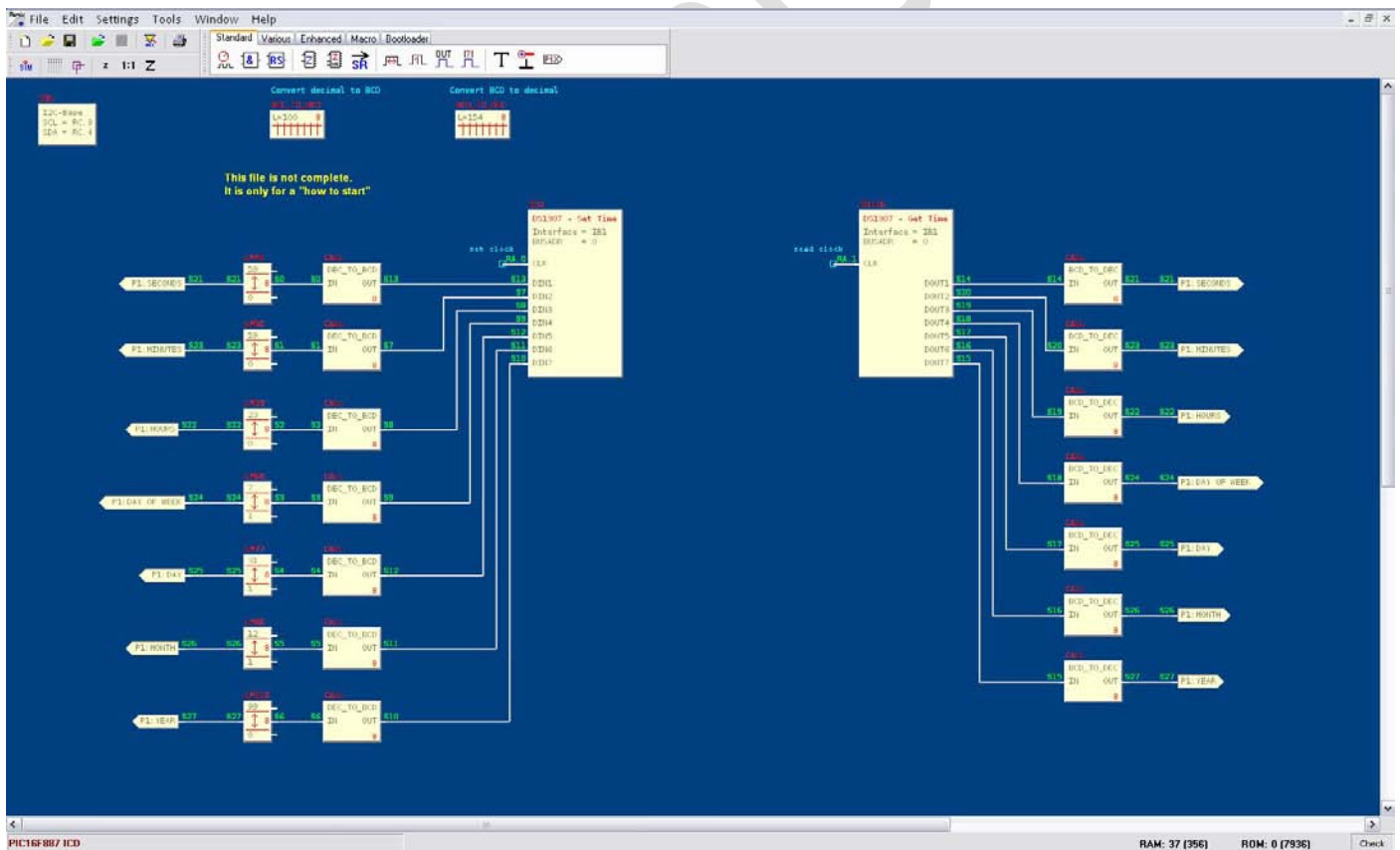




Visual Parsic V4

User manual

Version : 4.0.6.21 October 2013-10-10



Visual Parsic V4 user manual

Notes

1 . License agreement

Please read this software license agreement carefully before using the software. By clicking on accept button, opening the package, downloading the product, or using the equipment that contains this product, you are consenting to be bound by this agreement. If you do not agree to all of the terms of this agreement, click the “ do not accept” button and the installation process will not continue. Return the product to the place of purchase for a full refund.

All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice.

2. Single user license grant , Intellectual property rights - limited warranty

Parsic Italia snc grants to Customer ("Customer") a nonexclusive and nontransferable license to use the Visual Parsic V4 software ("Software") in object code form solely on a single central processing unit owned or leased by Customer or otherwise embedded in equipment provided by Parsic Italia snc.

Parsic Italia snc, worldwide distributor, does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Visual Parsic V4 . No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Parsic Italia snc or Swen Gosch. Visual Parsic V4 is a Swen Gosch product (Elmschorn Germany) . Swen Gosch (Elmschorn Germany) claims the copyright of, and retains the rights to, all material (software, documents, etc.) contained in this release. The Customer shall not copy, in whole or in part, software or documentation. Modify the software, reverse compile or reverse assemble all or any portion of the software, or rent, lease, distribute, sell, Visual Parsic V4 . Parsic Italia snc, warrants that for a period of ninety (90) days from the data of shipment of Visual Parsic V4, the media on which the software is furnished will be free of defects in material and workmanship under normal use, and the software substantially conforms to its published specifications. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of the products and application examples. The Customer are fully responsible for the incorporation of the HEX software, produced by Visual Parsic V4, in its equipment. Parsic Italia snc assumes no responsibility for any losses incurred by you or third parties arising from the use of software Visual Parsic V4.

Parsic Italia snc has used reasonable care in preparing the information included in this document, but Parsic Italia does not warrant that such information is error free. Parsic Italia assumes no liability whatsoever for any damages incurred by you resulting from errors or omissions from the information included herein. The customer assumes its responsibilities arising from the production of electrical automation, produced by software Visual Parsic V4. In no event and any reason, Parsic Italia snc, does warrant the usability or suitability of its software Visual Parsic V4 in any medical, aviation, military or other life critical applications.

Parsic V4 is intend for programming PIC® microcontrollers from Microchip® 8-bit family series 10/12/16/18F

3. Upgrade and support

Parsic Italia provide upgrades, from our web site www.parsicitalia.it or www.parsicitalia.com.

For technical support send Email at info@parsicitalia.com

4. Copyright

Microchip, Windows,PICkit2-3,MPLAB,MCP2200,Windows,Visual Parsic V4,and others are internationally registered trademark.

5. Requirements

Win 98, Windows XP, Vista, W7, W8, MPLAB and his accessories PicKit 2-3*, MCP2200* for USB connectivity by Arethusa Card Hardware system Arethusa Series V21xx*

6. Visual Parsic on PC

Automatic setup . A license of Parsic V4 is sold with an activation USB key

(* Optional component)



Win 98



Win2000 32 bit



Win XP 32-64 bit



Vista 32-64 bit



Win 7 32-64 bit



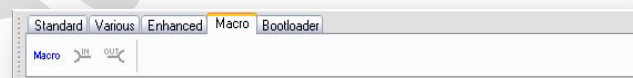
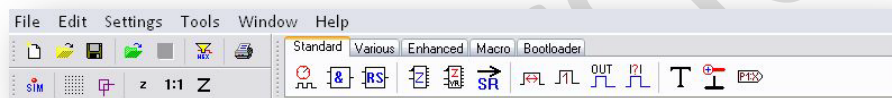
Win 8 64 bit

1.1 Visual Parsic V4 description

Parsic V4 was developed with ease of use in mind, which means that it does not require long manuals, extended tutorials or deep knowledge to use. Parsic V4 is a graphical programming tool for PICmicro devices. It features an intuitive and clean interface and enables the user to quickly and easily design, deploy and debug a program that gets transferred and executed by the PICmicro device. It boasts a rich set of features - I/O interfaces, timers, counters, configurable clock, algebra, memory, logic and non-linear operations, multiplexer and de-multiplexer, sleep function, absolute value comparator, LCD block display, UART, etc. Parsic V4 also features an efficient debug that enables the user correction of errors in real time

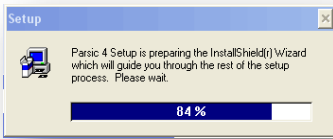
1.2 Main features

- Simple and intuitive graphical user interface with integrated support. Drag and Drop the function blocks from toolbar with mouse and connect them by clicking on the input/output ports. With just one click you can access the help function.
- Combinational logic block: AND / NOT / OR / XOR / EXOR
- Timing function and trigger : clock generator, counter 8bit, counter up/down 8-16bit, signal level trigger, pulse-timer, one-shot, mono-flop, PWM generator, Schmitt trigger
- Memory : S/R FF, limiter, memory block EE
- Mathematical functions: basic algebra functions for create simple conditional logics, absolute value comparators
- Special functions : conversion tables, digital limiters, multiplexer, de-multiplexer, sleep, dat, include, macro, bootloader
- Communication : UART, I2C, SPI, LCD
- Editor : I2C, SPI, object order
- Automatic functions : assembler, real time debugging, process control and simulation, automatic download in to device.

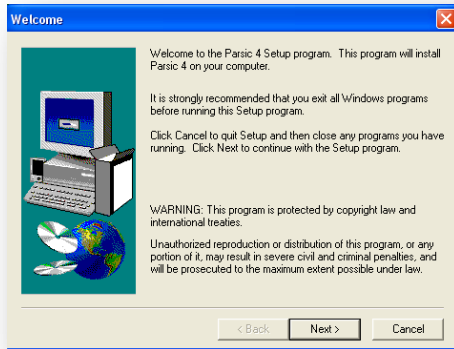


1.3 Installation

Visual Parsic V4 setup executable is located on Visual Parsic V4 CD. When you insert CD in your PC please launch setup.

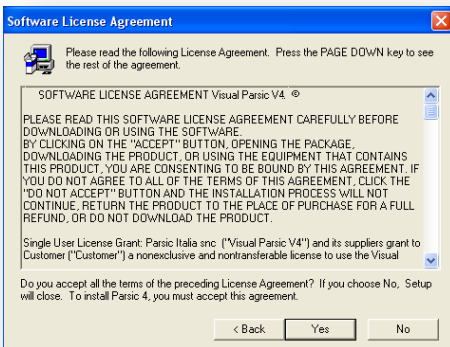


Step 1- Start installation



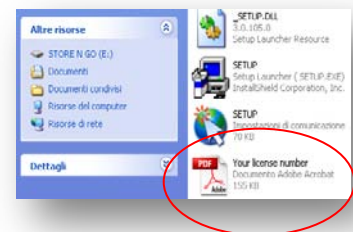
Welcome screen. Click Next button to proceed.

Step 2 – Licence Agreement



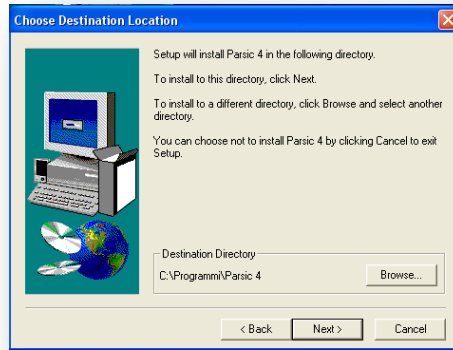
Carefully read end user license agreement. If you agree with it, click Next to proceed

Step 3 – Readme information and Select user



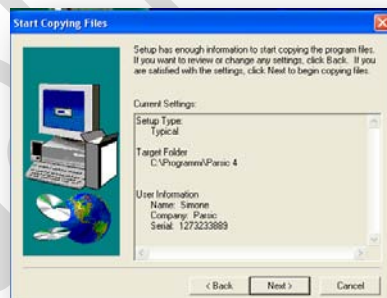
It's recommended to insert your Name, Company and serial number. In PDF setup page there is the number of your license. Click Next to proceed.

● **Step 4 - Choose destination**



We suggest destination folder, or select a different installation by clicking browse button

● **Step 5 - Setup Type, Select program Folder, Progress installation bar**



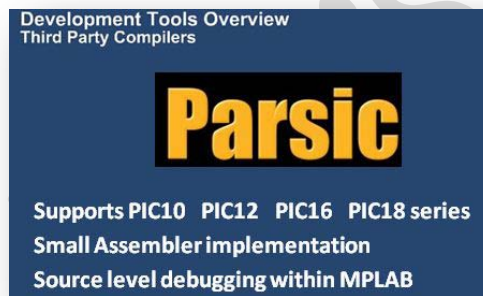
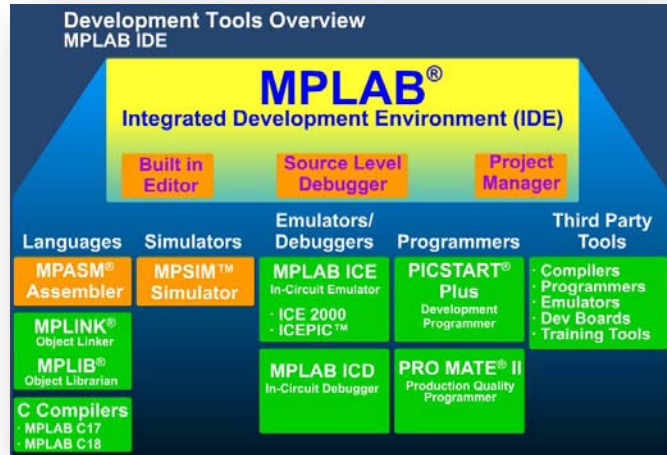
Click finish button to close Setup

After installation process, shortcut will appear on your desktop. **Double click for start Parsic 4.**

1.4 Instruments programming

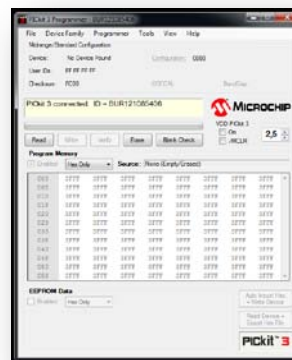
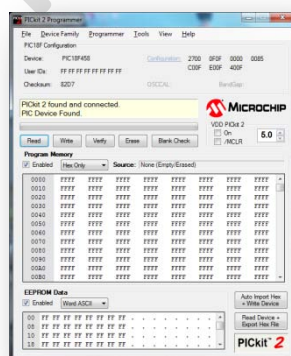
MPLAB IDE V8

For correct use, Parsic 4 need **MPLAB IDE V8** . On your PC please, go to C ::\Program \Parsic 4\ Microchip and locate the folder contain setup executable MPLAB. Click on Setup and install MPLAB on your PC. The MPLAB Integrated Development Environment (IDE) is a free, integrated toolset for the development of embedded applications employing Microchip's PIC® microcontrollers. MPLAB IDE also serves as a single, unified graphical user interface for additional Microchip and third party software and hardware development tools.



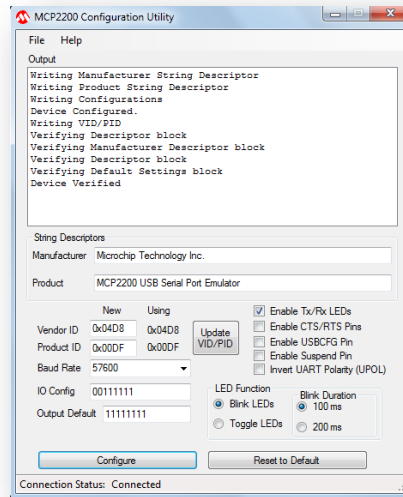
PICKit Development Programmer/Debugger

The most popular PIC programmer made by Microchip is "PICKit 2". It is a low cost development tool with an easy to use interface for programming and debugging Microchip's Flash families of microcontrollers. You can also use it as a stand-alone instrument for programming. The advanced programmer "PICKit3" In-Circuit Debugger/Programmer, as the PICKit2, uses in-circuit debugging logic incorporated into each chip with Flash memory to provide a low-cost hardware debugger and programmer. The PICKit2 and the PICKit3 share many common features, however, PICKit3 has some additional features. For comparison chart read the document from Microchip " Debugging and Emulation Product Line ". **PICKit2 and PICKit3 program are inside MPLAB IDE.** If you use another model programmer, do not use this software.

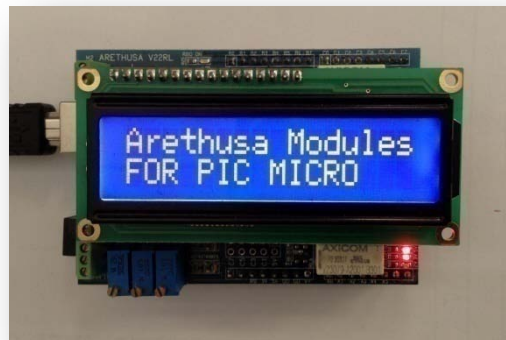
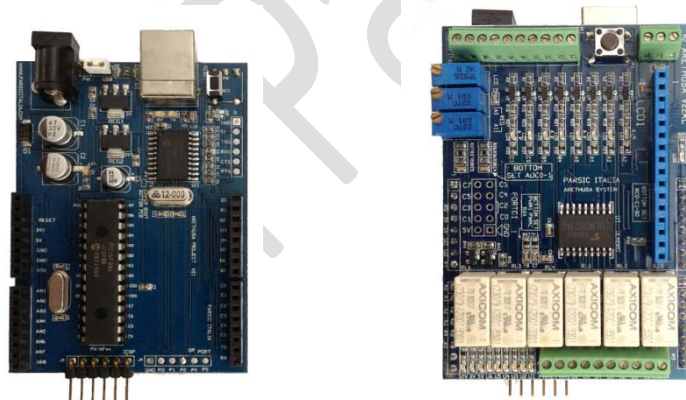


MCP2200 interface (optional installer)

The **MCP2200** is a USB-to UART serial converter which enables USB connectivity to applications that have an UART interface. It is used in the Arethusa prototyping platform. Arethusa is a single board microcontroller, ideal for building easy electronic projects, suitable for learning Parsic V4. The hardware consists of a board designed around a PIC16F886 8-bit microcontroller. The MCP2200 converter, allows you to connect thr Arethusa board to a PC , by using USB. It is allowing you to configure and programming the PICmicro by the Bootloader of Parsic V4. Install the MCP200 driver if it is necessary. On your PC go to C:\Program\Parsic4\Microchip and locate the folder which contains the executable MCP2200 Windows Driver & Installer.



Now, your installation is complete. Double click on icon Parsic 4 for start the program.



