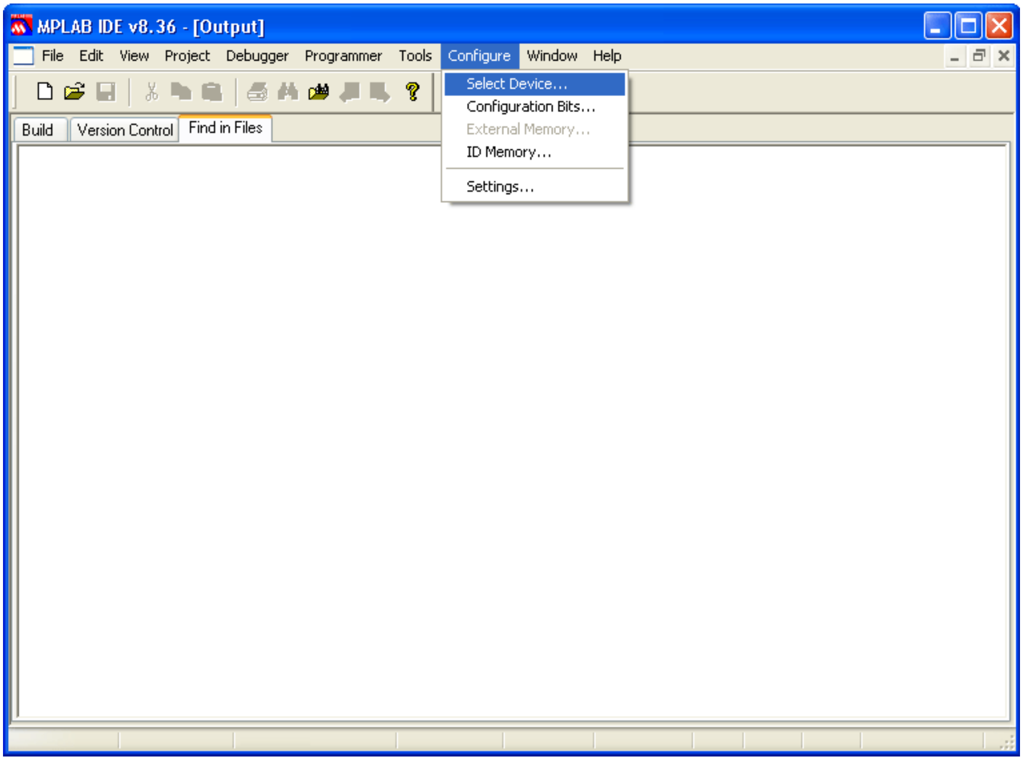
7. Connect second end of the modular interface cable to the **ICD** (RJ12) socket of EasydsPIC4A :



8. Put the J11 and J10 Jumpers in the correct position, as showed in the picture below:



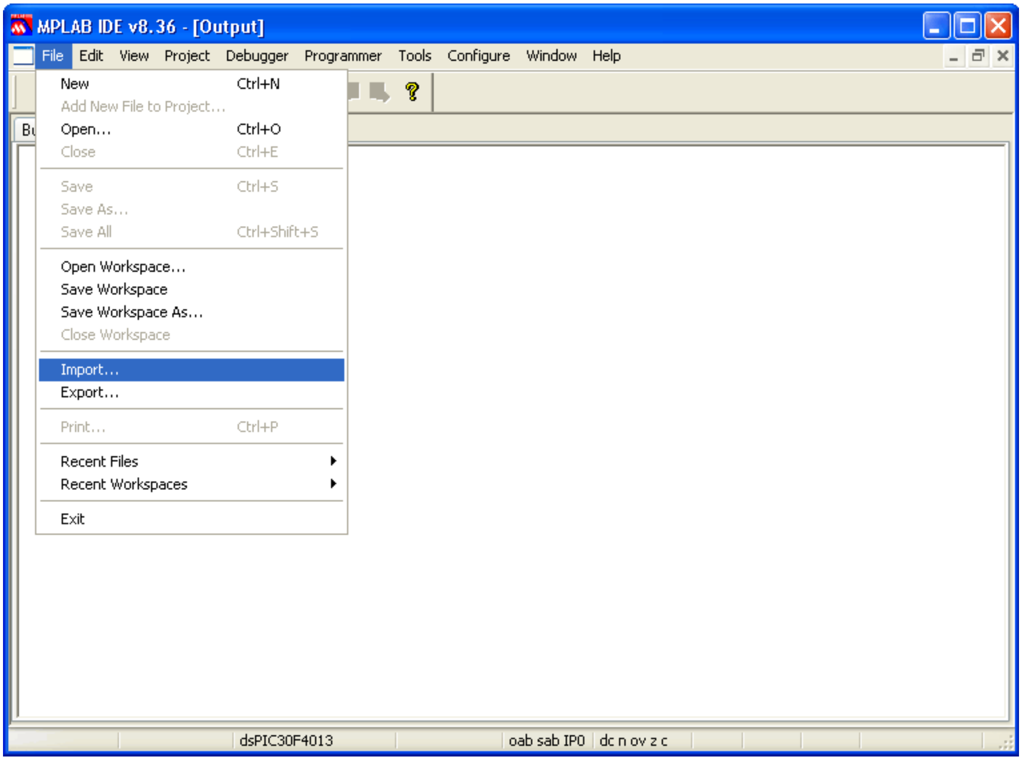
9. Next, open MPLAB®, and select the appropriate device by choosing **Configure › Select Device...** :