Arethusa V21 and V22RL card

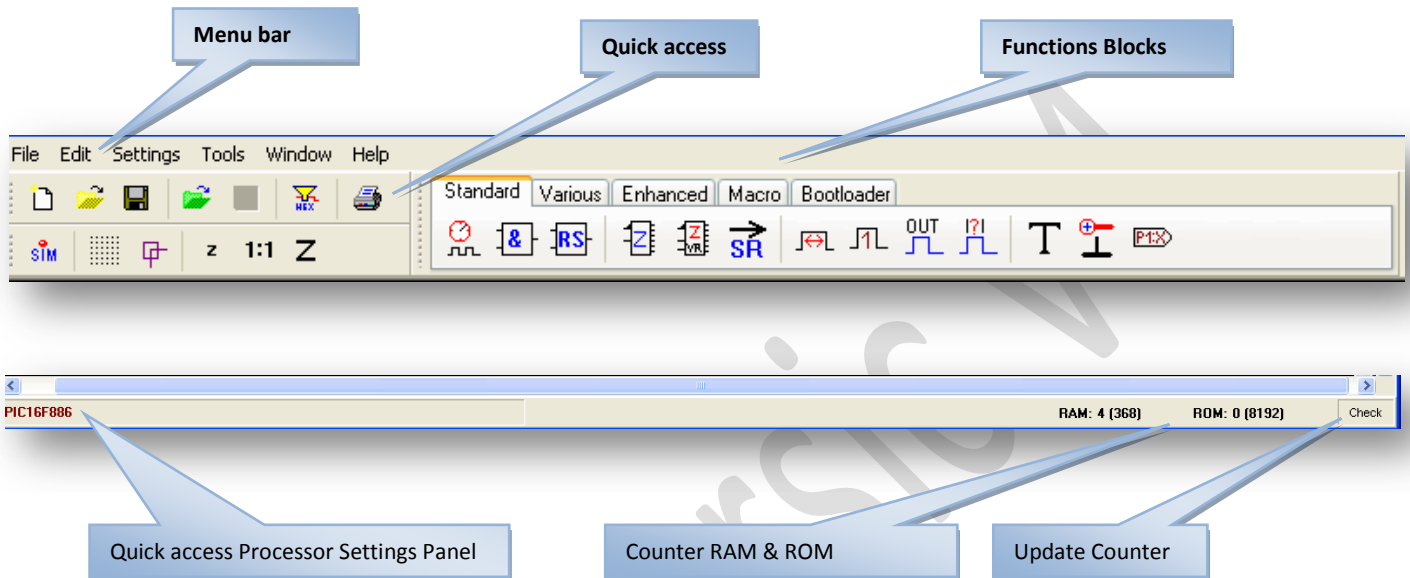
1.5 Parsic 4 user interface map

Description

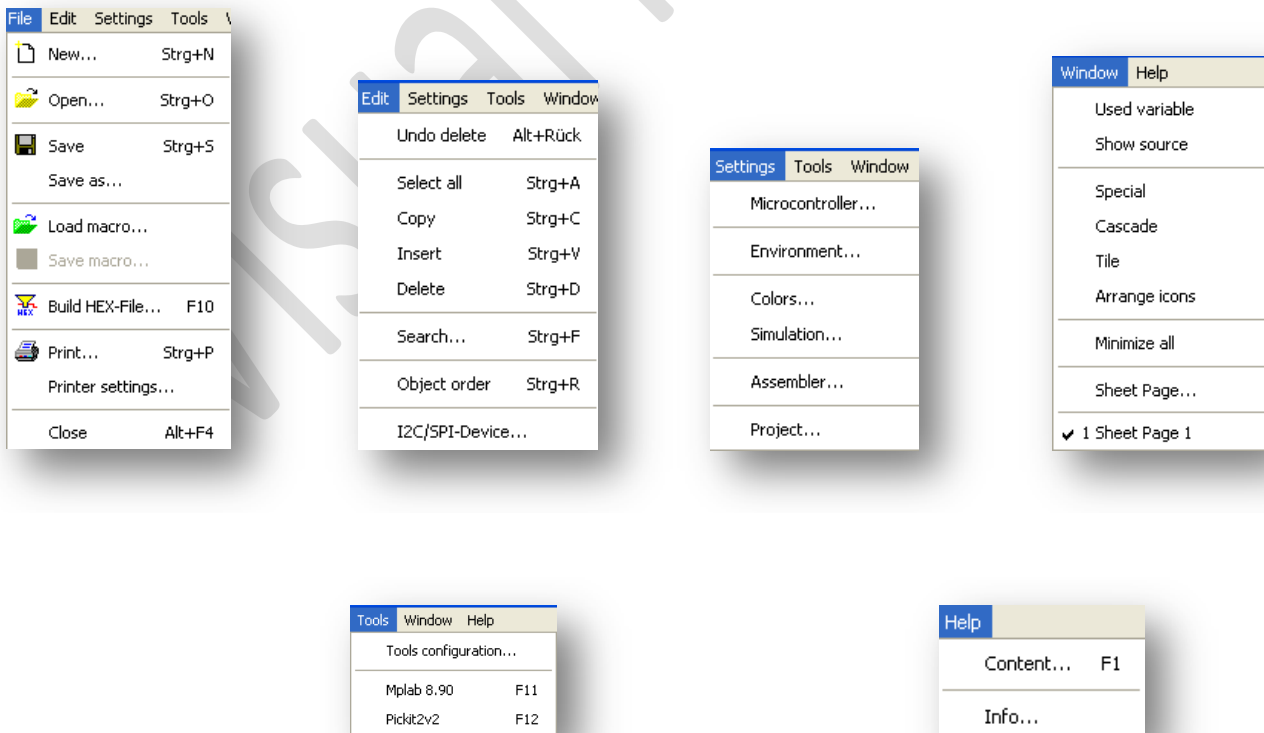
Parsic V4 user interface is displayed below. The upper area contains the menu bar : File, Edit, Setting, Tools, Windows, Help.

In the lower left area, there are Icons for quick access to the most frequently used functions: New file, Open project folder, Save, Load Macro file, Save Macro, Build Hex File, Print, Simulation function, Zoom and Grid.

In the right user interface are displayed the Functions Blocks of Parsic 4. This blocks are divided in four categories : Standard, Various, Enhanced, Macro. Boot loader is the serial programmer.



In the **Application Bar**, there is the icon for direct access to **Processor Settings Panel**, the counter of the occupied area **RAM** and **ROM** and the push button for **Updating the ROM Status**. The following figures shows the complete menu of Parsic V4



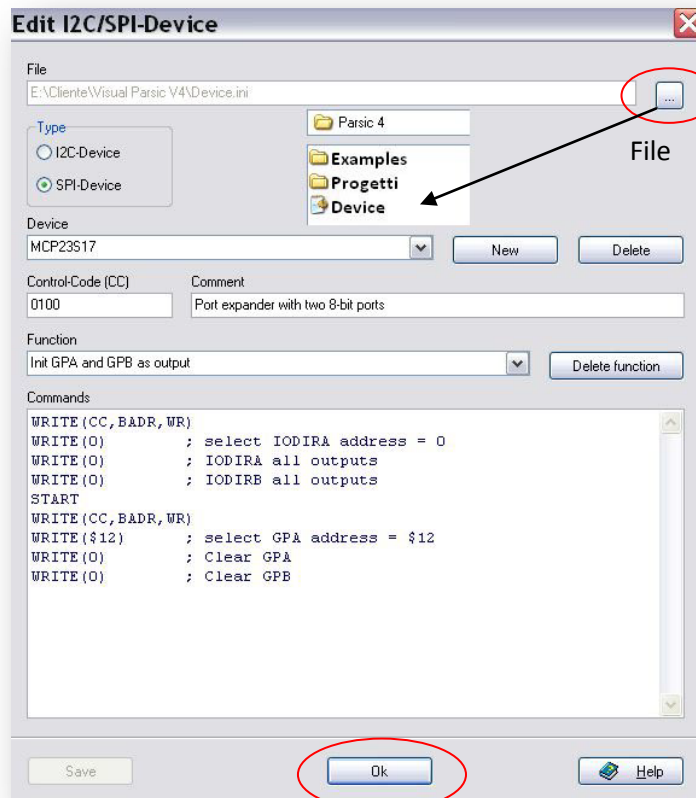
Start Parsic 4

Configuration Parsic 4

2.0 To program a PICmicro with a executable file. HEX, apply the following instructions :

● Step 1 Edit I2C –SPI menu

Open the **EDIT** menu and click the **Edit I2C/SPI-Device**. From button File.... locate, in **Parsic 4** folder, the file **Device.ini** Double click on it and next confirm OK.



● Step 2 AUX program MPASMWIN

Open the **Settings** menu and click **Assembler**. In **External Programs**, click the **Search** button to find **MPASMWIN.exe** This program is located in **C:\Program\Microchip\MPASM Suite**. Double click on icon **MPASMWIN.exe** and next click push button OK.



● Step 3 Environment

Open the **Settings** menu and click **Environment**. In this panel there are some settings for create the logical schematic. Its contents will be adjusted according your needs :

Create wire-names automatically

Activated, the wire-names are created automatically, when connecting objects. (default)

- With Bit-Signals > Sx.x (for example S0.1 is the first connection of logic AND 1)
- With Byte-Signals > Sx (for example S0 is the out signal of a arithmetic block)

● **Arrange Windows automatically**

If enabled, while opening or closing a window, the size of the remaining window is automatically adjusted.

● **Arrows at slanted connections**

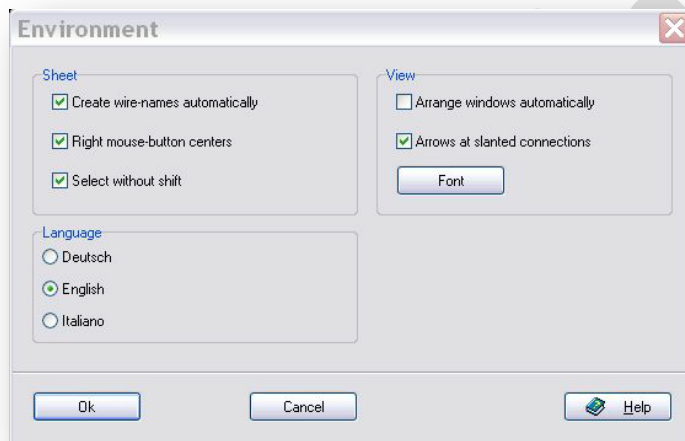
If enabled, the start and the end of a connection are marked whit arrows, when they are not at right angles

● **Select without shift**

If enabled, objects can also be selected without pressing the shift-key

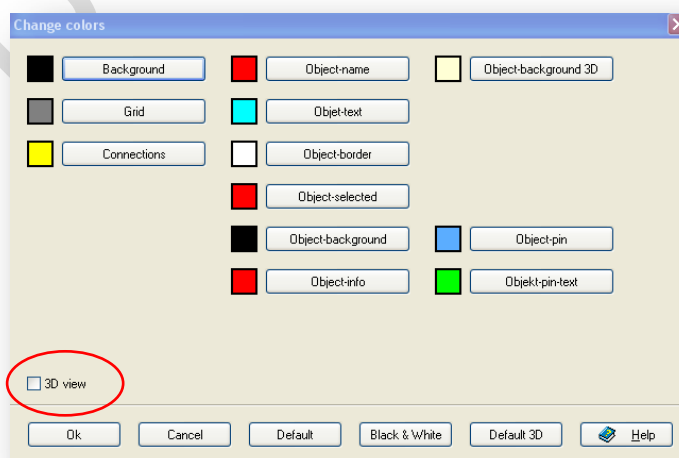
● **Language**

Choose your language



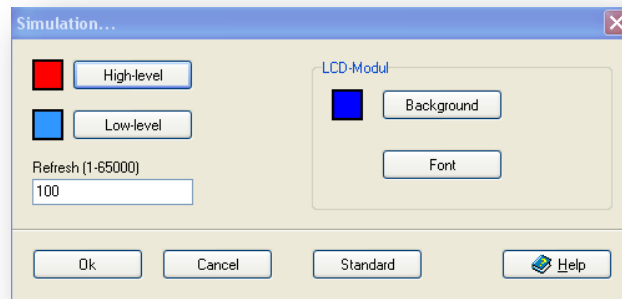
● Step 4 Color and Simulation

Open the **Settings** menu and click **Color**. On this control panel it is possible to change the colors mode of the objects, connections and background. Select 3D for to view the objects in **3-Dimensions** .



Simulation

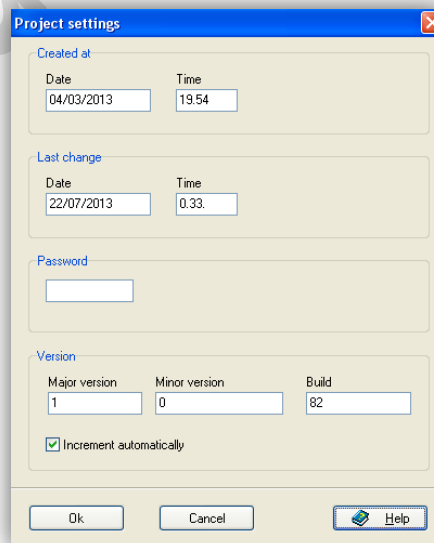
If you click **Simulation**, in Settings menu, you can change the color of the logic level line simulation (High – Low), and the background of LCD module. The field **Refresh**, determines the interval in milliseconds, the objects are updated in the simulation. The value must be between 1 and 65000 ms.



● Step 5 Project Setting

In Settings menu, click on **Project setting**. The following properties are available :

- **Created on**
Creating-date and time of the current file. It cannot be changed.
- **Last change**
Last saving date and time of the file. It cannot be changed.
- **Password**
If a password is used, it is requested loading this file. All characters are allowed.
- **Version**
- **Main version**
An arbitrarily number between 0 and 127 can stay here.
- **Minor version**
An arbitrarily number between 0 and 127 can stay here, also.
- **Build**
Arbitrarily number between 0 and 255.
- **Increase automatically**
If activated, the build version will be increased by 1, during saving the file. On overflowing from 255 to 256 the build version will be reseted to 0 and the minor version will increased by 1. The main version remains always unchanged. The main version and the side version typed in the assembler directive "**__IDLOCS**". Therefore it is possible to comprehend which program version is "branded" in the PIC.

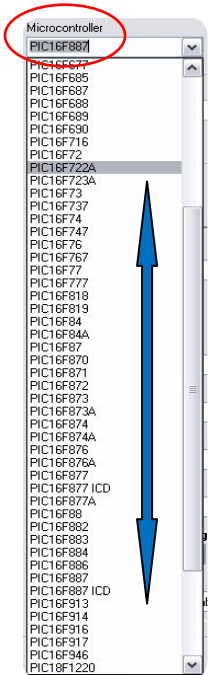


2.1 Config Directive Configuration Word Bits PICmicro MID-RANGE MCU Family

The **CONFIG** directive enables the configuration word to be present at assembly time rather than at programming time. When the code is assembled the configuration word is included in the hex file. This reduces the chances that devices will be inadvertently programmed with the incorrect configuration settings. These configuration bits specify some of the modes of the device, and are programmed by a device programmer, or by using the In-Circuit Serial Programming (ICSP) feature of the midrange devices. For additional information, please refer to the Programming Specification of the Device.

Step 1 Settings menu. Microcontroller

Open the **Settings** menu and click **Microcontroller**. Click it to expand panel containing MCU configuration option. Its contents will be adjusted depending on the selected PICmicro. From drop down list Microcontroller, **choose MCU** you want to use. You can select one MCU 8-bit family from a list of 74 PICmicro **10/12/16/18F series** (other MCU will be added in the next upgrade).



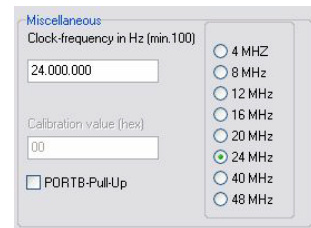
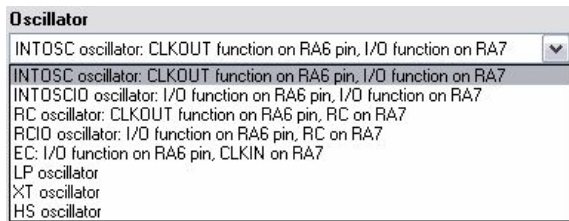
Step 2 Oscillator configuration

The oscillator configuration is set during programming. The oscillator mode is selected by the device configuration bits. Midrange PICmicro devices can have up to eight oscillator modes, this allows a single device type the flexibility to fit applications with different oscillator requirements. Each mode provides varying amounts of oscillator gain for various oscillator designs. Listed here are the currently available oscillator configurations. See your specific device datasheet for the ones that apply.

Select the oscillator : is possible choose 5 diverse type oscillator

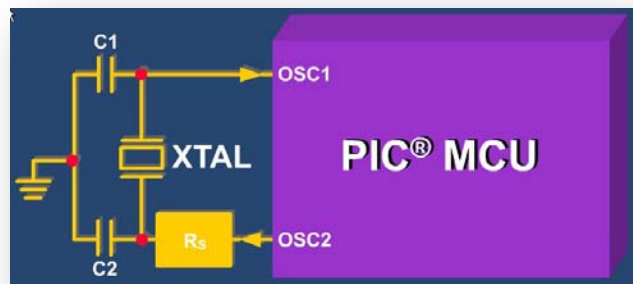
- **RC(ext)** . The clock frequency is determined by a RC combination
- **RC(int)**. Internal RC-oscillator, not available at all PICmicro, is the internal 4MHz Resistor/Capacitor clock
- **LP. Low-Power**, for quartzes or ceramic-swinger of 20KHz to 200KHz, and usually used with 32.768KHz
- **XT**. For quartzes or ceramic-swinger of 200 KHz to 4 MHz – 3MHz resonator
- **HS**. High speed, for quartzes or ceramic-swinger 4-20 MHz, and resonator over 3MHz

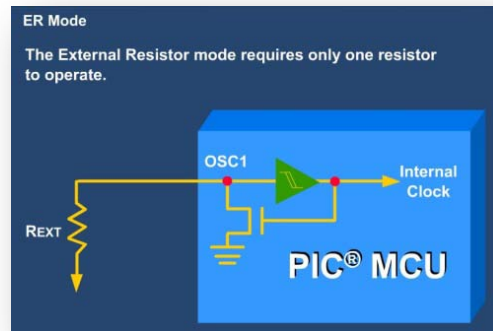
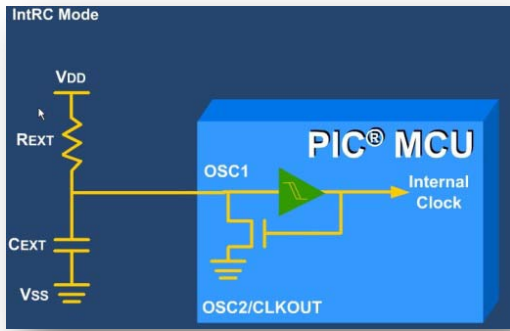
Note that additional oscillator configurations exist, so it is important to check your device datasheet for the ones that apply to your device.



Clock frequency table

Enter the value in Hz, with which the oscillator runs or select the correspondent frequency .





IntRC Mode and ER Mode should not be used for timing sensitive applications such as RS-232 communication



Step 3 Fail-Safe clock monitor

Is possible activate this function, if the device contains the fail-safe register. It monitors the external (primary) oscillator and will automatically switch over to the internal (secondary) oscillator if the primary oscillator fails.

Step 4 Internal External switchover (IESO)

Two-speed startup is a useful feature on some nanoWatt and all nanoWatt XLP devices which helps reduce power consumption by allowing the device to wake up and return to sleep faster. Using the internal oscillator, the user can execute code while waiting for the Oscillator Start-up (OST) timer to expire (LP, XT or HS modes). This feature (called "Two-Speed Startup") is enabled using the IESO configuration bit. A Two-Speed Start-up will clock the device from an internal RC oscillator until the OST has expired. Switching to a faster internal oscillator frequency during start-up is also possible using the OSCCON register.

Step 5 Watch dog timer – Watch dog prescaler

Activate, Parsic automatically insert commands, which reset the **WDT** regularly. If the program gets stuck a reset occurred after approx. 18ms. The Watchdog Timer (WDT) is a free running on-chip RC oscillator which does not require any external components. This RC oscillator is separate from the device RC oscillator of the OSC1/CLKIN pin. This means that the WDT will run, even if the clock on the OSC1 and OSC2 pins has been stopped, for example, by execution of a SLEEP instruction. The Watchdog Timer (WDT) is enabled/disabled by a device configuration bit. If the WDT is enabled, software execution may not disable this function. The WDT has a nominal time-out period of 18 ms, (with no postscaler). The time-out period varies with temperature, VDD and process variations from part to part (see DC specs). If longer time-outs are desired, a postscaler with a division ratio of up to 1:128 can be assigned to the WDT, under software control, by writing to the OPTION_REG register. Thus, time-out periods of up to 2.3 seconds can be realized. WDT period can be expanded up to 2.3 seconds by using max. prescaler value.



Watchdog Timer

Watchdog Timer Enable Bit (WDTEN)

CP	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	PWRWREN	WDTEN	FOSC1	FOSCO
----	-------	------	------	-----	-----	-------	---------	--------------	-------	-------

Enables the Watchdog Timer (WDT) at run time

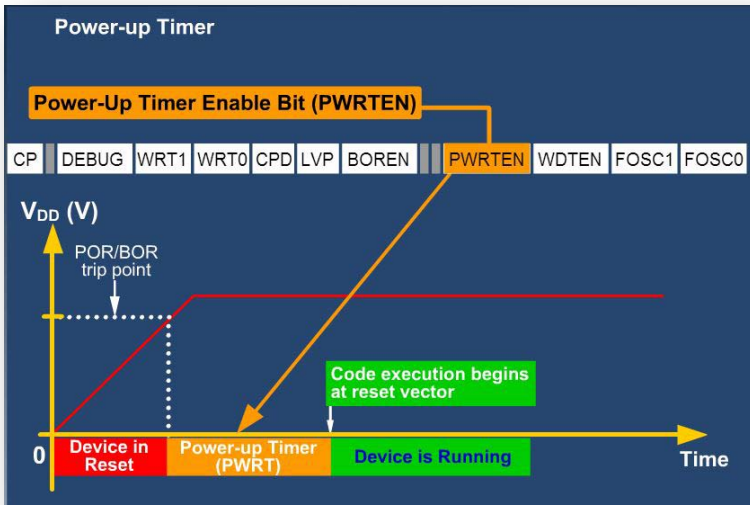
Watchdog Timer is typically used for one of two functions:

- Forces reset if code execution becomes unstable
- Forces MCU to wake from SLEEP mode on periodic basis

Postscaler can be used to increase delay time

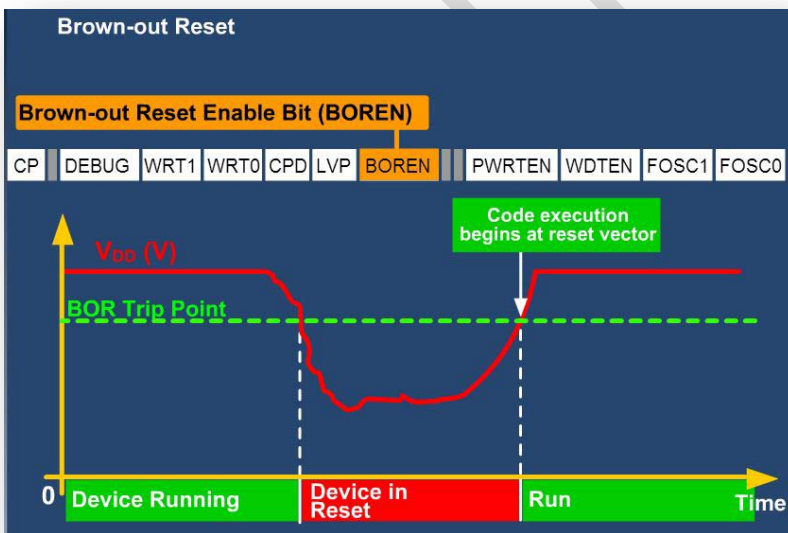
● **Step 6 Power-On timer.**

The Power-Up Timer or PWRT provides a time-out period to allow the VDD to rise to an acceptable level to prevent ambiguous operation. It operates on an internal RC oscillator that provides a nominal 72ms delay. When Brown-Out Reset is used the Power-Up Timer is always enabled. Power-Up Timer is always enabled when Brown-Out Reset is enabled, even if Power-Up-Timer is disabled. They work together to allow VDD to rise to an acceptable level after a Brown-Out Reset . Please see your device specifications for Power-Up Timer . Activated, additional 72ms maintained after a reset, till the microcontroller starts with the program.



● **Step 7 Brown-Out-Detect. (Brown Out Reset – Brown-Out-Voltage)**

Another feature enabled through the configuration bits is the Brown-Out Reset or BOR. The function of Brown-Out Reset is to reset the CPU when VDD drops below VBOR, this is typically at 4 volts. The BOR will hold the device in reset for as long as VDD remains below VBOR. When VDD rises above VBOR, the device is held in reset for additional time by the Power-up Timer (PWRT). If VDD should drop below VBOR at anytime, the BOR will then restart the process again. Disadvantage : additional electricity consumption of max 500 μ A.

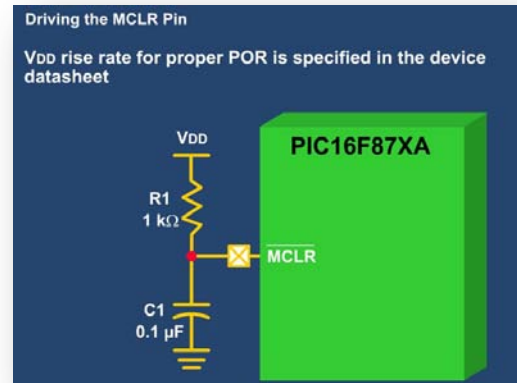
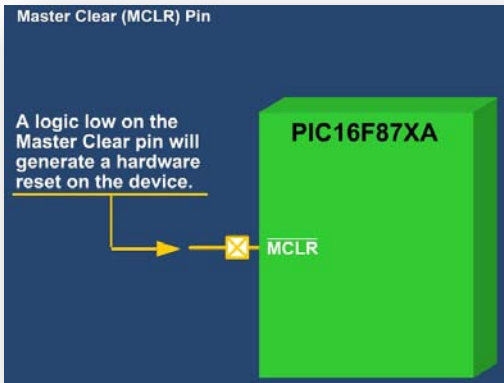


● **Step 8 Port-B pull up enable.**

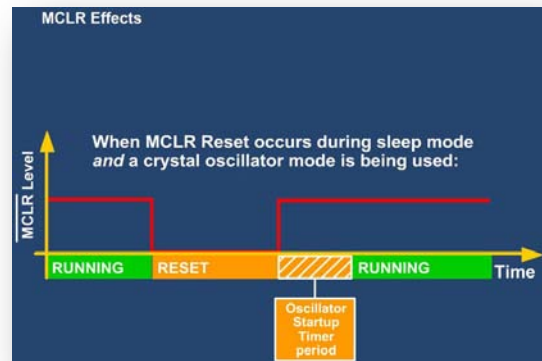
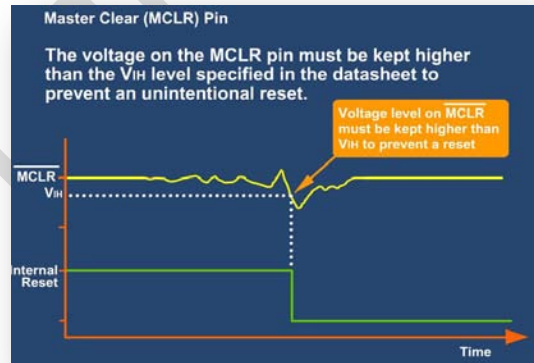
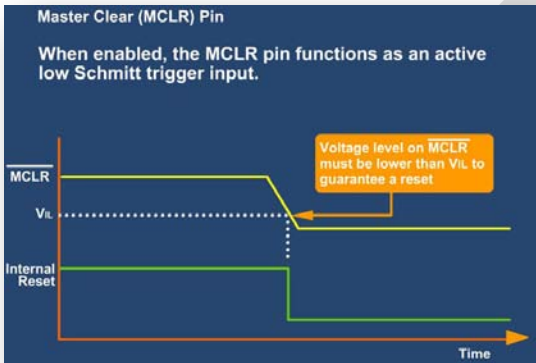
Activated, each PIN from RB=... RB7 internally get pull-ups, when configured as input. All the port pins have built in *pull-up* resistor, which make them ideal for connection to push-buttons, switches and optocouplers.

Step 9 MCLR Master Clear Enable

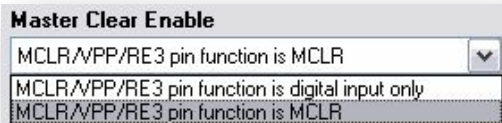
Master Clear (MCLR) pin, is used to generate a hardware reset on the device. On most PICmicro MCU's, the MCLR pin is always active. However, on some MCUs, the MCLR signal can be configured to be internally tied to VDD, which allows the pin to be used as a general purpose input pin.



It is very important to note that the MCLR pin has no internal pull-up and must NOT be left floating. It should also be noted that on some devices the MCLR is used to place the device in programming mode with a VPP voltage on the pin, which is usually 13V. Because of this, there is no upper protection diode on this pin, although there is a voltage clamp. When the MCLR function of the pin is enabled, it is an active low Schmitt trigger input. This means that to guarantee a reset, the voltage on the pin must be lower than the V_{IL} (maximum input low voltage) level specified in the datasheet. To make sure that the device does not get reset unintentionally during normal operation, the voltage on the MCLR pin must be kept higher than V_{IH} (minimum input high voltage) level specified in the datasheet. The noise in the system can cause the accidental reset of PICmicro.



When MCLR is brought low, the device will be held in a reset state until MCLR is brought high. During this time the oscillator will run and the I/O pins will be held in the reset condition (normally tri-state). If the device was reset from sleep and a crystal oscillator mode is used, then the oscillator startup timer will keep the device in the reset condition to give the oscillator time to start up, even if the MCLR signal is only held low for a short period. When an RC oscillator is used, the oscillator is running during reset, but the output on the OSC2 pin will stop. In the following table, there are some examples of the MCLR settings :



- MCLR = MCLR normal reset- input
- MCLR = Input, MCLR int. tied to VDD. The MCLR-Pin is input (e.g. GP3). MCLR is internally tied to Vcc (+)
- MCLR = MCLR pin enabled ; RG5 input pin disabled
- MCLR = RG5 input pin enabled; MCLR disabled
- MCLR = MCLR pin enabled: RA5 input disabled
- MCLR = MCLR disabled; RA5 input pin enabled
- MCLR = RE3/MCLR pin is MCLR
- MCLR = RE3 pin is digital input. MCLR internally tied VDD
- MCLR = MCLR/VPP/RE3 pin function is digital input only
- MCLR = MCLR/VPP/RE3 pin function is MCLR

● Step 10 Calibration value

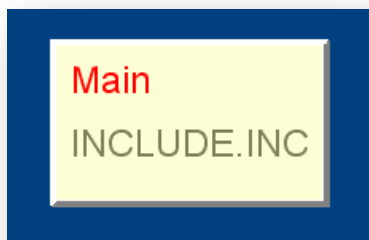
This entry is important for PIC's with internal RC-oscillator and deleting window (JW-types). This value which has to be entered, is investigated by Microchip during the production of the PIC. After a reset this value is written into the OSCCAL-register and takes care, that the internal oscillator runs if possible with 4 MHz. Higher values produce a higher clock frequency.

Note
Before first programming, this value has to be selected from the PIC. This value is always at the end of the ROM in form of a command. Otherwise this value is lost irrevocably after erasing of the PIC's.

Example PIC12CE519
The command MOVLW H'84' is at the address \$3FF
You must write down "84" and type in at Calibration value.

● Step 11 User-RAM

Who wants to include assembler over the INCLUDE-object, at this point it is possible to close an area within the PICs for Parsic. The **start-** and **end-**address must within one **RAM-bank**. Information about the memory areas of the chosen PIC can you see clicking on **PIC-Info** looking for > **Free-RAM**



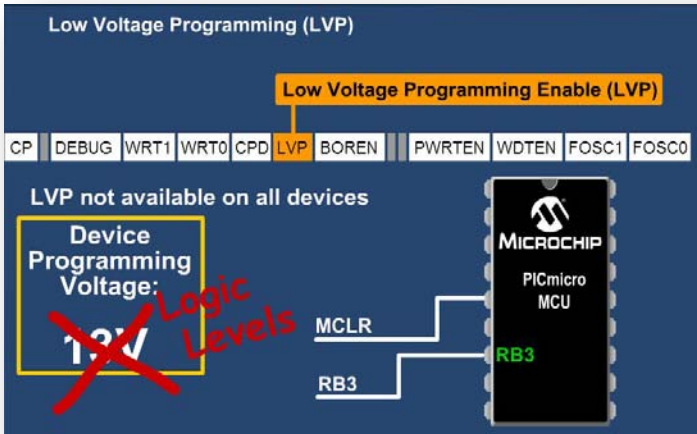
● **Step 12 Low Voltage Programming**

The PICmicro are programmed using a serial method. The Serial mode will allow the PICmicro to be programmed while in the user's system. The programming algorithm used depends on the operating voltage (VDD) of the PICmicro device. There are two methods for programming PICmicro. The first is called ICSP, In Circuit Serial Programming; the second method is called Low Voltage ICSP (LVP).

Both algorithms can be used with the two available programming entry methods. The first method follows the normal Microchip Programming mode entry of holding pins RB6 and RB7 low, while raising MCLR pin from VIL to VIH (13V ± 0.5V).

The second method, Low Voltage ICSP (LVP) for short, applies VDD to MCLR and uses the I/O pin RB3 to enter Programming mode.

It is important to note that when LVP is enabled it dedicates the RB3 pin for use during the low Voltage Programming. A rising edge on RB3 followed by a rising edge on MCLR will cause the device to enter programming mode. Please consult the datasheet for details.

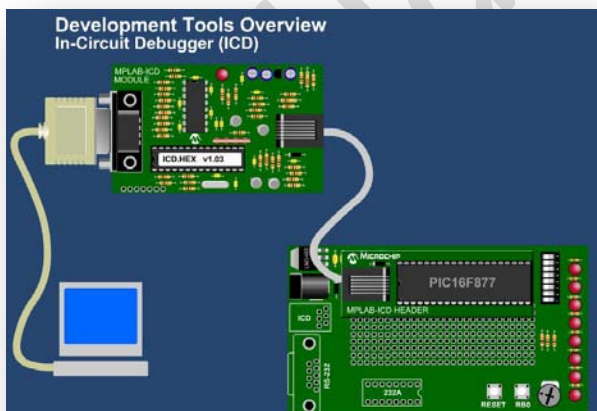


Low Voltage Programming

- RB3/PGM pin has PGM function
- RB3 is digital I/O, HV on MCLR must be used for programming
- RB3/PGM pin has PGM function
- Data EEPROM code protection off

● **Step 13 In-Circuit Debugger (ICD)**

One of Microchip's most innovative tools is the real-time In-Circuit Debugger called MPLAB ICD. The ICD utilizes the In-Circuit Debugging capability of the PICmicro. This feature, along with Microchip's In-Circuit Serial Programming (ICSP) protocol, offer cost effective in-circuit Flash programming and debugging from the MPLAB IDE interface. In the PARASIC4, PICmicro that have extension ICD can used by ICD debugger.



In-Circuit Debugger (ICD)

- In-Circuit Debugger disabled, RB6 and RB7 are general purpose I/O pins
- In-Circuit Debugger disabled, RB6 and RB7 are general purpose I/O pins
- In-Circuit Debugger enabled, RB6 and RB7 are dedicated to the debugger

Code protection – Different levels of code protection.

Code Protection is a user selectable feature that provides additional protection for your code and prevents program and data EEPROM from being read out. You'll notice that there may be several bits labeled CPx in the configuration word. They must all be programmed in order for full protection. When the device is programmed, a verify should be performed before the code protection bits are set. Since code protection prevents reading out the data, it is best to verify the part before setting the code protection bits. Each microcontroller device has different levels of code protection. Some devices allow the entire part to be code protected, others only allow partial protection. For example, the PIC16F87X family devices, will allow 4 different ranges of program memory to be code protected. They are:

- 0100h to 1FFFh
- 1000h to 1FFFh
- 0000h to 1FFFh

PIC18Fxx family devices, will allow up to 7 different ranges of program memory to be code protect.

Therefore, it is best to refer to your device datasheet for code protection ranges that affect you. It is important to note that each type of microcontroller has code protection limitations. For example, setting the code protection on a windowed device is a permanent change to that part. UV erasing the device will not erase the code protection bits. For this reason we do not recommend code protecting windowed devices. FLASH microcontrollers, however, are different. Setting the code protection on a FLASH device is not a permanent change, but, disabling the code protection will require a bulk erase. Visual Parsic V4 program only Flash PICmicro

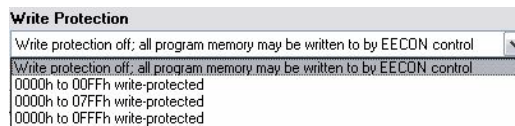
● Step 14 Code Protected – EEData code protection – EEData write protection

Activated, the controller (area) cannot be read or write after programming.

● Step 15 Write Protection

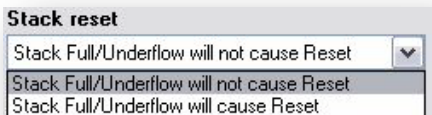
Write protection OFF : all program memory may be written to by EECON control. Write protection ON : program memory protected in the selectable memory area :

- 0000h to 00FFh
- 0000h to 07FFh
- 0000h to 0FFFh



● Step 16 Stack reset overflow – underflow

The key thing to understand about the 8 bit PIC architecture is that the stack size is fixed. It varies from a depth of 2 for the really low end devices to 31 for the high end 8 bit devices. The resets mechanism, in Parsic V4, for stack register, is available on Microchip's PIC18xxx . Please consult the appropriate device datasheet for device specific information. Microchip has provided for the possibility of two disastrous events for the stack register : the overflow and underflow. For this reason, the overcoming of the stack causes the reset of the microcontroller.