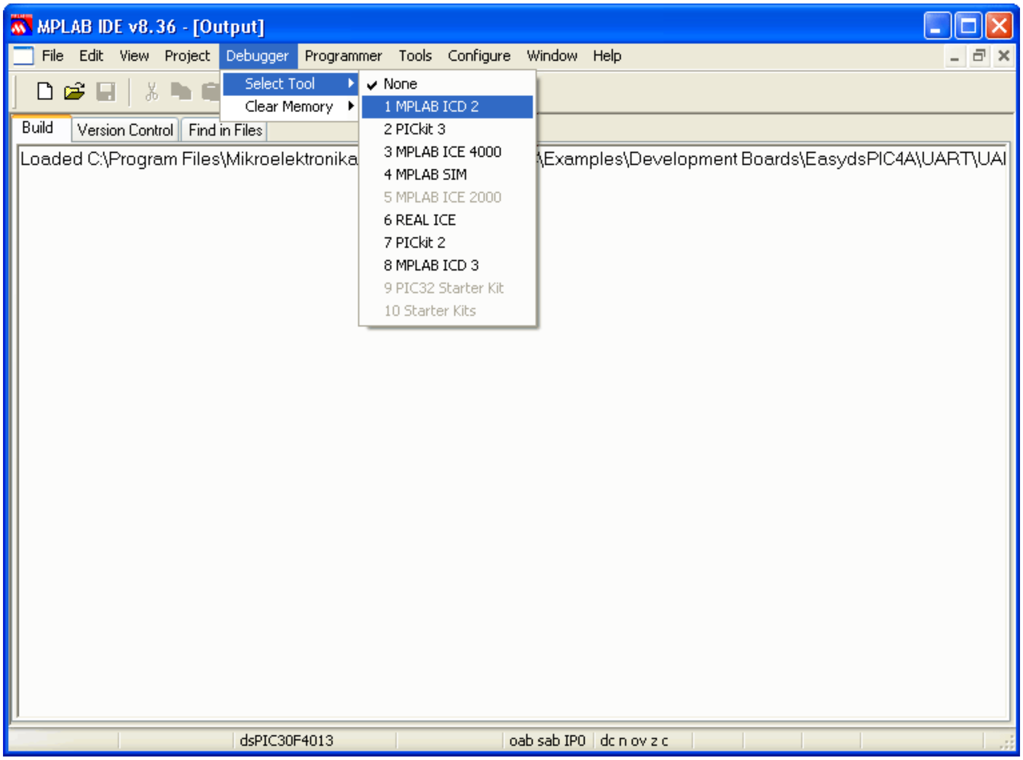
10. After device selection, click on the **File › Import**. Open file dialog box should appear. Then, go to the project folder and open the generated HEX file, `UART.hex`.

Note: This is very important, because `hex` file contains configuration bit settings which are essential for the proper functioning of the user code.

11. Next, click the **File › Import.** Open file dialog box should appear. Then, go to the project folder and open the generated COFF file, `UART.cof`:



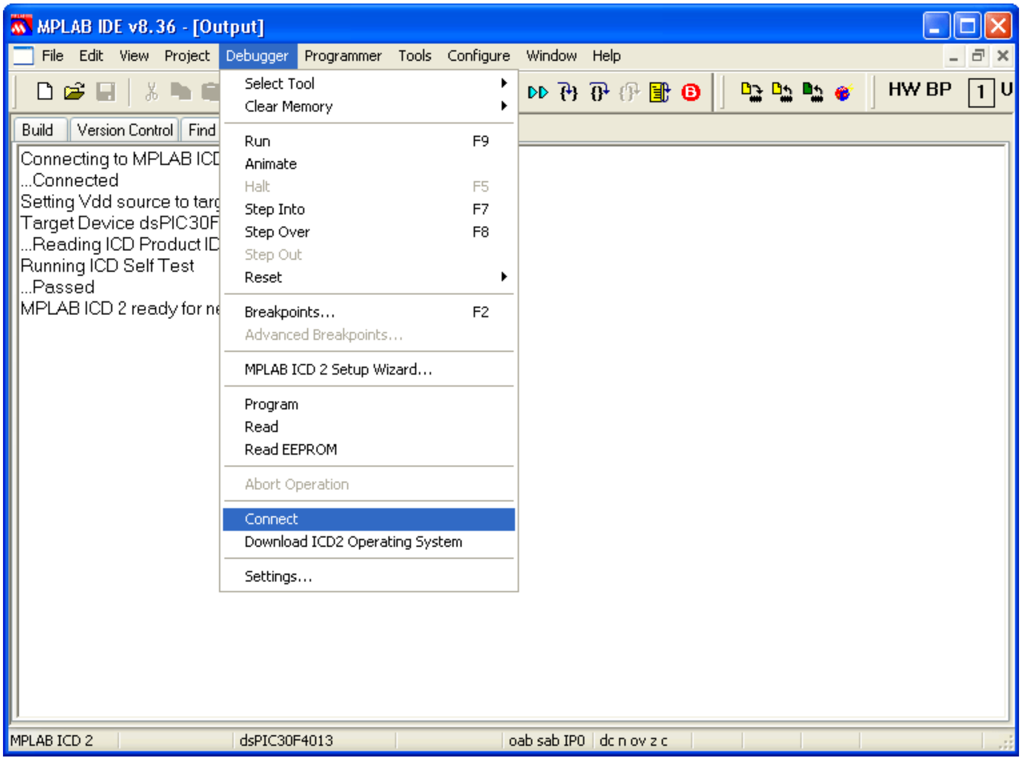
12. Then, select the **MPLAB® ICD 2** from the **Debugger › Select Tool** menu for hardware debugging:



```
Loaded C:\Program Files\Mikroelektronika \Examples\Development Boards\EasydsPIC4A\UART\UAI
```

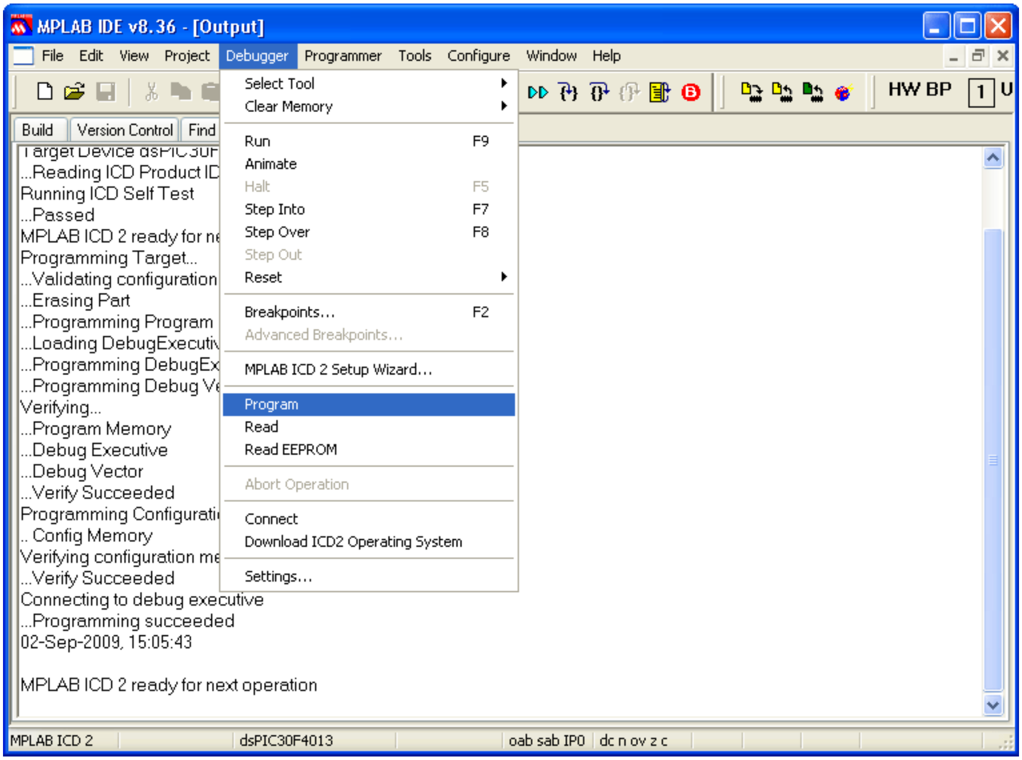
13. Complete the MPLAB® ICD 2 Setup Wizard from the **Debugger** menu (if needed).