Memory Area Protection PIC18Fxx series

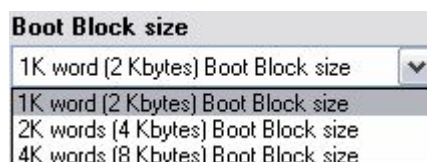
Among the many features built into Microchip's Enhanced FLASH Microcontroller devices is the capability of the program memory to self-program. This very useful feature has been deliberately included to give the user the ability to perform bootloading operations. Devices like the PIC18Fxx are designed with a designated 'boot block', a small section of protectable program memory allocated specifically for bootload firmware. The PIC18xx series, allow a detailed selection of memory areas that can be protected from external reading or writing. Normally, these functions are disabled and must be enabled depending on your security needs. While debugging, you should not enable these features.

Below we listed the memory areas that can be protected according to the PICmicro used :

● Step 17 Boot Block size 1K word

- 2K words - 4K words

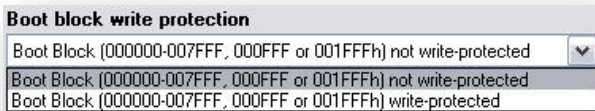
Select size :



● **Step 18 Boot block code protection (000000- 0007FF)**



● **Step 19 Boot block write protection (000000-007FFF or 001FFF)**

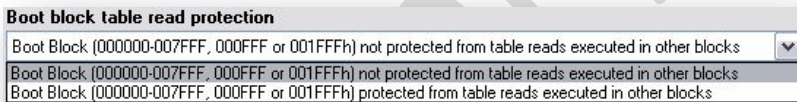


Enhanced devices have two memory spaces. The program memory space and the data memory space. The program memory space is 16 bits wide, while the data memory space is 8 bits wide. Table Reads and Table Writes have been provided to move data between these two memory spaces through an 8-bit register (TABLAT). The operations that allow the processor to move data between the data and program memory spaces are:

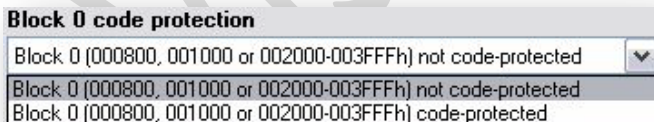
- **Table Read (TBLRD)**
- **Table Write (TBLWT)**

Table Read operations retrieve data from program memory and place it into the data memory space.

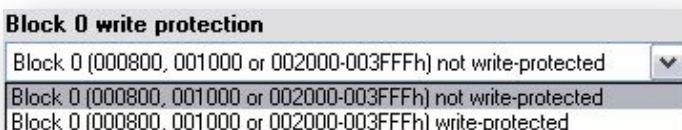
● **Step 20 Boot block table read protection (000000-007FFF or 001FFFh)**



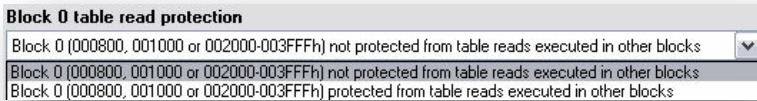
● **Step 21 Block 0 code protection (000800, 001000 or 002000-003FFFh)**



● **Step 22 Block 0 write protection (000800, 001000 or 002000 – 003FFFh)**



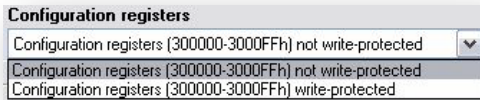
● **Step 23 Block 0 table read protection (000800, 001000 or 002000-003FFFh)**



Repeat steps 21- 22 - 23 up to block table 7.

● **Step 24 Configuration Register**

The Device Configuration registers allow each user to customize certain aspects of the device to fit the needs of the application. Device Configuration registers are nonvolatile locations in the program memory that hold settings for the PIC18 device during power-down. The Configuration registers hold global set-up information for the device, such as the oscillator source, Watchdog Timer (WDT) mode, code protection settings and others. The Device Configuration registers are mapped in program memory locations, starting at address 300000 to 300FFh, and are accessible during normal device operation. This region is also referred to as 'configuration space'. The Configuration bits can be programmed (read as '0') or left unprogrammed (read as '1') to select various device configurations



Additional Notes :

All notes in this text are only for the PIC18F family

Code-Protection Bits - CP(x):

If only the CP(x) bits are enabled, you cannot read or write block(x) with an external Programmer. (PICkit 2/3 or equal)
 BUT, - when there is code inside the PIC, that can transfer program-memory from each block(x) over the serial-interface to outside,
THIS WILL WORK !!

Write-Protection Bits - WRT(x):

If only the WRT(x) bits are enabled, you cannot change the code inside block(x). (using an external Programmer or self writing)
BUT - you can read Block(x)

External Block Table Read Bits - EBTR(x)

For example only EBTR(3) and EBTR(4) are enabled and all other EBTR(x) bits are disabled:

1. Code inside block(4) has full access to all blocks (including block(4)), BUT has no access to block(3)
2. Code inside block(3) has full access to all blocks (including block(3)), BUT has no access to block(4)
3. All other blocks have full access to all blocks except block(3) and block(4)



So, now take a small part of your brain, to find out the right combination for your little PIC !

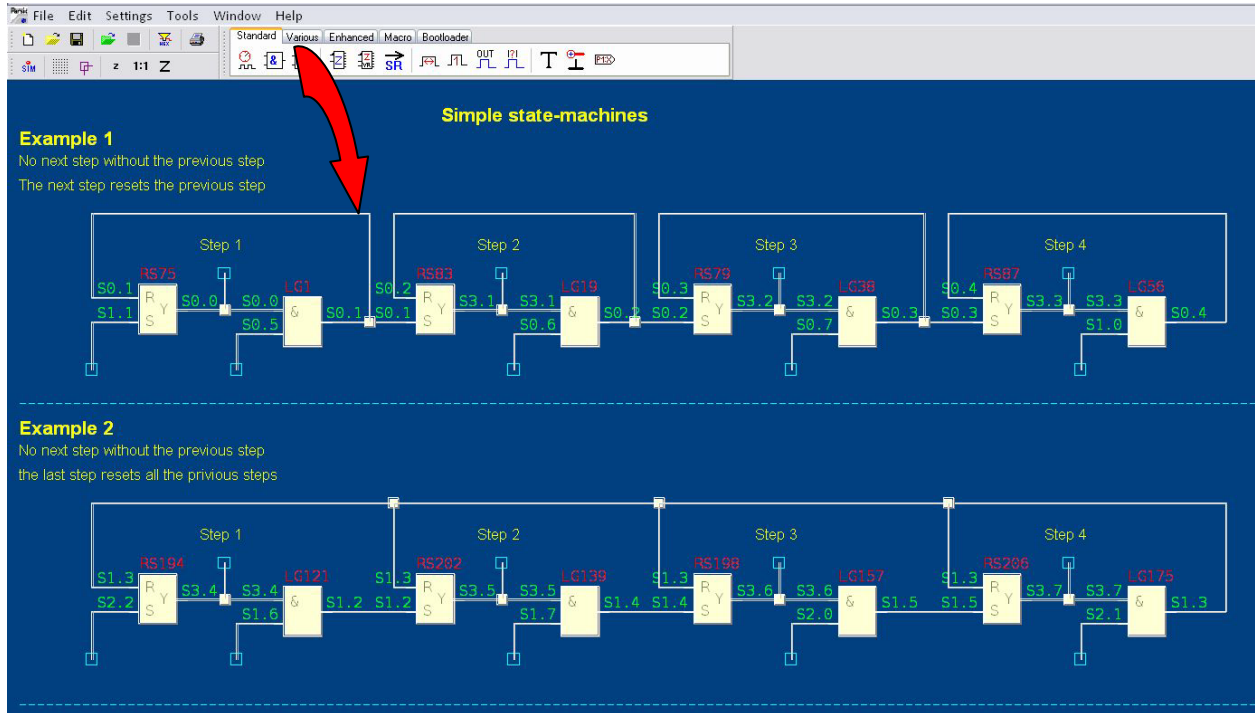
Legend : (x) = (B=boot), 0, 1, 2, 3,...

Schematic - User Functional Block

3.0 Description

Insert Block on schematic

Use left mouse button to drag a selected block (one click) and drop in design area (one click).



Removing Block.

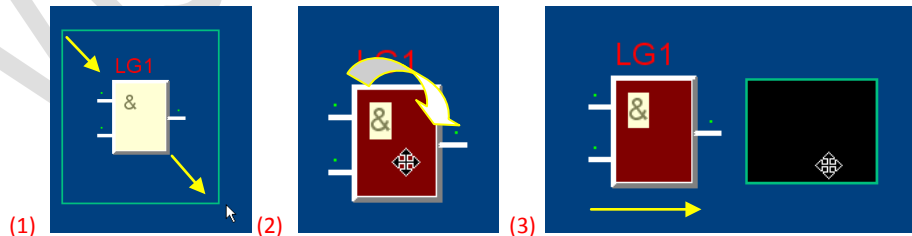
Push left mouse button and move the cursor in diagonally area, around the block; release and, for removing, press **Delete** button. Note : when you have selected the object, it change color.

Move Block

Move the cursor above the Block. Click on left mouse button and drag the Block in another position: release the button when you have reached the new position

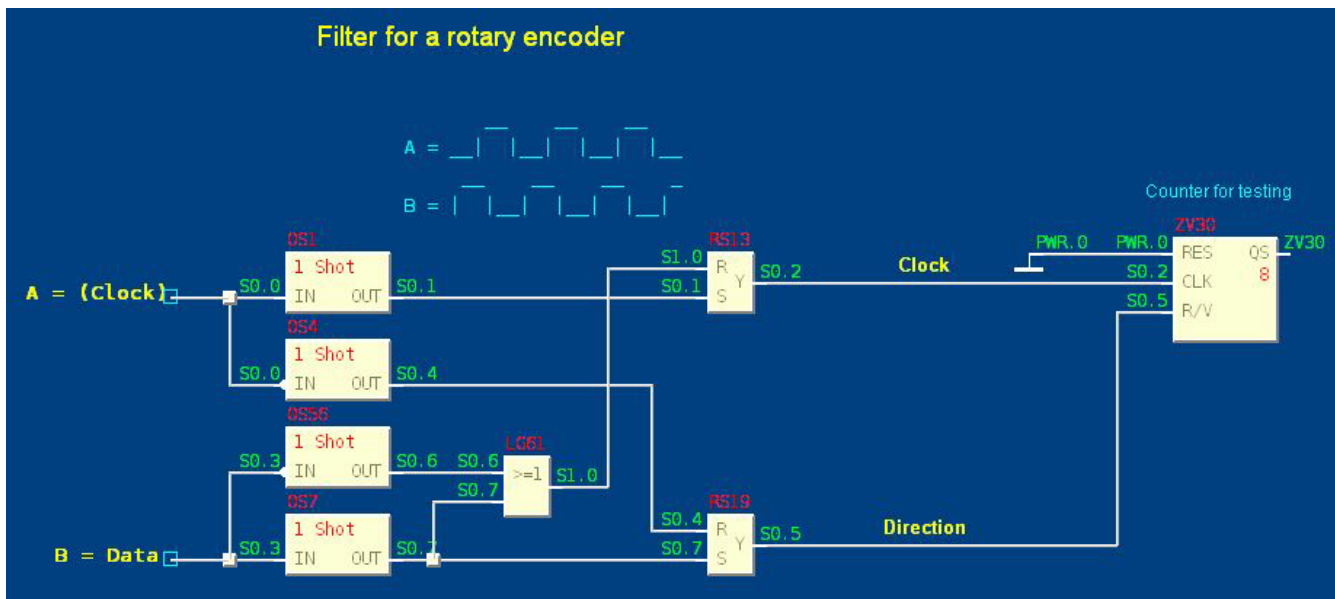
Move Block or selected schematic area

Push the left mouse button and move the cursor in the diagonally area you want select (1). Release mouse button: all area selected change color (2). Push the left mouse button above selected object and drag the object in a new area : release mouse button (3)



Connecting terminal Blocks

The port of one block can be connected to the port of another block or to an existing connection. For connect two terminal of different block, (1) move the mouse cursor above a terminal of the first Block, (2) push left mouse button and drag the connection to another port, (3) release the mouse button when you are above the terminal. The connection is valid only when Parsic allocate equal progressive number name to the terminals. When you move the cursor, **use the shift key, to change the direction of the cursor**. In the following figure below a complete schematic.



Schematic example

In the **datasheets** of Microchip the connection pin's are marked by RA0, RA1, RA3, RB7, RC6 etc. You will find a little further also the description PORTA, PORTB etc. in principle. Parsic uses the description RX.X or GP.X.

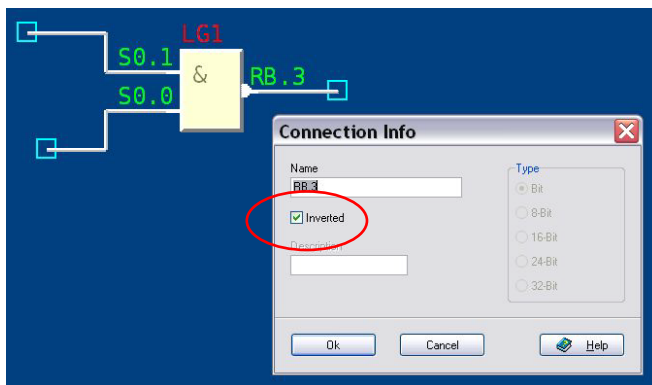
To connect the Pin RB3 with the output of an AND-Gate, you have to click with the right mouse button at the output of the AND-Gate. In the following dialog enter the name RB.3. Like also at the signals, you can turn over the function of the output or input with a click on the Invert-Button. As you have placed RB3 on an output of an object now, Parsic will program the PIC RB3 during initialization as an output, also.

Notes

- Once output, always output
- Unused Pins are initialized as input

Special features (at some devices)

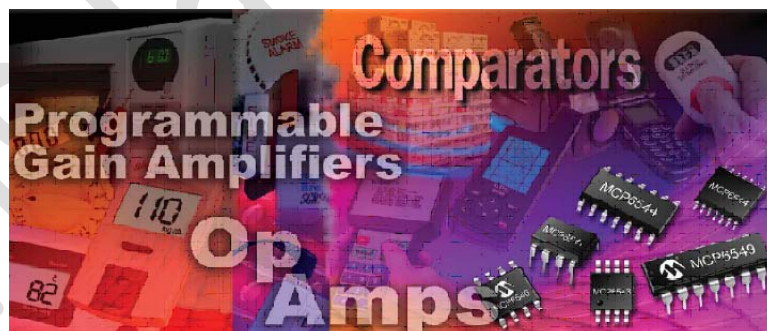
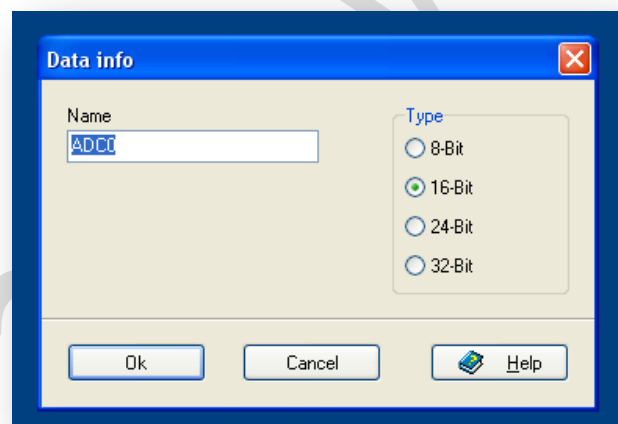
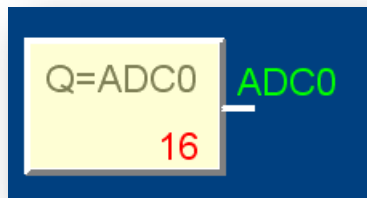
- RA4 output programmed, it is a Open-Collector type (maybe a pull-up resistor is required)
- RB0...7 input programmed, pull-up resistors could be programmed internally
- RB0, 4...7 can trigger an Interrupt



The analogue inputs

Take a data source and put it in the active window. Click with the right mouse button on the data source and enter the value e.g. ADC0. From the output of data source you can pull connections to the Schmitt-Trigger or to other objects with a 8-Bit inputs. The allocation of the A/D channels under Parsic is the following:

Pin	A/D-channel inside Parsic
AN0	ADC0
AN1	ADC1
AN2	ADC2
AN3	ADC3



Name objects

Parsic usually allocates the names for objects automatically and numbers these. Under special circumstances, however, it can be required to allocate a solid identifier. Click with the right mouse button into the object below on the left. Change the first two characters of the name in the dialog. A Maximum of 30 characters for the name is allowed. The first character must be a letter, always.

Example

The AND-gate has the name LG12.

Change the name to L1.

Now the name will not be changed by Parsic anymore.

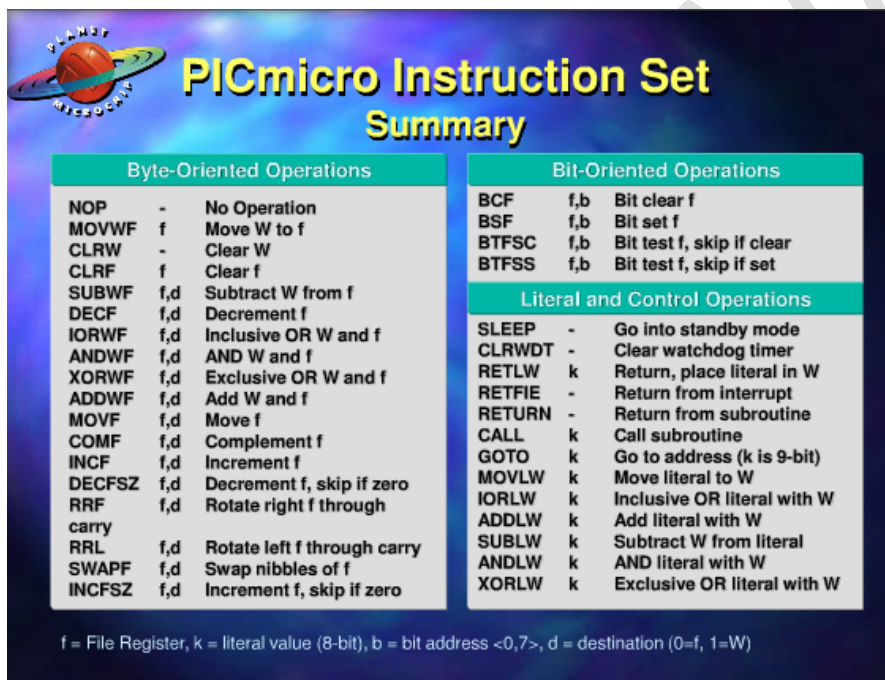
Special names under Parsic

S... used for signals

Rx, GPx used for allocate in- and output of the PIC

ADCx describes an analogue input, where "x" is a number between "0" and "7"

The following names are commands of the Microchip assembler. Parsic accepts the names, but the assembler will create an error message.