14. After completing MPLAB® ICD 2 Setup Wizard, click on the **Debugger › Connect**:



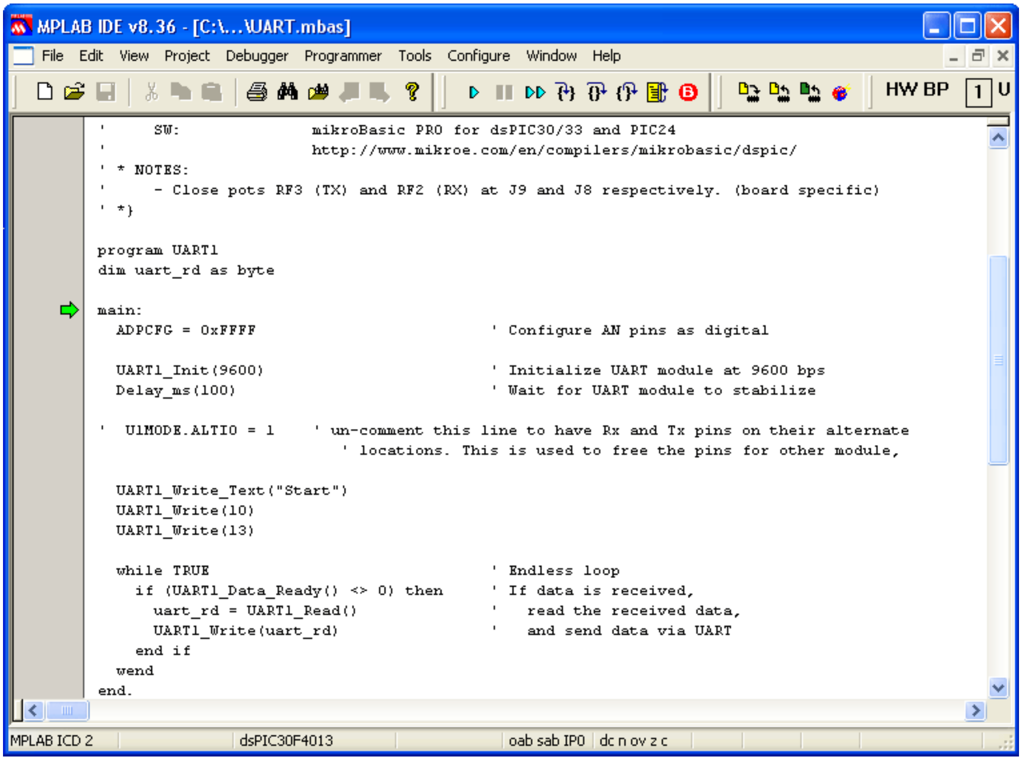
Connecting to MPLAB ICD
...Connected
Setting Vdd source to targ
Target Device dsPIC30F
...Reading ICD Product ID
Running ICD Self Test
...Passed
MPLAB ICD 2 ready for ne

15. Finally, click on the **Debugger › Program**:



```
Target Device dsPIC30F
...Reading ICD Product ID
Running ICD Self Test
...Passed
MPLAB ICD 2 ready for ne
Programming Target...
...Validating configuration
...Erasing Part
...Programming Program
...Loading DebugExecutiv
...Programming DebugEx
...Programming Debug Ve
Verifying...
...Program Memory
...Debug Executive
...Debug Vector
...Verify Succeeded
Programming Configurati
.. Config Memory
Verifying configuration me
...Verify Succeeded
Connecting to debug executive
...Programming succeeded
02-Sep-2009, 15:05:43

MPLAB ICD 2 ready for next operation
```

16. Now, you can start debugging the code by clicking Step Over button on the Debug toolbar, or by pressing F8:



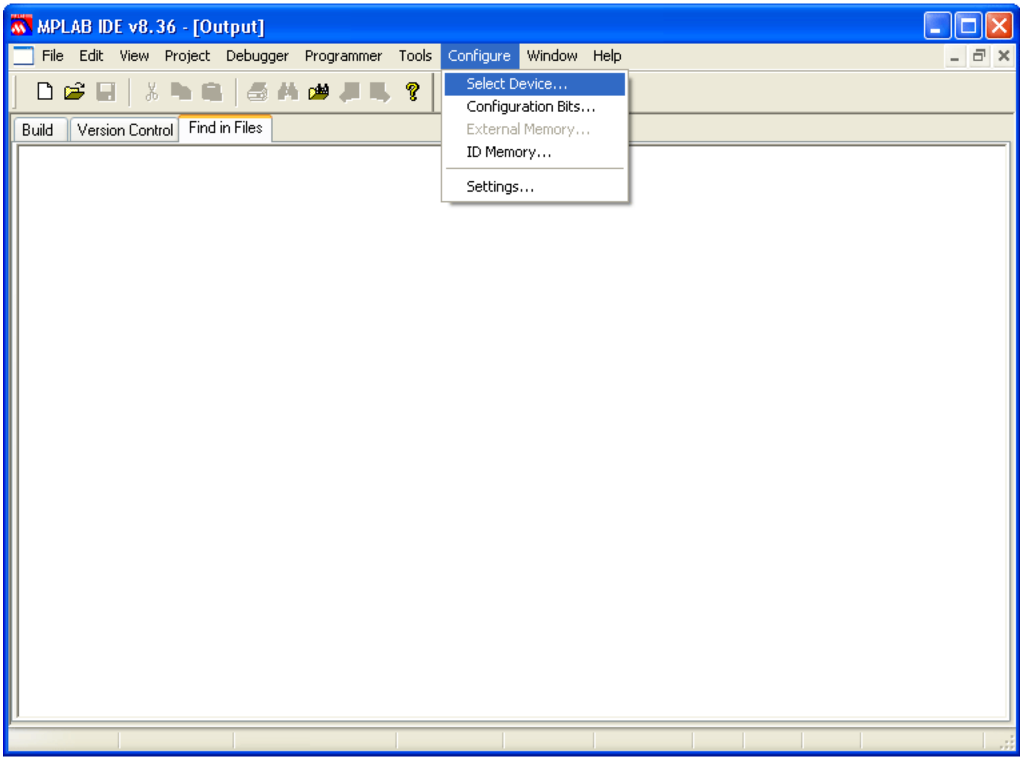
```
'     SW:                 mikroBasic PRO for dsPIC30/33 and PIC24
'                         http://www.mikroe.com/en/compilers/mikrobasic/dspic/
' * NOTES:
'     - Close pots RF3 (TX) and RF2 (RX) at J9 and J8 respectively. (board specific)
' *}

program UART1
dim uart_rd as byte

main:
  ADPCFG = 0xFFFF                          ' Configure AN pins as digital

  UART1_Init(9600)                         ' Initialize UART module at 9600 bps
  Delay_ms(100)                            ' Wait for UART module to stabilize

'  U1MODE.ALTIO = 1    ' un-comment this line to have Rx and Tx pins on their alternate
                          ' locations. This is used to free the pins for other module,

  UART1_Write_Text("Start")
  UART1_Write(10)
  UART1_Write(13)

  while TRUE                               ' Endless loop
    if (UART1_Data_Ready() <> 0) then      ' If data is received,
      uart_rd = UART1_Read()               '   read the received data,
      UART1_Write(uart_rd)                 '   and send data via UART
    end if
  wend
end.
```