The slide features a logo with a globe and the text 'PICmicro INSTRUCTION SET SUMMARY'. It is divided into three sections: 'Byte-Oriented Operations', 'Bit-Oriented Operations', and 'Literal and Control Operations'. A legend at the bottom defines the symbols: f = File Register, k = Literal value (8-bit), b = bit address <0,7>, d = destination (0=f, 1=W).

Byte-Oriented Operations		Bit-Oriented Operations	
NOP	- No Operation	BCF	f,b Bit clear f
MOVWF	f Move W to f	BSF	f,b Bit set f
CLRW	- Clear W	BTFS	f,b Bit test f, skip if set
CLRF	f Clear f	BTFS	f,b Bit test f, skip if set
SUBWF	f,d Subtract W from f	Literal and Control Operations	
DECf	f,d Decrement f	SLEEP	- Go into standby mode
IORWF	f,d Inclusive OR W and f	CLRWD	- Clear watchdog timer
ANDWF	f,d AND W and f	RETLW	k Return, place literal in W
XORWF	f,d Exclusive OR W and f	RETFIE	- Return from interrupt
ADDWF	f,d Add W and f	RETURN	- Return from subroutine
MOVF	f,d Move f	CALL	k Call subroutine
COMF	f,d Complement f	GOTO	k Go to address (k is 9-bit)
INCF	f,d Increment f	MOVLW	k Move literal to W
DECFSZ	f,d Decrement f, skip if zero	IORLW	k Inclusive OR literal with W
RRF	f,d Rotate right f through carry	ADDLW	k Add literal with W
RRL	f,d Rotate left f through carry	SUBLW	k Subtract W from literal
SWAPF	f,d Swap nibbles of f	ANDLW	k AND literal with W
INCFSZ	f,d Increment f, skip if zero	XORLW	k Exclusive OR literal with W

f = File Register, k = Literal value (8-bit), b = bit address <0,7>, d = destination (0=f, 1=W)

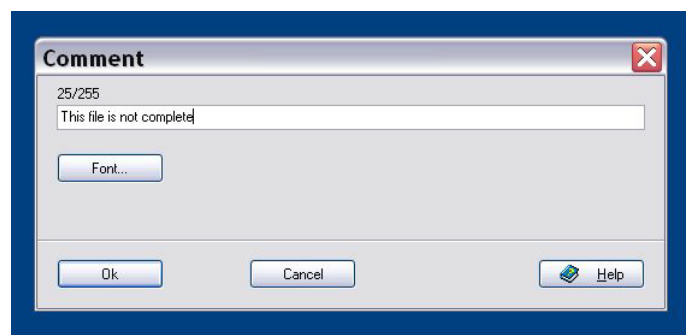
Note : An exact description of the assembler commands you will find e.g. in the quick-reference-guide of MPASM of Microchip.

<http://ww1.microchip.com/downloads/en/devicedoc/31029a.pdf>

Comment

A comment is generally an added piece of information, or an observation. These are usually marked with an abbreviation, such as Timer, Clock, Converter or "this file is not complete".....









Good use of the comments is very important, as the meaning and purpose of a sequence of instructions is difficult to remember, after a long time. Wise use of comments can greatly simplify the problems of understanding the schematic that need change in the future.

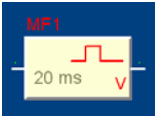

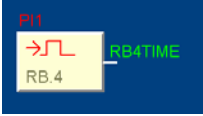


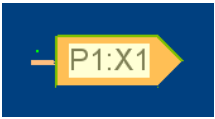


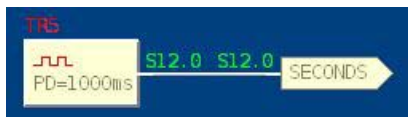
3.1 Functional Blocks

Here is a short overview of which functional Block supported in Visual Parsic 4. You can see what each Block does and it representative icon. For other information use HTML help.

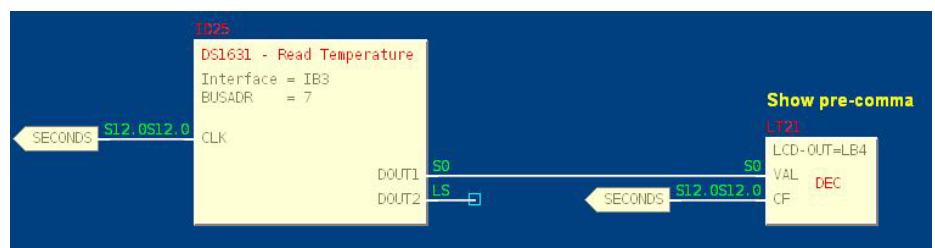
Standard Blocks

Toolbar Icon	Name	Description																									
	Timer 3.11	The timer distributes a definite frequency to its output. With the right mouse button the name and the period time can be changed. An error message will follow after invalid input of the period time.																									
  	Gate 3.37	<p>A gate has up to 16 inputs. All inputs and the output can get inverted.</p> <table border="1" data-bbox="788 651 1283 943"> <thead> <tr> <th>Input 1</th> <th>Input 2</th> <th>AND-Gate</th> <th>OR-Gate</th> <th>Ex-Or-Gate</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	Input 1	Input 2	AND-Gate	OR-Gate	Ex-Or-Gate	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	1	0
Input 1	Input 2	AND-Gate	OR-Gate	Ex-Or-Gate																							
0	0	0	0	0																							
0	1	0	1	1																							
1	0	0	1	1																							
1	1	1	1	0																							
	R/S FF 3.42	<p>Condition for setting Condition for reset</p> <p>S = High R = High</p> <p>R = Low S = High or Low</p> <p>Note</p> <p>Inverted in - or outputs, turn over the function.</p>																									
	Counter 3.19	<p>The counter is a 1 byte variable and therefore it can count from 0 up to 255. The outputs (max. 8) can be distributed on various bits.</p> <ul style="list-style-type: none"> • Source text is created only for the outputs which have a signal name. • All in- and outputs can be inverted. • Priority has the Reset-Input. 																									
	Counter UP-Down 3.18	<p>The counter is a 1 byte or 2 byte variable and therefore it can count from 0...255 or 0...65535. The outputs (max. 16) can be distributed on various bits. The QS-Output is the collecting output and can get connected directly with another object with 8 or 16 bit input. The R/V input determine the counter advance sense.</p> <ul style="list-style-type: none"> • on low level, the counter counts upwards. • on high level, the counter counts downwards. 																									
	Shift Register 3.46	Every shift-register is a 1 bytes large variable. At every positive impulse at the CLK-input, the data starting at Bit-0 are taken over into the shift register. The output (max. 8) can be distributed to various bits.																									

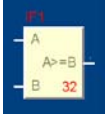
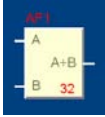
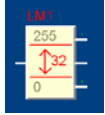
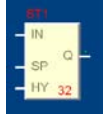
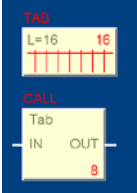

Toolbar Icon	Name	Description
	Mono-Flop, On-delay, Off-delay 3.52	Depending on input allocation, the Mono-Flop get triggered by a positive or negative edge. After a click with the right mouse button a dialog for entering appears : <ol style="list-style-type: none"> 1. Pulse duration in milliseconds or by detail of a variable 2. Retriggerable or not 3. The name 4. Mono-Flop or Delay 5. Inverted in- or outputs turn over the function. Read more
	One-Shot 3.51	This object generates an impulse for one program pass. At every positive edge at the input, the output is setted. Independently of the status of the Input, the output is reseted at the next pass. The input and output also can get inverted.
	Impuls - Input 3.31	With the impulse input the length of an external impulse is determined with a resolution of 16 bits. For that Parsic uses PORTB change interrupt (RB4..7). The precision depends on several factors: <ul style="list-style-type: none"> • The clock frequency • Number of used timers and Mono-Flops. The time is decided with help of the internal clock frequency. This corresponds with a resolution of 1 microsecond (20 MHz -> 200 nanoseconds) at a 4 MHz quartz. Read more
	Comments 3.33	Only for schematic commenting , no code created.
	GND- VCC 3.28	With the GND and Vcc object it is possible to put unused inputs to a defined level. (e.g. reset at a counter) Conduction's which are connected at this object gets GND = always low Vcc = always high
	Label 3.33	Labels are competent for the connection of the same signal-names on different function plans. Parsic provides a clear name. It is possible for you to change this name. With the right mouse button you reach the dialog for the settings. You can choose there <ol style="list-style-type: none"> 1. whether the label is source or destination 2. assigning the source label to the destination label. 3. change the name To restore the name, allocated by Parsic, simply enter "P:X". Read more

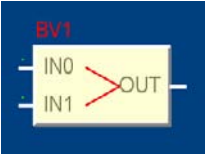

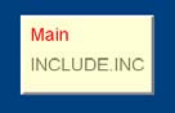
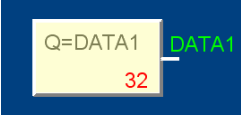


Label Example





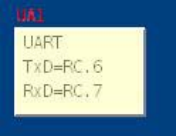

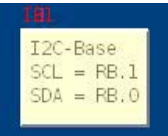

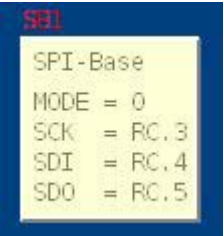

Various Blocks

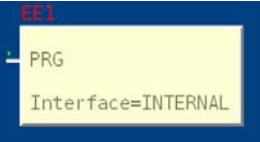

Toolbar Icon	Name	Description								
	Compare 3.17	With the Compare-Object, variables and constants can be compared with each other and in dependence of Bits they could be setted or deleted. The upper (A) variable or constant is always compared with the below (B) variable or constant and accordingly the output is setted to high or low. The output can also be inverted In the Info-dialog (right mouse button) you can adjust how to compare.								
	Arithmetic 3.5	All arithmetical functions work without omens. The first operand always stands above. The following possibilities are available for the selection : <table border="1" data-bbox="965 551 1179 730"> <tr><td>+</td><td>addition</td></tr> <tr><td>-</td><td>subtraction</td></tr> <tr><td>x</td><td>multiplication</td></tr> <tr><td>/</td><td>division</td></tr> </table> <p>Further you got the possibility to select, whether the evaluation shall be carried out with a resolution of 8 or 16-Bit.</p> <p>Note : Possible occurring overflows are ignored.</p> <p>Example A 16x16 multiplication results usually 256. At a resolution of 8-Bit the result is cutted to 8-Bit. The result is 0. With a division by 0, the result is set to 255 (8 bit) or 65535 (16 bit).</p>	+	addition	-	subtraction	x	multiplication	/	division
+	addition									
-	subtraction									
x	multiplication									
/	division									
	Limiter 3.36	The limiter checks the contents of the input variables with the upper and lower limit. <ul style="list-style-type: none"> • If the input > upper limit, then output = upper limit • If the input < lower limit, then output = lower limit <p>Otherwise output = input.</p>								
	Schmitt – Trigger 3.45	The inputs work with 8-bit up to 32-bit resolution. For SP and HY input applies: The values may be constant or variable The IN input must be variable. It is possible to invert the function of the output. Read more								
	TAB – Call 3.50	The table stores data inside the ROM. The access on the table always takes place over the CALL-Object. Tables always created as a subroutine inside the ROM. The Call object serves to invoke subroutines and tables. Read more								
	Multiplexer 3.43	In electronics, a multiplexer (or mux) is a device that selects one of several analog or digital input signals and forwards the selected input into a single line. A multiplexer of n input has a selection line (binary select), which is used to select which input line to send to the output. The multiplexers are mainly used to increase the amount of data to process, saving resources within the PIC. A multiplexer is also called a data selector. Sets the range for in - and outputs 8/16/24/32 - bit								

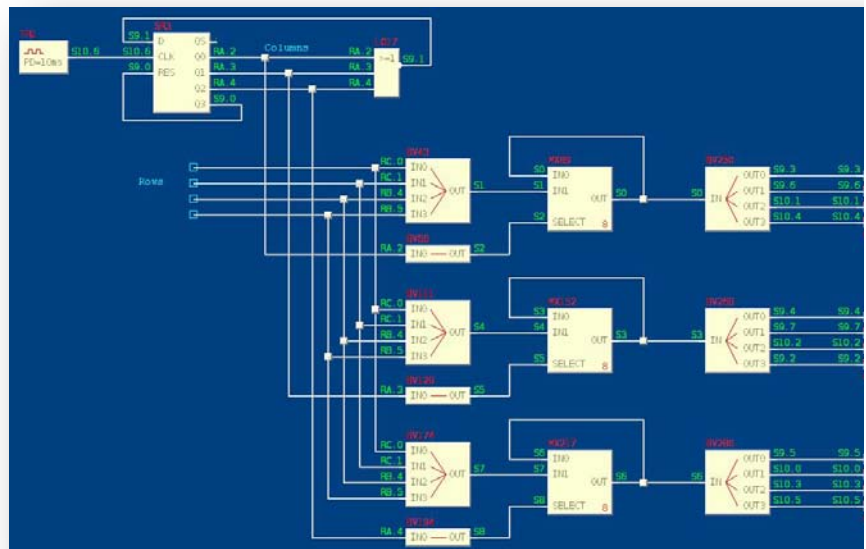
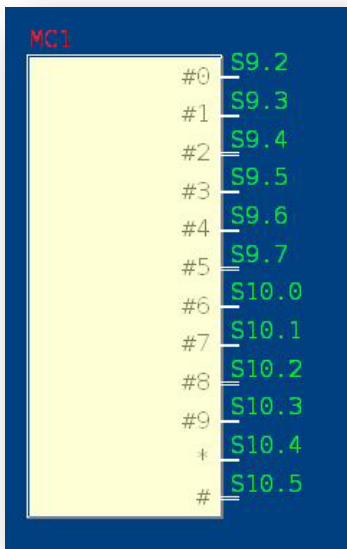
	<p>Dispatcher Encoder-Decoder</p> <p>3.26</p>	<p>An encoder is a circuit or algorithm that compresses multiple binary inputs into a smaller number of outputs. The output of a encoder is the binary representation of the original number starting from zero of the most significant input bit. A decoder is a device which does the reverse operation of an encoder, undoing the encoding so that the original information can be retrieved. The same method used to encode is usually just reversed in order to decode.</p>
	<p>Sleep</p> <p>3.47</p>	<p>With this object it is possible to move the microcontroller into the sleep mode. If the IN-Input has high level, then the sleep mode is activated, the program is stopped and the oscillator is switched off.</p>
	<p>Include</p> <p>3.32</p>	<p>INCLUDE-files are to embed subroutines and assembler source into Parsic. With the right mouse button you can chose, on which place in the created source text the INCLUDE-file will be embedded.</p>
	<p>DATA source</p> <p>3.20</p>	<p>The data source accomplishes several tasks: Supply variables for own program parts, which are embedded with the "Include object". Allocation of the A/D-converter at PIC's with A/D.</p> <p>Read more</p>

Visual Parsic

Enhanced Blocks - Macro

Toolbar Icon	Name	Description
	LCD Modul Info 3.34	All LCD modules, based on the Hitachi HD44780 or KS0073, are supported .To have enough in- and outputs at small PIC's, Parsic use only 6 of 11 control lines. Read more
	LCD-Module, display text or variable 3.35	With this object texts or variable can be distributed on a LCD module. At every edge at the CF-Input, the text or the variable is shown on the predefined position. Read more
	UART Base 3.54	Universal Asynchronous Receiver Transmitter With this object you define the settings of the serial interface. Not all microcontrollers have a built-in UART. If you use a PIC without hardware UART, an error message is created.
	UART Data 3.55	With this object data are sent or received about the serial interface. Active-Output (ACT) For both function applies: <ul style="list-style-type: none"> • Transferring = The ACT-Output remains on high level, until the data is transferred. • Receiving = The ACT-Output remains on high level, until the data are correctly received or an error is occurred.
	I2C Base 3.29	The I²C-Base object is the base for accessing the I²C-bus . It provides subroutines, for accessing the I²C-bus . I2C-Data accesses the selected device over I²C-Base object. Read more
	I2C Data 3.20	The I²C-Base object is the base for accessing the I²C-bus . It provides subroutines, for accessing the I²C-bus . I2C-Data accesses the selected device over I²C-Base object. Read more information
	SPI Base 3.48	The SPI-Base object is the base for accessing the SPI-bus . It provides subroutines, for accessing the SPI-bus . The SPI-Data object accesses the selected device over SPI-Base object. Read more
	SPI Data 3.49	The SPI-Base object is the base for accessing the SPI-bus . It provides subroutines, for accessing the SPI-bus . The SPI-Data object accesses the selected device over SPI-Base object. Read more

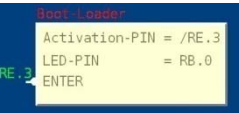
	<p align="center">EE- Prom</p> <p>3.25</p>	<p>With the EEPROM-Object it is possible to save the contents of variables in an internal or external EE-Prom. This happens at every positive edge at the input of the object. After a reset, the indicated variables are read in automatically. At the selection all present variables are displayed in the left list. All variables which you would like to save must be moved into the right list. Read more</p>
	<p align="center">Macro</p> <p>3.41</p>	<p>In a macro, multiple objects can be combined into one object. Saved macros can be added to the function-plan with File > Load macro. If you click with the right mouse-button on a macro, a function-plan will be opened with the title "Macro". It behaves like a normal function-plan. In addition, the Toolbar > Macro will show two more objects. These objects are needed to add inputs and outputs to the macro. Each macro can be added a maximum of 16 inputs and 16 outputs. As long as a macro is opened, it can be saved by choosing File > Save macro for other projects Read more</p>



Macro

- (1) Left: module and his terminals.
- (2) Right: schematic inside Macro.

Bootloader

	<p align="center">Bootloader</p> <p>3.9</p>	<p>The bootloader is a small program that receives data via the serial interface and stores them in the program-memory or EEPROM of the PIC. It is so designed, that it never overwrites itself. Read more</p>
---	---	---

3.2 Block Description

In Parsic 4, a function block contains input variables, output variables, and his properties.

The Function Block Diagram is a graphical language for PICmicro design, that can describe the function between input variables and output variables. A function is described as a set of elementary blocks. Input and output variables are connected to blocks by connection lines. Inputs and outputs of the blocks are wired together with connection lines, or links. Single lines may be used to connect two logical points of the diagram:

An input variable and an input of a block

An output of a block and an input of another block

An output of a block and an output variable

The connection is oriented, meaning that the line carries associated data from the left end to the right end. The left and right ends of the connection line must be of the same type.