Related topics: COFF File, Using MPLAB® Simulator

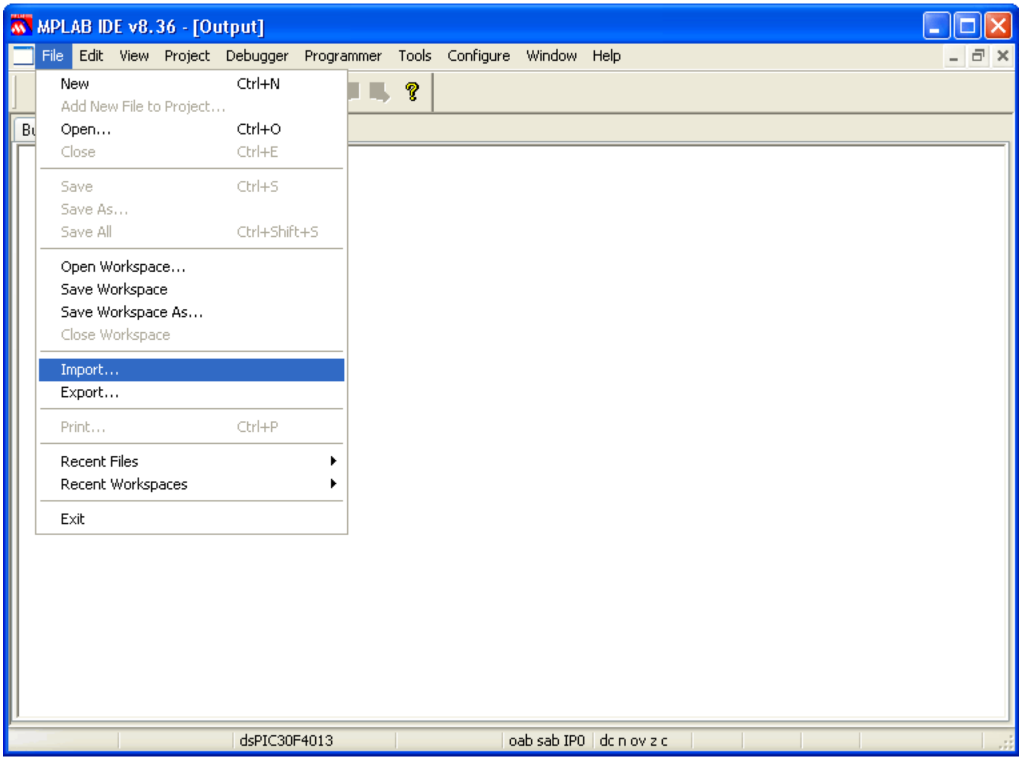
Using MPLAB® Simulator

Note: It is assumed that MPLAB® is previously installed.

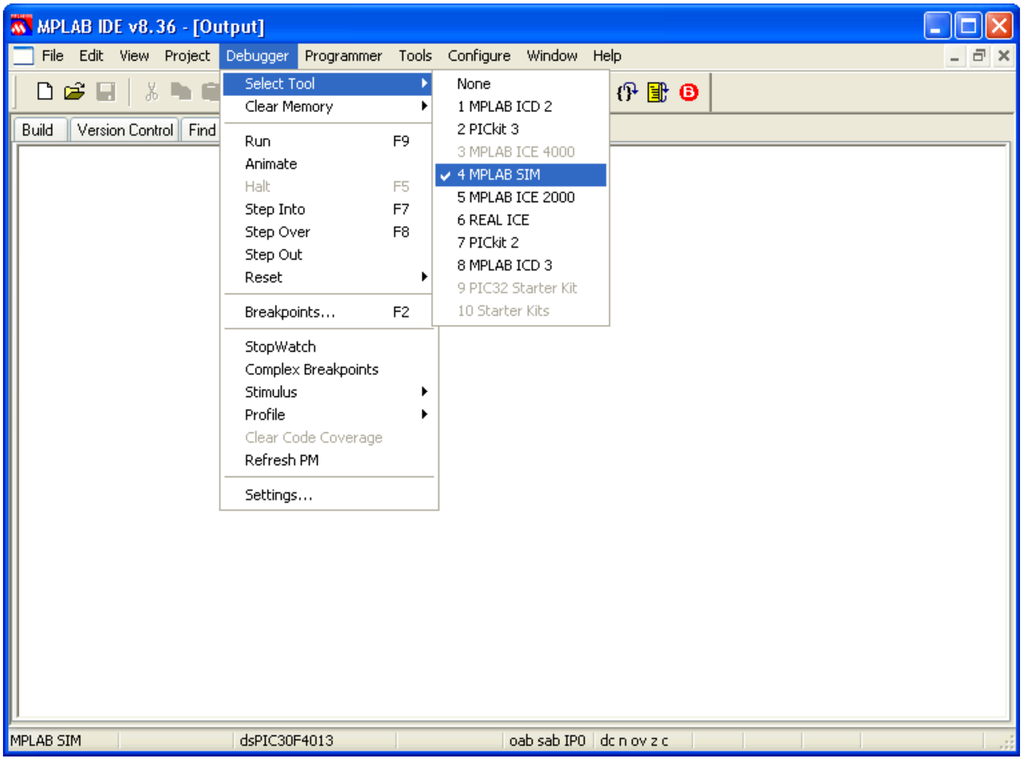
1. First of all, start mikroBasic PRO for dsPIC30/33 and PIC24 Help and open the desired project. In this example, UART project for EasydsPIC4A board and dsPIC30F4013 will be opened.
2. Open **Tools › Options › Output settings**, and check the “**Generate COFF file**” option, and click the OK button.
3. After that, compile the project by pressing Ctrl + F9.
4. Next, open MPLAB®, and select the appropriate device by choosing **Configure › Select Device...** :


MPLAB IDE v8.36 - [Output]
File
Edit
View
Project
Debugger
Programmer
Tools
Configure
Window
Help
Select Device...
Configuration Bits...
External Memory...
ID Memory...
Settings...
Build
Version Control
Find in Files

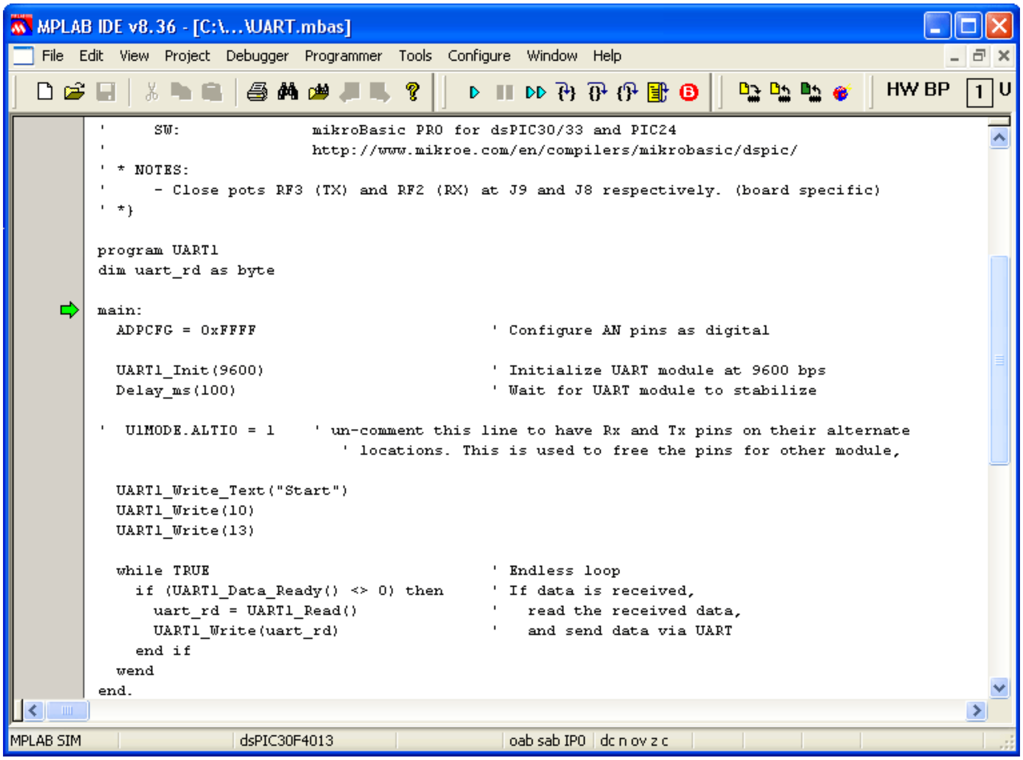
5. After device selection, click on the **File › Import**. Open file dialog box should appear. Then, go to the project folder and open the generated COFF file, `UART.cof` :



6. Then, select the **MPLAB® SIM** from the **Debugger › Select Tool** menu for software debugging:



7. Now, you can start debugging the code by clicking Step Over button {}+ on the Debug toolbar, or by pressing F8:



```
'     SW:              mikroBasic PRO for dsPIC30/33 and PIC24
'                      http://www.mikroe.com/en/compilers/mikrobasic/dspic/
' * NOTES:
'     - Close pots RF3 (TX) and RF2 (RX) at J9 and J8 respectively. (board specific)
' *}

program UART1
dim uart_rd as byte

main:
  ADPCFG = 0xFFFF                         ' Configure AN pins as digital

  UART1_Init(9600)                        ' Initialize UART module at 9600 bps
  Delay_ms(100)                           ' Wait for UART module to stabilize

'  U1MODE.ALTIO = 1    ' un-comment this line to have Rx and Tx pins on their alternate
                          ' locations. This is used to free the pins for other module,

  UART1_Write_Text("Start")
  UART1_Write(10)
  UART1_Write(13)

  while TRUE                              ' Endless loop
    if (UART1_Data_Ready() <> 0) then     ' If data is received,
      uart_rd = UART1_Read()              '   read the received data,
      UART1_Write(uart_rd)                '   and send data via UART
    end if
  wend
end.
```

Related topics: COFF File, Using MPLAB® ICD 2 Debugger

Frequently Asked Questions

This is a list of frequently asked questions about using mikroElektronika compilers.
If your question is not answered on this page, please contact mikroElektronika Support Desk.

Can I use your compilers and programmer on Windows Vista (Windows 7) ?