- 3.3 Absolute value
- 3.4 Analog input
- 3.5 Arithmetic divide
- 3.6 Arithmetic multiply
- 3.7 Arithmetic subtract
- 3.8 Arithmetic sum
- 3.9 Bootloader
- 3.10 Call Table
- 3.11 Clock generator
- 3.12 Compare <
- 3.13 Compare <=
- 3.14 Compare <>
- 3.15 Compare =
- 3.16 Compare >
- 3.17 Compare >=
- 3.18 Counter Up/Down 8-16 bit
- 3.19 Counter 8-bit
- 3.20 DAT (Analog or Include)
- 3.21 Decoder
- 3.22 De-Multiplexer
- 3.23 Digital Input
- 3.24 Digital output
- 3.25 EE-Prom
- 3.26 Encoder
- 3.27 Edit Device
- 3.28 Ground & Vcc
- 3.29 I2C – Base
- 3.30 I2C – Data
- 3.31 Impulse input
- 3.32 Include
- 3.33 Label
- 3.34 LCD – Base
- 3.35 LCD – Text
- 3.36 Limiter
- 3.37 Logical AND operator
- 3.38 Logical NOT operator
- 3.39 Logical OR operator
- 3.40 Logical XOR operator
- 3.41 Macro
- 3.42 Memory – Set/Reset latch
- 3.43 Multiplexer
- 3.44 PWM output
- 3.45 Schmitt Trigger
- 3.46 Shift register
- 3.47 Sleep
- 3.48 SPI – Base
- 3.49 SPI – Data
- 3.50 Table Editor
- 3.51 Timing Falling edge

- 3.52 Timing Pulse timer
- 3.53 Timing Rising Edge
- 3.54 UART – Base
- 3.55 UART – Data

Visual Parsic V4

3.3 Absolute value - Comparator



The Absolute Value function block provides the absolute value of an integer input value.

Possible combinations : > / < / >= / = / <= / <>

Inputs : selectable 8...32-bit integer

A = 8... 32-bit integer input variable or constant value

B = 8... 32-bit integer input variable or constant value

A...B = digital output value

>	greater
>=	greater or equal
=	equal
<=	less or equal
<	less
<>	different

3.4 Analog input



Analog input (8-32 bit integer) is used to read one analog input value, by DAT Functional Block.

The data source accomplishes several tasks:

- Supply variables for own program parts, which are embedded with the "Include object".
- Allocation of the **A/D-converter** at PIC's with A/D.

With a click with the right mouse button you can enter an arbitrary value or name.

Resolution	Range	Name(s) of the variable (example)
8-Bit	0...255	DAT
16-Bit	0...65535	DAT, DAT_1
24-Bit	0...16777215	DAT, DAT_1, DAT_2
32-Bit	0...4294967295	DAT, DAT_1, DAT_2, DAT_3

The analog inputs are distinguished by the following names : **ADC0 – ADC1 – ADC2..... ADC15**

Notes

- It is not allowed to start a signal name with 'S'.
- Wires which connected at the data source, always assigned the value, which given the data source as signal name.

3.5 Arithmetic divide



The Divide (A : B) function block divides one input value by another input value and generates an output value.

A = 8...32-bit integer first variable or constant value

B = 8...32-bit integer second variable or constant value

A:B = operation result

Possible occurring overflows are ignored

3.6 Arithmetic multiply

The Multiply (A x B) function block multiplies two inputs values and generates an output value.

A = 8...32-bit integer first variable or constant value

B = 8...32-bit integer second variable or constant value

A x B = operation result.

Possible occurring overflows are ignored

3.7 Arithmetic subtract

The subtract ($A - B$) function block subtracts one input variable from another input value and generates an output value.

A = 8....32-bit integer first variable or constant value

B = 8....32-bit integer second variable or constant value

$A - B$ = operation result.

Possible occurring overflows are ignored

3.8 Arithmetic sum

The Add function block ($A + B$) sums the values of two to inputs and generates an output value.

A = 8....32-bit integer first variable or constant value

B = 8....32-bit integer second variable or constant value

$A + B$ = operation result.

Possible occurring overflows are ignored

3.9 Bootloader



The bootloader is a small program that receives data via the serial interface and stores them in the program-memory or EEPROM of the PIC. It is so designed, that it never overwrites itself.

Activation bootloader (only Power-On)

This port is polled when switching on the PICs. When active, the program branches into the bootloader, otherwise an existing program called.

With the setting "if low" or "if high" you decide which level must on the selected port to enable the bootloader.

Signaling "bootloader active" at

If the bootloader is active, this port is set to high.

Baudrate

Here you can select the speed, with which the data should be transferred.

Depending on the frequency of the PIC, the resulting Baudrate deviates more or less from the selected Baudrate. The indicated error should not exceed 3%.


Connection "Enter" (running program)

In order to have the possibility to activate the boot-loader from a running program, the Enter-connection exists.

There are three choices to use this port:

- Same port and same polarity as in "only power-on" (inverted = low, not inverted = high)
- Connection remains empty. The bootloader can only be activated at power-on.
- Through any other object with a bit-output.

Burning the bootloader


1. Make sure that the **boot-manager** is **disabled**.
2. Place the bootloader in your sheet.
3. Check in **Settings -> Microcontroller** the frequency and make sure that "Write protected" is **disabled**.
4. Click with the right mouse-button on the boot-loader and choose a valid transfer rate and determine the connections for the boot loader activation.
5. Compile the program with <F10> or click on the icon  Build HEX file.
6. Burn the program with a "normal" programmer.
7. Remove the programmer. The port "**bootloader active**" should go high.
8. Activate the **Boot-Manager**.

Hint

- If the bootloader is "freshly burned", the PIC always starts in the bootloader-mode, until a valid program is received.

Boot-Manager

The boot-manager establishes the connection between Parsic and the PIC to be programmed. When you activate the boot-manager, the adjusted COM-port (default **COM1**) opens. If the COM port doesn't exist or is already in use (Parsic started 2 times ?), a short error message will be displayed.

To change the com-port, click the icon 



and select a com-port and the same Baudrate you also selected in the bootloader. The parity must always be set to "none". Now the boot-manager is ready. Be sure that the PIC is in bootloader-mode.



When you click the icon, the program will be compiled and when there are no errors it will be transmitted to the PIC. If everything was successful, the program starts.

If you inadvertently set the PIC into the bootloader-mode, click the icon . The boot-manager then sends an "end-of-line" to the PIC. The PIC leaves the bootloader-mode and starts over the existing program.

Possible errors

-  <time-out> The PIC is not in bootloader mode or the Baudrate isn't correct. In any case the PIC doesn't answer.
-  <checksum-error> The Baudrate is eventually too high.



3.10 Call Table

The **Call object**, 8-16 bit integer, serves to invoke subroutines and **tables**.

Input 8-16 bit integer

Output 8-16 bit integer



Use **table editor** to set values of the entries.

Tables

The contents of the variables at the IN input is the index of the entry in the **table**.

Example

The table which shall invoked has a length of 16. The contents of variables at IN-Input must lie between 0 and 15. If the value is higher it can be that the PIC executes a reset.

-  Type 8-bit = the output is a 8 bit variable.
-  16-bit = the output is a 16 bit variable. This type have to be used accessing 16-bits tables.



3.11 Clock generator

The timer distributes a definite frequency to its output. With the right mouse button the name and the period time can be changed. An error message will follow after invalid input of the period time.

Name

The name of the timing circuit must be clear. (max. 30 characters) and Parsic is allocating automatically.

If a 8 bit variable is indicated for the period time, then the period time is calculated as follows:
(value of the variables (0 -255) x 2 x 256)+program transmission delay of the complete program.

Example

You have typed in the variable ZV1. It describes a counter (ZV1).

At the moment this counter has the value 16.

Than the timer has at present a period time of:

$16 \times 2 \times 256 = 8192 \text{ clocks} = 8192 \mu\text{S}$ at 4 MHz plus program transmission delay.

If a 16 bit variable will indicate for the period time, then the period time is calculated as follows:

(value of the variables (0 -65535) x 2 x 256)+program transmission delay of the complete program.

3.12 Compare

Read [3.3 Comparator](#)

3.18 Counter Up/Down 8-16 bit



In digital logic and computing, a **counter** is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

The counter is a 1 byte or 2 byte variable (8-16 bit integer) and therefore it can count from 0 -255 or 0 -65535. The outputs (max. 16) can be distributed on various bits. The QS-Output is the collecting output and can get connected directly with another object with 8 or 16 bit input.

The R/V input determine the counter advance sense.

- On low level, the counter counts upwards.
- On high level, the counter counts downwards.

Notes

- Source text is created only for outputs which have a signal name.
- All in- and outputs (apart from QS) can get inverted.
- Priority has the Reset-Input.

Name

The name of the counter must be unique (max. 30 chars) and is assigned automatically by Parsic.

3.19 Counter 8-bit



The counter is a 1 byte variable (8-bit integer) and therefore it can count from 0 up to 255. The outputs (max. 8) can be distributed on various bits.

- Source text is created only for the outputs which have a signal name.
- All in- and outputs can be inverted.
- Priority has the Reset-Input.

Name

The name of the counter must be unique (max. 30 chars) and is assigned automatically by Parsic.

3.20 DAT

Read [3.4 Analog input](#)

3.21 Dispatcher or Encoder - Decoder



An encoder is a circuit or algorithm that compresses multiple binary inputs into a smaller number of outputs. The output of an encoder is the binary representation of the original number starting from zero of the most significant input bit. A decoder is a device which does the reverse operation of an encoder, undoing the encoding so that the original information can be retrieved. The same method used to encode is usually just reversed in order to decode.

Encoder 8-16 bit integer read binary input value

Decoder 8-16 bit integer read byte input value

Function bit -> byte

Specifically definite bits will be transferred to the output variable.

Hold for this

- Output Bit-0 = IN0
- Output Bit-1 = IN1
- Output Bit-2 = IN2
- ...
- Output Bit-15 = IN15
- Notes
- If the number of inputs is > 8, then the variable at the output is a 16 bit variable.
- The inputs don't have to be filled throughout.
- For inputs which are not occupied no source text is created.

Function byte-> bit

Definite Bits of the input variable could be transferred specifically to single Bits of the outputs.

Hold for this

- OUT0 = input Bit-0
- OUT1 = input Bit-1
- ...
- OUT15 = input Bit-15

Notes

If the number of the output is > 8, then the variable at the input is a 16 bit variable.

The outputs don't have to be filled throughout.

For outputs which are not occupied no source text is created.

3.22 De-Multiplexer



An electronic multiplexer can be considered as a multiple-input, single-output switch, and a demultiplexer as a single-input, multiple-output switch. In electronics, a multiplexer (or mux) is a device that selects one of several analog or digital input signals and forwards the selected input into a single line. A multiplexer of 2ⁿ inputs has n select lines, which are used to select which input line to send to the output. Multiplexers are mainly used to increase the amount of data that can be sent over the network. A multiplexer is also called a data selector.

Conversely, a demultiplexer (or demux) is a device taking a single input signal and selecting one of many data-output-lines, which is connected to the single input.

Multiplexer 8-32 bit integer

Type

Sets the range for in- and outputs

- 8-Bit
- 16-Bit
- 24-Bit
- 32-Bit

Function n to 1

At this function the Multiplexer has 2...8 inputs. The inputs, select and the output are 8-Bit broad (Byte). The inputs may also be constant (0-255).

Select-Input	Output
0	IN0
1	IN1
2	IN2
etc.	...

Note

If the value at select is greater than the number of inputs, nothing happens.

Function 1 to n

At this function the multiplexer has 2 -8 outputs. The outputs, select and the input are 8 bits broad (byte).

Select-input Output

0 OUT0
1 OUT1
2 OUT2

etc...

Note

If the value at select is greater than the number of outputs, nothing happened.

Name

The name of the Multiplexer must be clear. (max. 30 characters) and is automatically assigned by Parsic.

3.23 Digital Input

In computing, **input/output or I/O** is the communication between an information processing system (such as a MCP) and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system, and outputs are the signals or data sent from it. In Parsic 4, a function block contains digital input variables, digital output variables, and his properties.

3.24 Digital output

Read [3.23](#) Digital Input

3.25 EE-Prom



With the **EEProm-Object** it is possible to save the contents of variables in an internal or external EE-Prom. This happens at every positive edge at the input of the object. After a reset, the indicated variables are read in automatically.

At the selection all present variables are displayed in the left list. All variables which you would like to save must be moved into the right list.

Note

Pay attention that 16-Bit, 24-Bit and 32-Bit variable consist of more then one byte.

Example

You would like to save the counter reading of the 16-Bit forwards- and backwards counter **ZV1**.

Then the variables **ZV1** and **ZV1_1** are shown in the left list. Both variables have to be marked and moved into the right list.

Internal EE

If you meet this selection, the internal EE-Prom is used. Not all Microcontroller have an internal EE-Prom.

External EE

If you meet this selection, the external EE-Prom is used, you need the Object [I2C-Base](#) and select the *Device 24CXX* to make the connection.

Note :

At the first start with a "freshly" burned PIC, the EE-Prom contains only \$FF values (255). To avoid malfunction use the Limiter or activate the checksum.

Only one external EE-Object is allowed in a project, even if Parsic allows several.

Saving with checksum

If the values are written into the EE, a 1 Byte checksum is created.

After a reset, the checksum is controlled. If it doesn't match, all variables, which are indicated in the right list, will be initialized with zero.

Check after writing (Verify)

After writing of a byte, it is read once again. If it do not match, the writing process is repeated.

3.26 Encoder

Read [3.21 Dispatcher or Encoder - Decoder](#)

3.27 Edit Device

Connections between objects 8

Note

Connections always start at an object-input or output. The junctions are the core of connections.

Therefore the following connections are possible:

1. Output > junction
2. Junction > junction
3. Input > junction

Making connection

Click with the left mouse button on the object-connection and hold it. A green line appears. You can move this green line with the mouse. Parsic always starts with a horizontal line. With **<shift>** you can switch the direction between horizontal or vertical.

Signals

Each connection becomes a automatically generated signal-name from Parsic.

If it is necessary to give a connection a fixed name, please click with the right mouse button on an arbitrary junction of the connection.

In the following dialog you enter a signal name which doesn't start with an "S".

Examples

for bit oriented connections: -> **S0.3**

for byte and word oriented connections: > **S7**

Cancellation of fixed signal names

- for bit oriented connections > enter a dot for the signal name -> . <-
- for byte and word oriented connections > delete the signal name

3.28 Ground & Vcc



With the **GND** and **Vcc** object it is possible to put unused inputs to a defined level. (e.g. reset at a counter) Conduction's which are connected at this object gets

- **GND = always low**
- **VCC = always high**

3.29 I²C – Base



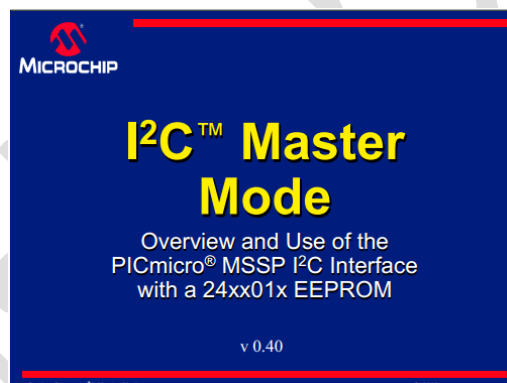
I²C-Base

Also see: [I2C-Data](#) , [Interaction between I2C-Base and I2C-Data](#)

I2C stands for Inter-Integrated Circuit Communications. I2C is implemented in the PICmicro by a hardware module called the Master

Synchronous Serial Port, known as the MSSP module . This module is built into many different PICmicro devices. It allows I2C serial communication between two or more devices at a high speed and communicates with other PICmicro devices and many peripheral IC's on the market today.

For more technical information on the I2C protocol please read Microchip document I²C™ Master Mode on : <http://ww1.microchip.com/downloads/en/devicedoc/i2c.pdf>



This presentation answers some questions about I2C and explains with a full example how to connect a PICmicro MSSP module to an EEPROM

The **I²C-Base** object is the base for accessing the **I²C-bus**. It provides subroutines, for accessing the **I²C-bus**.

[I2C-Data](#) accesses the selected device over **I²C-Base** object.

Name

The name of the object must be clear (max. 30 characters) and is assigned automatically by Parsic.

SDA

This is the data line of the **I²C-bus**. It can connected to any pin. The port-pin is an **Open-Collector** type.

SCL

This is the clock line of the **I²C-bus**. It can also connected to any pin. The port-pin is an **Open-Collector** type.

3.30 I2C – Data

I²C-Data

Also see: [I2C-Base](#) , [Interaction between I2C-Base and I2C-Data](#) , [Edit Device](#)

Over the I²C-Data object, data will be read or written from a *device*. Depending on the used device and the selected function, the ADR connectors, DIN and DOUT are displayed or hidden.

Name

The name of the object must be clear (max. 30 characters) and is assigned automatically by Parsic.

Interface

Here you choose the [I2C-Base](#) object, over which the communication runs.

Device

Here you select the device, which you want to use. An non existing device can also be created by yourself, but you need basic knowledge of the I²C-bus.

Bus-address

This is the address at which the device responds. Allowed are values between 0...7. This address must match the hardware-address (A0...A2).

Function

Here you can choose the function that shall be executed. At every positive edge at the CLK-input, the function will be executed. Depending on the function, corresponding inputs and outputs are displayed in the I²C-Data object.

Interaction between I²C-Base and I²C-Data

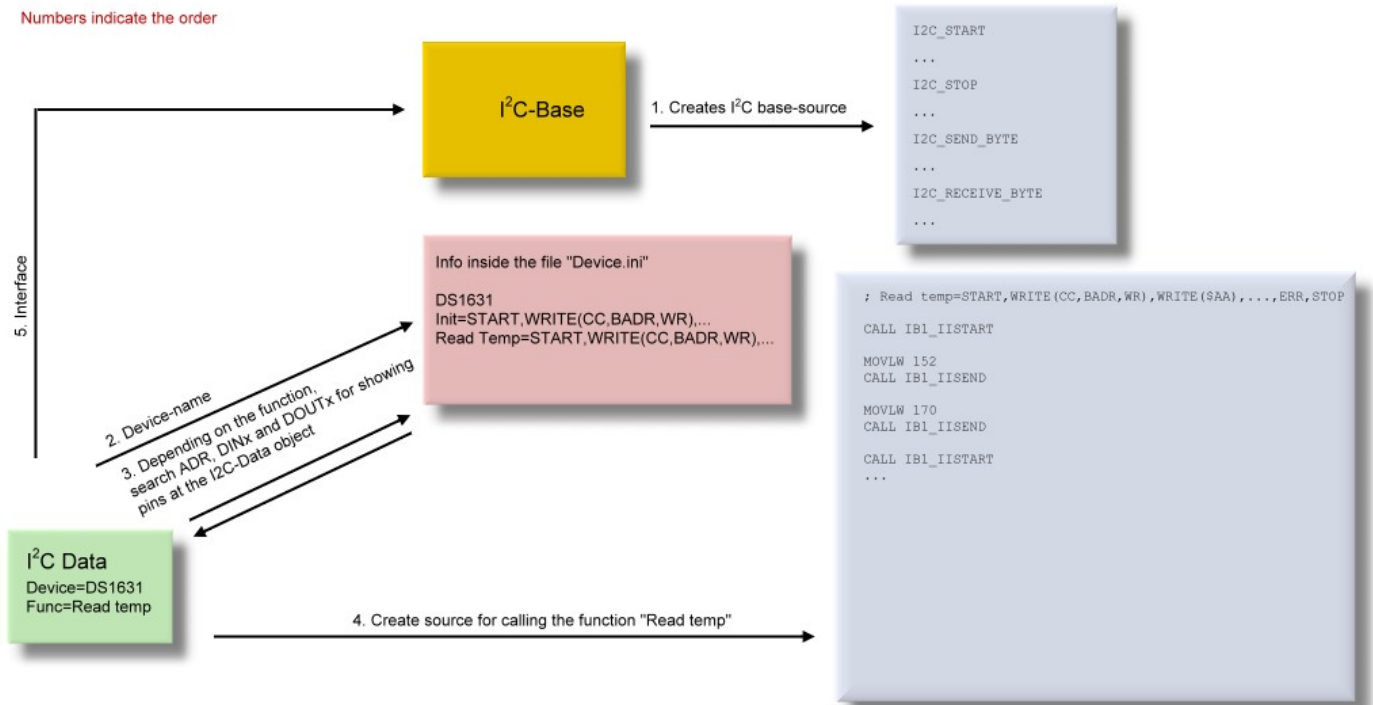
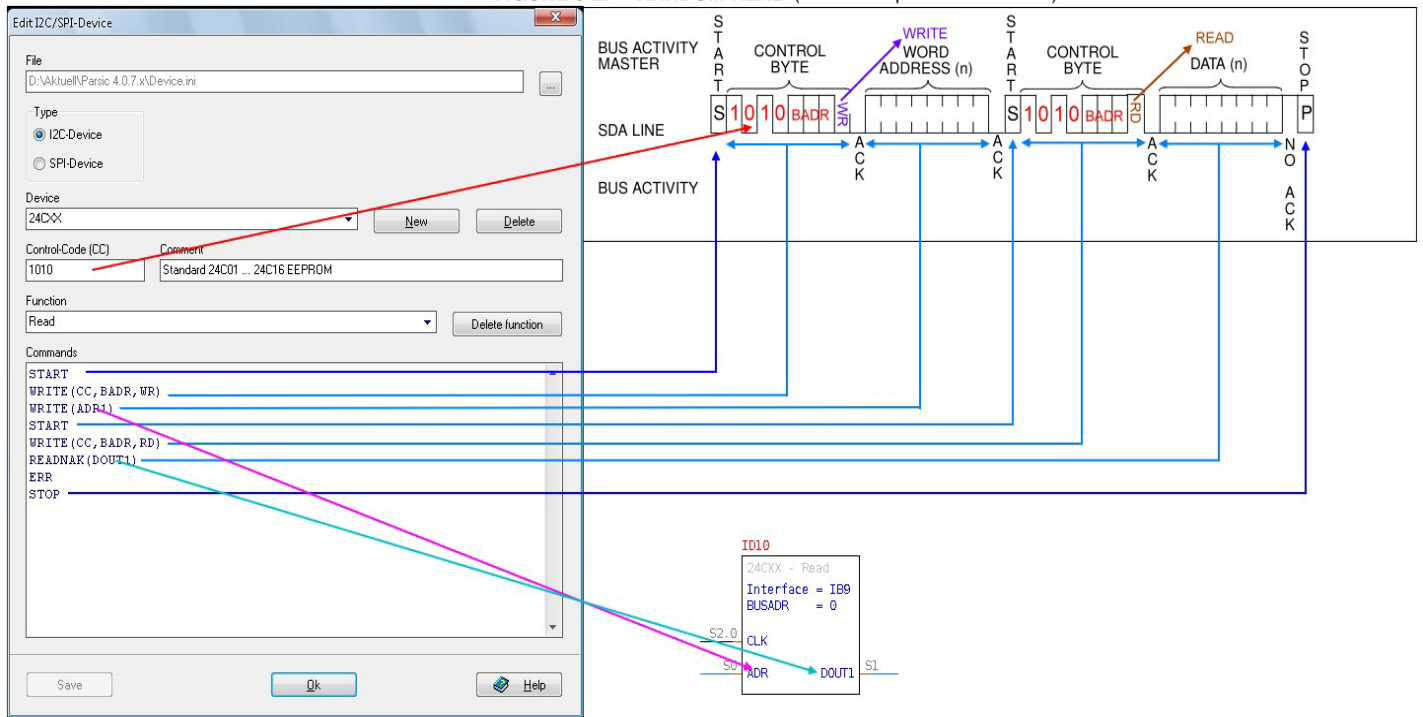


FIGURE 8-2: RANDOM READ (Datasheet piece from 24C02)



3.31 Impulse input

With the impulse input the length of an external impulse is determined with a resolution of 16 bits. For that Parsic uses PORTB change interrupt (RB4..7). The precision depends on several factors:

- 3 The clock frequency
- 4 Number of used timers and Mono-Flops.

The time is decided with help of the internal clock frequency. This corresponds with a resolution of 1 microsecond (20 MHz -> 200 nanoseconds) at a 4 MHz quartz. The length of impulse to be measured must lie between 50 and 65500 times. The time measurement starts with the positive edge. After the negative edge appears, the time is stored in a variable with the name RBxTIME and RBxTIME_HI. In the Info-dialog you can adjust, whether the start of measuring shall start at a positive or negative edge.

Notes

- many timer-objects or Mono-flops reduce the precision
- This object is not supported with all microcontrollers.
- Further any accesses to the PORTB register (RB.X), should be avoided.



3.32 Include

INCLUDE-files are to embed subroutines and assembler source into Parsic. With the right mouse button you can chose, on which place in the created source text the INCLUDE-file will be embedded. There are 5 positions where INCLUDE-Files can be embedded.

1. at beginning of the initialization
2. at the end of the initialization
3. in the main program
4. in the interrupt routine
5. Subroutine
6. Only definitions (no code)

Note

- Embedding in the interrupt-routine, at minimum, one object with times must be available, otherwise the INCLUDE-File will be ignored.
- Parsic V4 has a built-in pre-assembler, so it is not necessary to set bank-switching commands
- User of Parsic V3 have to remove all bank switching commands like "BANKSEL PORTB" from their existing ONCLUDE-files, to use them with Parsic V4

Read following example :

Example

```

;
#define sda portc,4    ; PIN data
#define scl portc,3    ; PIN clock
#define sdadir trisc,4 ; data pin direction
#define scldir trisc,3 ; clock pin direction
;
,*****
; Subroutines for setting SDA & SCL
; high-level = pin is input
; low-level = pin is output and low
,*****
SCLHigh
    bsf scldir    ; input
    goto iideLAY

SCLLow
    bcf scl
    bcf scldir    ; output
    goto iideLAY

SDAHigh
    bsf sdadir    ; input
    goto iideLAY

SDALow
    bcf sda
    bcf sdadir    ; output

,***** I12 Delay *****
I1Delay
    Clrwdt
    return
;

```

3.33

Label



Labels are competent for the connection of the same signal-names on different function plans. Parsic provides a clear name. It is possible for you to change this name.

With the right mouse button you reach the dialog for the settings.

You can choose there

1. whether the label is source or destination
2. assigning the source label to the destination label.
3. change the name

To restore the name, allocated by Parsic, simply enter "P:X".

Renaming

If a **source label** is renamed, all names are changed, also the destination labels.

If a **destination label** is renamed, nothing happens.

If you change the name of the signals (e.g. S0.7-> M2.3) which are connected by source- and destination label, then all signals of this "string" are renamed.

Important

Be careful that the source label will have different name. At present Parsic will not check whether a name already exists

3.34

LCD – Base



All LCD modules, based on the Hitachi HD44780, are supported. To have enough in- and outputs at small PIC's, Parsic use only 6 of 11 control lines.

Connected to port (Rx)

Type in at which output port the LCD module is connected. Several LCD modules can be connected parallel. Type in the same port e.g. **RB**. Only the bits Rx.0...Rx.4 are used. The bits Rx.5...Rx.7 can furthermore be used as normal in- or outputs.

[Connection diagram with 2 LCD modules](#)

Enable output (E)

This connection will activate the display if characters are transferred. Each LDC module needs an own release output e.g. RB.5.

Lines

Put in how many lines the LCD module got. Typical values are 1..4.

Characters per line

Indicate how many characters fit in to a line. Typical values are 8...40.

Process delay

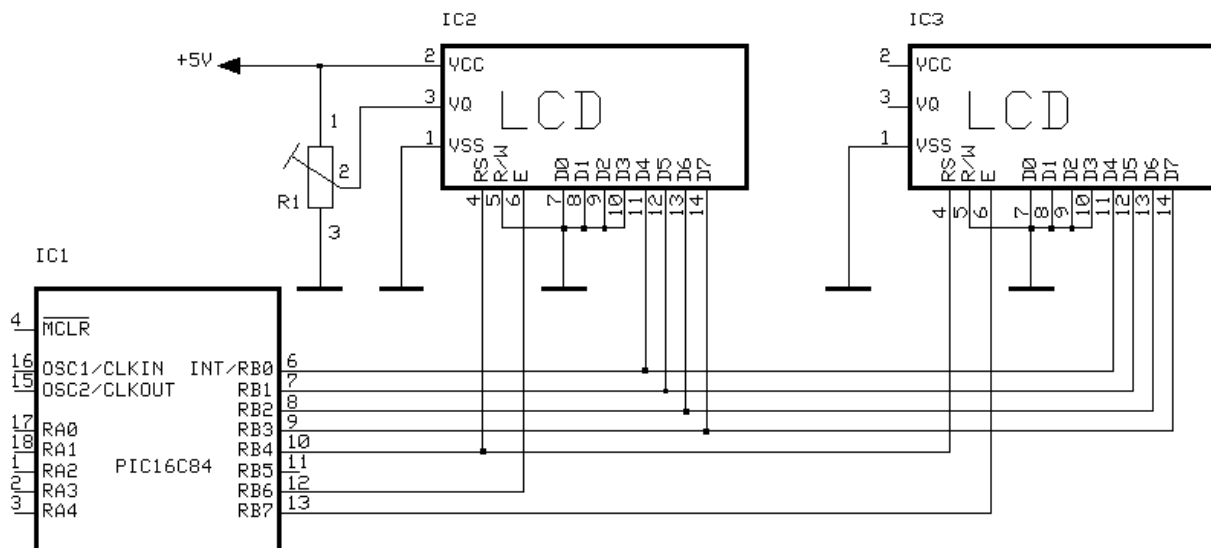
Parsic is waiting app. 50 μ sec after every character till the next character is distributed. It can be possible that some LCD modules are timed fundamentally slower than mentioned in the datasheets (type 270 kHz). If this applies, not readable characters or character on incorrect positions are issued. In a low extent you can determine how long the microcontroller shall wait for the transfer of the next character.

- x 1 = approx. 50 μ Sec. delay
- x 2 = approx. 100 μ Sec. delay
- x 3 = approx. 150 μ Sec. delay

Connection assignment

LCD-Pin	Function	PIC
1	GND	GND
2	Vcc	Vcc
3	VQ	Display contrast, Potentiometer
4	RS	Rx.4
5	R/W	not used, must be tied to GND
6	E	any pin usable
7	D0	not used
8	D1	not used
9	D2	not used
10	D3	not used
11	D4	Rx.0
12	D5	Rx.1
13	D6	Rx.2
14	D7	Rx.4

Connection diagram with 2 LCD modules





With this object texts or variable can be distributed on a LCD module. At every edge at the CF-Input, the text or the variable is shown on the predefined position.

Position

Type in the position on which the distributed data shall appear. The left upper corner has the position 0=X and Y= 0. The typed in values, are allowed to be constant or variable (8-Bit).

Output to

Select a LCD module out of this list, on which the text or the variable shall be displayed. The list is empty, if a no LCD-Module available.

Multi-Text

This attitude makes possible to show more than one text line on the LCD-Module. The position is ignored.

Display text

Enter the text which shall be displayed in the lower editing field. If you leave the editing field empty, a command to delete the display will be created.

Special sign and vowel-mutations must be entered directly as character-code.

At the simulation a '#' is shown instead of a character-code. However, it is possible to indicate a sign which is shown at the simulation. For the production of source text this sign doesn't have any consequence.

Examples:

ß = #E2# or #226# or #226,ß#

ä = #E1# or #225# or #E1,ä#

ö = #EF# or #239#

[Font HD44780A00](#)

Charset HD44780-A00

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Character Code	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000			0	@P`P							-	9	3	α	p	
xxxx0001 (2)		!	1	AQa9							.	7	4	ä	q	
xxxx0010 (3)		"	2	BRbr							「	イ	ツ	×	β	θ
xxxx0011 (4)		#	3	CScs							」	ウ	テ	モ	ε	∞
xxxx0100 (5)		\$	4	DTdt							、	エ	ト	カ	μ	Ω
xxxx0101 (6)		%	5	EUeu							・	オ	ナ	1	σ	Ü
xxxx0110 (7)		&	6	FVfv							ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111 (8)		'	7	GWgw							ア	キ	ヌ	ウ	q	π
xxxx1000 (1)		<	8	HXhx							イ	ク	ネ	リ	7	×
xxxx1001 (2)		>	9	IYiy							ウ	ケ	ル	リ	4	γ
xxxx1010 (3)		*	:	JZjz							エ	コ	ン	レ	j	κ
xxxx1011 (4)		+	;	K[k<							オ	ザ	ヒ	ロ	*	π
xxxx1100 (5)		,	<	L¥ll							カ	シ	フ	ワ	φ	π
xxxx1101 (6)		-	=	M]m>							ユ	ズ	ン	モ	÷	
xxxx1110 (7)		.	>	N^n→							ヨ	セ	ホ	ン	ñ	
xxxx1111 (8)		/	?	O_oe							ツ	ツ	マ	°	ö	■

Note: The user can specify any pattern for character-generator RAM.

Charset HD44780-A02

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A02)

Upper 4 bits Lower 4 bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	▀	0	@	P	`	P	E	α		°	À	θ	à	ä	ë	
xxxx0001	(2)	!	1	A	Q	a	9	A	J	i	±	Á	Ñ	á	ñ	
xxxx0010	(3)	“	”	2	B	R	b	r	W	Γ	φ	²	À	ò	ä	ö
xxxx0011	(4)	”	#	3	C	S	c	s	3	π	ε	³	À	ó	ä	ö
xxxx0100	(5)	⌚	\$	4	D	T	d	t	W	Σ	×	ℜ	À	ô	ä	ö
xxxx0101	(6)	₯	%	5	E	U	e	u	Ÿ	σ	¥	℥	À	õ	ä	ö
xxxx0110	(7)	●	&	6	F	V	f	v	J	J	!	9	€	ö	ø	
xxxx0111	(8)	¢	'	7	G	W	g	w	Π	τ	§	•	Ç	×	ç	÷
xxxx1000	(1)	↑	<	8	H	X	h	x	Y	‡	¶	ω	È	ϕ	è	ϕ
xxxx1001	(2)	↓	>	9	I	Y	i	y	U	Ø	¹	É	Ù	é	ù	
xxxx1010	(3)	→	*	:	J	Z	j	z	4	Q	³	É	Ú	é	ú	
xxxx1011	(4)	←	+	;	K	[k	[W	δ	«	»	È	Û	è	Û
xxxx1100	(5)	£	,	<	L	\	l		W	≈	⊞	⊞	Ë	Ü	ë	Ü
xxxx1101	(6)	≥	-	=	M]	m]	b	⊙	⊙	⊙	Ë	Ý	ë	Ý
xxxx1110	(7)	▲	.	>	N	^	n	^	W	ε	⊞	⊞	Ë	Þ	ë	Þ
xxxx1111	(8)	▼	/	?	O	_	o	_	⊞	⊞	⊞	⊞	Ë	ß	ë	ß

Display variable

In the function plan an additional connection is displayed in the object. At this the signal is connected whose value shall be reported. The alignment in the display is align – right .

Display decimal

If it is a 8-Bit variable, then a 3-digit decimal value is displayed.
 If it is a 16 bit variable, then a 5-digit decimal value is displayed.
 Under > leading zeros you can vote, whether leading zeros shall be shown or replaced by blanks.
 If **constant length** is assigned, then the issue is limited on the set length.

Example

Length = 2
 If the variable at the input contains a 123, then only 23 are displayed.
 If the set length is larger than 3 (8-Bit variable) or 5 (16-Bit variable), then the length is ignored.

Display hexadecimal

If it is a 8-Bit variable, then a 2-digit hexadecimal value is displayed.
 If it is a 16-Bit variable, then a 4-digit hexadecimal value is displayed.
 If it is a 24-Bit variable, then a 6-digit hexadecimal value is displayed.
 If it is a 32-Bit variable, then a 8-digit hexadecimal value is displayed.

Display binary

If it is a 8-Bit variable, then a 8-digit binary value is displayed.
 If it is a 16-Bit variable, then a 16-digit binary value is displayed.
 If it is a 24-Bit variable, then a 24-digit binary value is displayed.
 If it is a 32-Bit variable, then a 32-digit binary value is displayed.

Under **leading zeros** you can choose, whether leading zeros are shown or replaced by spaces

3.36 Limiter



The limiter 8-32 bit integer, checks the contents of the input variables with the upper and lower limit.

- If the input > upper limit, then output = upper limit
- If the input < lower limit, then output = lower limit

Otherwise

output = input.

Note

For upper and lower limit constants and/or variables are allowed.

Error outputs

The Limiter has 2 error outputs.

- If the upper limit is exceeded, then the upper output goes high
- If the lower limit is fallen below, then the lower output goes high

Note

Using of these outputs is optional. If nothing is connected at them, then Parsic doesn't create any source text for these outputs.

Type

8 bit:

The input is treated as **8-bit** variable. This applies to the upper and lower limit as well, if they are indicated as variables.

16 bit:

The input is treated as **16-bit** variable. This applies to the upper and lower limit as well, if they are indicated as variables.

3.37 Logical AND operator



A bitwise AND takes two binary representations of equal length and performs the logical AND operation on each pair of corresponding bits. The result in each position is 1 if the first bit is 1 and the second bit is 1; otherwise, the result is 0. In this, we perform the multiplication of two bits.

For example $1 \times 0 = 0$ and $1 \times 1 = 1$

A gate has up to **16 inputs**. All inputs and the output can get inverted.

Inputs Integer, 1 to 16 number of logical inputs
Output Result of logical AND operation
Inverted Inputs or Output Negated result of logical AND operation (NAND operation)

Note

At only one input the output gets inverted automatically => Inverter.

Name

The name of the gate must be clear, max 30 characters and Parsic is allocating automatically.

3.38 Logical NOT operator

The **bitwise NOT**, or **complement**, is a unary operation that performs logical negation on each bit, forming the ones' complement of the given binary value. Bits that are 0 become 1, and those that are 1 become 0.

Input Input signal
Output Negated input signal

Name

The name of the gate must be clear, max 30 characters and Parsic is allocating automatically.

3.39 Logical OR operator

A **bitwise OR** takes two bit patterns of equal length and performs the logical inclusive OR operation on each pair of corresponding bits. The result in each position is 1 if the first bit is 1 or the second bit is 1 or both bits are 1; otherwise, the result is 0.