Our compilers and programmer software are developed to work on and tested on Windows 98, Windows 2000, Windows ME, Windows XP (32 and 64 bit), Windows Vista (32 and 64 bit) and Windows 7 (32 and 64 bit) and they work fine on these operating systems.
You can find the latest drivers on our website.

I am getting “Access is denied” error in Vista, how to solve this problem ?

Please turn off User Account Control (UAC). This should make your software fully functional. To do this, follow the path in your Windows Vista (logged in as administrator) **Control Panel › User Accounts › Turn User Account Control** on or off, uncheck Use User Account Control (UAC) and click OK.

What are differences between mikroC PRO, mikroPascal PRO and mikroBasic PRO compilers ? Why do they have different prices ?

Basically, there is little differences between these compilers. mikroC PRO is standardized with ANSI C, and it is much more complex and it is far more difficult to write the compiler for it. We used a lot more resources for making it than what we used for mikroPascal and mikroBasic. We also worked on some very complex topics such as floating point, typedef, union, a completely new debugger and many other. Because of that there is difference in price.

Why do your PIC compilers don’t support 12F508 and some similar chips ?

Unfortunately our PIC compilers don’t support 12F508 and similar chips because these chips are designed to use 12-bit wide instructions. Our compiler support MCUs which use 14-bit or wider instructions.

What are limitations of demo versions of mikroElektronika’s compilers ?

The only limitation of the free demo version is that it cannot generate hex output over 2K of program words. Although it may sound restrictive, this margin allows you to develop practical, working applications without ever thinking of demo limit. If you intend to develop really complex projects in one of our compilers, you should consider purchasing the license key.

Why do I still get demo limit error when I purchased and installed license key ?

If you are first time installing and registering compiler, you need to follow instructions exactly as described in registration procedure. License is valid only for the computer from which request is made, so license requested from one computer won’t work on another computer. You can find on our site manual and video describing in detail how to get your license.
If you previously had an older version of our compiler and have working license key for it but it doesn’t work with new compiler, you have to repeat registration procedure from the new compiler and you will get a new license.

I have bought license for the older version, do I have to pay license for the new version of the compiler ?

No, once you pay for the license key you get a lifetime license. When we release a new major release of the compiler, you might need to repeat registration procedure from your new compiler and you will get new license free of charge.

Do your compilers work on Windows Vista (Windows 7) ?

Yes!

What does this function/procedure/routine do ?

Please see your compiler’s Help where all of the functions are explained in detail.

I try to compile one of the provided examples and nothing happens, what is the problem?

You need to open project, not file. When you want to open an example, go to **Project › Open Project**, then browse through projects and choose project file. Now you will be able to compile and program with success.

Can I get your library sources ? I need to provide all sources with my project.

It is our company’s policy not to share our source code.

Can I use code I developed in your compilers in commercial purposes ? Are there some limitations ?

Regarding your code, there are no limitations. Your application is your own property and you can do whatever you like with it. If you want to include some of code we provide with our compilers or on our site, you may include them in your project, however, you are not allowed to charge your users for these.

Why does an example provided with your compilers doesn’t work ?

All of the examples provided with our compilers are tested and work fine. You need to read commented header of the example and be sure that you have used the same MCU example is written for and that you have hardware connections (DIP switches, jumpers etc.) set as described.

Your example works if I use the same MCU you did, but how to make it work for another MCU ?

You should read your MCU’s datasheet. Different MCUs can have different pin assignments and may require different settings. If you need help regarding this, you can find free online books on our website and recommend you starting there. You can also ask for help on our forum.

I need this project finished, can you help me ?

We currently do not do custom projects, however, we can give you some directions when you start working on your project and come to a problem. Also, our forum is very active community and as you can find there experts in different fields, we encourage you to look for help there.

Do you have some discount on your compilers/development systems for students/professors ?

Since large percentage of our customers are schools, laboratories and students, our prices are already scaled for these kinds of users. If you plan ordering more than one of our products, see special offers page on our website. Also, you can contact our Sales Department and see if you are eligible for some additional discount.

I have a question about your compilers which is not listed here. Where can I find an answer ?

Firstly, look for it in your compiler's Help. If you don't find an answer there, please create a support ticket on our website.

MikroElektronika

SOFTWARE AND HARDWARE SOLUTIONS FOR EMBEDDED WORLD ...making it simple

If you want to learn more about our products, please visit our website at www.mikroe.com

If you are experiencing some problems with any of our products or just need additional information, please place your ticket at www.mikroe.com/en/support

If you have any questions, comments or business proposals, do not hesitate to contact us at office@mikroe.com