Inputs Integer, 2 to 16 number of logical inputs
Output Result of logical OR operation
Inverted Inputs or Output Negated result of logical OR operation (NOR operation)

Name

The name of the gate must be clear, max 30 characters and Parsic is allocating automatically.

3.40 Logical XOR operator

A **bitwise XOR** takes two bit patterns of equal length and performs the logical exclusive OR operation on each pair of corresponding bits. The result in each position is 1 if only the first bit is 1 or only the second bit is 1, but will be 0 if both are 0 or both are 1. In this we perform the comparison of two bits, being 1 if the two bits are different, and 0 if they are the same.

Inputs Integer, 2 to 16 number of logical inputs
Output Result of logical XOR operation
Inverter Inputs or Outputs Negated result of logical XOR operation (XNOR operation)

Name

The name of the gate must be clear, max 30 characters and Parsic is allocating automatically.

Input 1	Input 2	AND-Gate	OR-Gate	Ex-Or-Gate
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

3.41 Macro



In a macro, multiple objects can be combined into one object.

Saved macros can be added to the function-plan with **File > Load macro....**

If you click with the right mouse-button on a macro, a function-plan will be opened with the title "**Macro**".

It behaves like a normal function-plan. In addition, the **Toolbar > Macro** will show two more objects. These objects are needed to add inputs and outputs to the macro.

Each macro can be added a maximum of 16 inputs and 16 outputs.

As long as a macro is opened, it can be saved by choosing **File > Save macro ...** for other projects.

Notes

- Inside a macro, labels not allowed.
- Macros can't be nested. (macro inside a macro)
- Do not use fixed names for objects and signals inside a macro. There are problems, if this macro is added more than once in a project.

Macro Input



The macro-input is the interface between the function-plan and the objects within a macro.

Clicking with the right mouse button on the **macro-input**, opens the connection-info dialog. Here you can change the type, the description and name.

Type

The type of the **macro-input** must match the type of the **object-input** to which it is connected inside the macro.

The following types are available

- Bit
- 8-Bit
- 24-Bit
- 16-Bit
- 32-Bit

Example

The macro input is directly connected with the input of an **OR-Gate**.

The type of the OR-Gate input is **Bit**. The **macro-input** must then also set to **bit**.

Description

Here you can enter a short description of the connection.

The length is limited to max. 6 chars.

Macro output



The macro-output is analogous to the macro-input, with the difference that he is connected with the object-outputs.

3.42 Memory – Set/Reset latch

In electronics, a flip-flop or latch is a circuit that has two stable states and can be used to store state information. Flip-flops and latches are used as data storage elements. Such data storage can be used for storage of state, and such a circuit is described as sequential logic. When used in a finite-state machine, the output and next state depend not only on its current input, but also on its current state (and hence, previous inputs). It can also be used for counting of pulses, and for synchronizing variably-timed input signals to some reference timing signal.

Condition for setting

- S = High
- R = Low

Condition for reset

- R = High
- S = High or Low

Note

Inverted in- or outputs turn over the function.

Priority

The priority determines what happens at the output when both inputs are high level.

- R-input = the output will be set to low
- S-output = the output will be set to high

Name

The name of the RS-Flip-Flop must be clear. (max. 30 characters) and Parsic is allocating automatically.

3.43 Multiplexer

Read [3.22 De-multiplexer](#)

3.44 PWM output Impulse - Output

Impulse-Output

With the impulse-output, impulses can be created with a precision of 16-Bit. To reach the precision, the remaining program is stopped during the issue. Only one impulse is created, if the EN-Input is high (not inverted).

The time is decided with assistance of the internal clock frequency. This corresponds to a resolution of 1 microseconds (20 MHz = 200 nanoseconds) at a 4 MHz.

The length of the impulse must be between 50 and 65500 times.

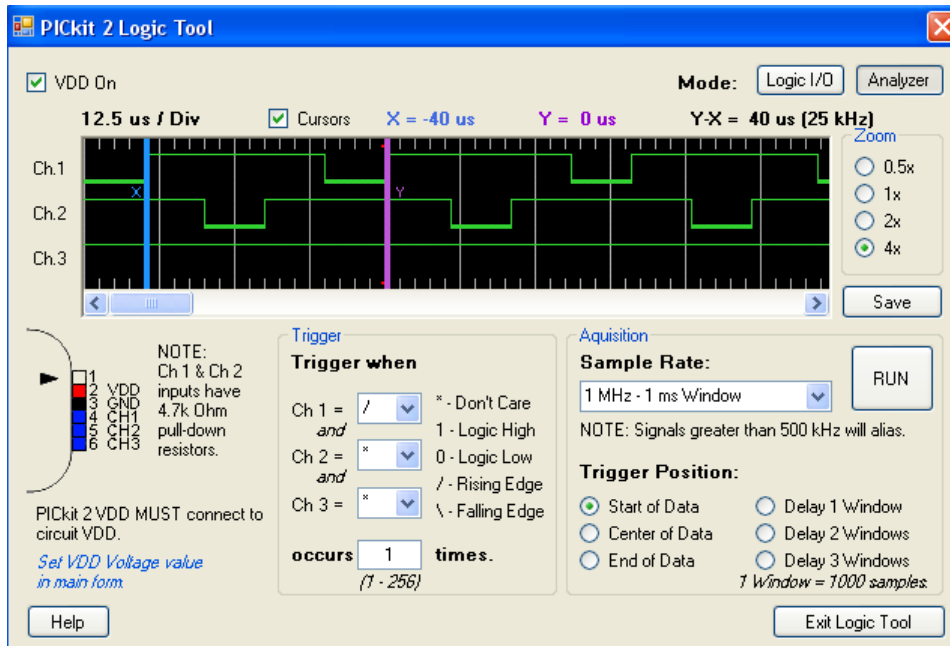
PWM

Pulse-width modulation (PWM), or **pulse-duration modulation (PDM)**, is a modulation technique that conforms the width of the pulse, formally the pulse duration, based on modulator signal information. Although this modulation technique can be used to encode information for transmission, its main use is to allow the control of the power supplied to electrical devices, especially to inertial loads such as motors. In addition, PWM is one of the two principal algorithms used in photovoltaic solar battery chargers, the other being MPPT (Maximum power point tracking).

This object uses the hardware PWM and therefore it is not available at all PIC's.

You can use this module to create analogue outputs.

The output creates a constant output frequency (approx. 4 KHz at 4 MHz clock frequency). The pulse-break- relationship can be changed. At each positive edge at the CF-Input the value at the upper input is written into duty-cycle-register. The value must lie between 0 and 1023.



PICKit 2 Logic Tool

3.45 Schmitt Trigger



The inputs work with 8-bit up to 32-bit resolution.
For SP and HY input applies:

- The values may be constant or variable
- The IN input must be variable.
- It is possible to invert the function of the output.

Function

The signal to be measured is given on the IN input.
The SP input defines the upper switching point.
The HY input determines the hysteresis.
The lower switching point is calculated out of SP - HY input.

Example

SP = 100
HY = 20

The output get high at values ≥ 100 and at values $< (100-20)$ it get low.

Note

The hysteresis may never be greater than the switching point.

Name

The name of the Schmitt-Trigger must be clear. (max. 30 characters) and Parsic is allocating automatically.

3.46 Shift register



Every shift-register is a 1 bytes large variable. At every positive impulse at the CLK-input, the data starting at Bit-0 are taken over into the shift register. The output (max. 8) can be distributed to various bits. Source text is only created for outputs which have a signal name.

Notes

- All in- or outputs can get inverted.
- Priority has the Reset-Input.

Name

The name of the shift-register must be unique (max. 30 chars) and is assigned automatically by Parsic.

3.47 Sleep



With this object it is possible to move the microcontroller into the sleep mode. If the IN-Input lies on high level, then the sleep mode is activated, the program is stopped and the oscillator is switched off. To restore the normal function 2 possibilities are available.

1. a positive edge at the RB0-Input
2. an amendment of level at RB4, RB5, RB6 ... RB7

If the IN-Input remains on high level, the program is passed through one-time and the sleep mode is active, again.

Note

- To be able to use this function, the Watch-Dog Timer may not be activated. It would trigger a reset after a particular time.

World's Lowest Sleep Current MCU

MICROCHIP nanoWatt XLP

As more electronic applications require low power or battery power, energy conservation becomes paramount. To enable applications like these, PIC® microcontrollers with nanoWatt XLP Technology offer the industry's lowest currents for Sleep, where extreme low power applications spend 90%-99% of their time. Benefits of nanoWatt XLP Technology include: Sleep currents down to 20 nA, Brown-out reset down to 45 nA, Watch-dog timer down to 400 nA and Real-time clock/calendar down to 500 nA.

www.microchip.com/XLP

0901294A

3.48 SPI – Base



Also see: **SPI-Data**

The **SPI-Base** object is the base for accessing the **SPI-bus**. It provides subroutines, for accessing the **SPI-bus**. The **SPI-Data** object accesses the selected device over **SPI-Base** object.

Name

The name of the object must be clear (max. 30 characters) and is assigned automatically by Parsic.

SPI Mode

Also see: **SPI-Modes**

The mode defines how the data will be transmitted.
The following applies to the CPOL

- For CPOL = 0, than the idle-state of the clock-pin is **low**
- For CPOL = 1, than the idle-state of the clock-pin is **high**
- The following applies to the CPHA
- For CPHA = 0, the data will be sampled at the first clock-edge
- For CPHA = 1, the data will be sampled at the second clock-edge

If CPHA = 0: The slave and master puts the first data on the bus, when **chip-select** will be activated.
If CPHA = 1: The slave and master puts the first data on the bus, after the first clock-edge.

SDO

This is the data output. It can connected to any pin of the PIC.

SDI

This is the data input. It can connected to any pin of the PIC.

SCK

This is the clock output. It can also connected to any pin of the PIC.

3.49 SPI – Data

Also see: **3.27 Edit Device**

Over the **SPI-data** object, data will be read or written from a *device*. Depending on the used device and the selected function, the **ADR** connectors, **DIN** and **DOUT** are displayed or hidden.

Name

The name of the object must be clear (max. 30 characters) and is assigned automatically by Parsic.

Interface

Here you choose the **SPI-Base** object, over which the communication runs.

Device

Here you select the device, which you want to use. An non existing device can also be **created** by yourself.

Bus-address

This is the address at which the device responds. Allowed are values between 0...7. This address must match the hardware-address (A0...A2).

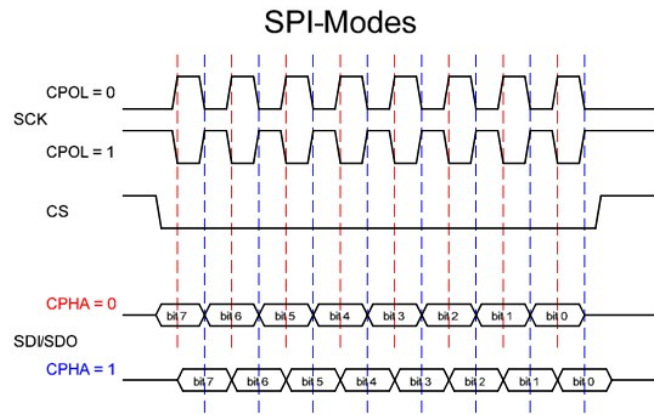
Function

Here you can choose the function that shall be executed. At every positive edge at the **CLK-input**, the function will be executed. Depending on the function, corresponding inputs and outputs are displayed in the **SPI-Data** object.

Chip-Select

This is the **Chip-Select** output. It can connected to any pin of the PIC.

Motorola-Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



For more technical information on SPI Interface please read PDF Microchip document :



<http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf>



SPI Commands

Also see: **Constants**

START

The device will be disabled (CS-high) and then activated (CS-low)

This command is useful within a function, to read or write to different register-addresses.

WRITE(<DATA>|<DINx>)

Sends one byte <data> over the bus.

READ(<DOUtx>)

Receives one byte <data> from the bus.

Example

```
WRITE(CC,BADR,WR) ; Sends one byte, formed from the Control-Code, bus-address and WR (next is a WRITE)
WRITE(ADR1) ; Sends the lower byte from the connection ADR of the object
WRITE(DIN2) ; Write data from input DIN2
WRITE($AA) ; Sends $AA (decimal 170) over the bus
WRITE(255) ; Sends 255 (Hexadecimal $FF) over the bus
START ; new command begins
WRITE(CC,BADR,RD) ; Sends one byte, formed from the Control-Code, bus-address and RD (next is a READ)
READ(DOUT1) ; Reads one byte and write it to the output DOUT1
```

SPI-Constants 212

Also see: **SPI-Commands**

Hint

- The constants **CC**, **BADR**, **RD** and **WR** are normally for devices, which have separate versions for the I2C-Bus and for the SPI-Bus eg. MCP23017 (I2C), MCP23S17 (SPI)

Control Code (CC)

The Control-Code is part of the Slave-Address (Bit 7...4).

Each *Device* has his own Control-Code. He is pretended from the manufacturer and must taken from the data-sheet.

An EEPROM 24Cxx has e.g. the control code 1010 (binary).

Bus-Address (BADR)

The bus-address is the second part of the address, (Bit 3...1), under which the device will be addressed. It is required if several identical devices are driven at one bus.

The bus-address is led out by many devices in hardware.

RD and WR

These two constants decide whether the **next** command is a read or write command.

Summary

Bit	Bit 7...4	Bit 3...1	Bit 0
Constant	CC	BADR	RD or WR
Example (binary)	1010	111	1 or 0
.	Slave-Address		read or write

ADR1, ADR2, DIN1...DIN16, DOUT1...16, ERR

These constants are place-holders for the connections of the **SPI-Data** object.

The assignment is following:

ADR1

Is the **least significant** byte from the connection **ADR** of the object **SPI-Data**

ADR2

Is the **most significant** byte from the connection **ADR** of the object **SPI-Data**

DIN1...DIN16

This is are placeholders for the inputs of the I2C data object. They can only used with the **WRITE** command.

DOUT1...DOUT16

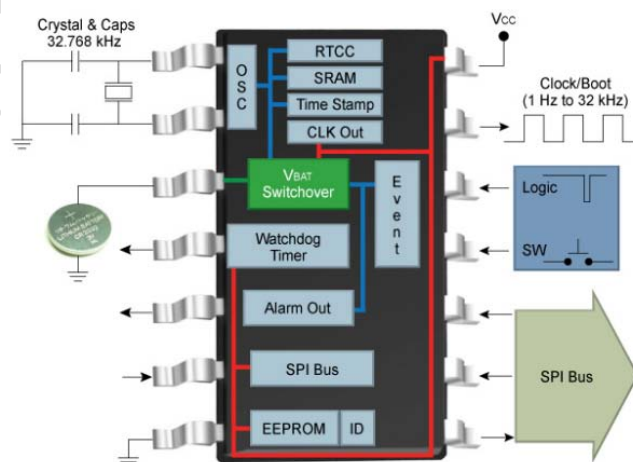
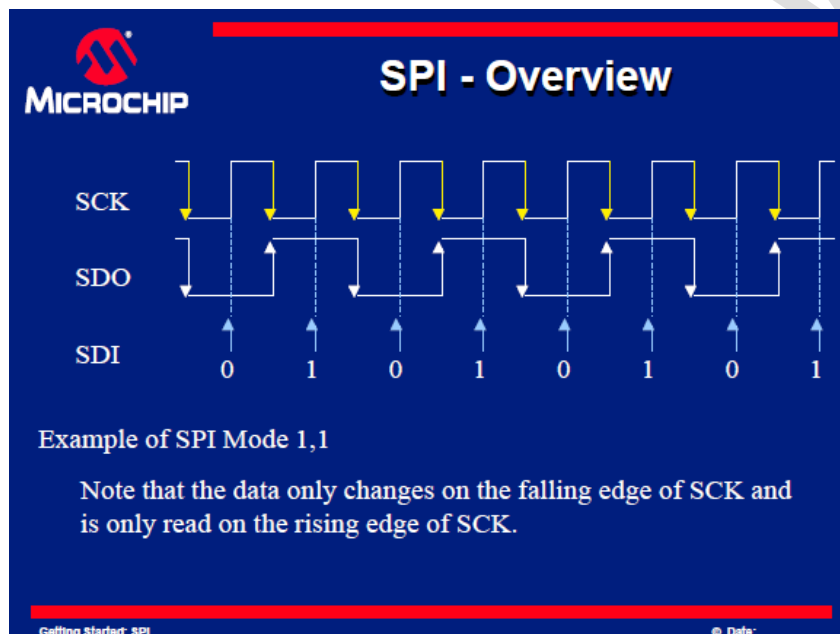
This is are placeholders for the outputs of the I2C data object. They can only used with the **READNAK** and **READACK** command.

ERR

Internal it is only a label, to which is jump, if the command **WRITE** isn't confirmed by an **<ACK>**.

Notes

- The constants **ADR1**, **DINx** and **DOUTx** decide whether the corresponding connections are inserted into the **SPI-Data** object or not.
- Using the constants **ADR2**, the corresponding connection changes to **16-Bit** (otherwise **8-Bit**)



3.50 Table Editor

1	98
2	17

Also see 3.10 Call

The table stores data inside the ROM. The access on the table always takes place over the CALL-Objekt. Tables always created as a subroutine inside the ROM.

Type 8-bit

Values with a range of 0...255 are stored in the table. The length of the table may lie between 1 and 2048.

Type 16-bit

Values with a range of 0...65535 are stored in the table. The length of the table may lie between 1 and 2048. Each value is stored in the order low-byte, high-byte.

Processing table

1. Click with the right mouse button into the table-object.
2. Give the table a clear name.
3. Define the length.
4. Enter the values. With the right scroll bar you can reach the remaining values. The counting always starts at 0
5. During entering the values, you are able to vote between different formats, independently to what you have chosen under <display>.

Display

Display	Input
decimal	all numbers between 0...9 are allowed
binary	with preceding % the numbers 0 and 1 are allowed. High-order zeros be able to be left out.
ASC	with preceding ' one arbitrary ASC character is allowed.
Hexadecimal	with preceding \$ the characters 0...9 and A...F are allowed

Save

The contents of the table are saved in the ASCII-format. One value is written in each line.

Load

The contents of the table came out of an ASCII-file. A value have to be in each line. Blank lines are ignored. If the type is 8 bits, then higher values are cut off to 8 bits.

Application examples

- 7 segment decoder
- Linearization of analogue values

3.51 Timing Falling edge - One Shot



This object generates an impulse for one program pass.

Function

At every positive edge at the input, the output is setted.
Independently of the status of the Input, the output is reseted at the next pass.
The input and output also can get inverted.

3.52 Timing Pulse timer - Mono Flop On Delay Off Delay



Depending on input allocation, the Mono-Flop get triggered by a positive or negative edge.
After a click with the right mouse button a dialog for entering appears

1. Pulse duration in milliseconds or by detail of a variable
2. Retriggerable or not
3. The name
4. Mono-Flop or Delay
5. Inverted in- or outputs turn over the function.

On-Delay

In that moment where the input is going high, the predefined time starts to run. After process of the time, the output is getting high.
If the input gets low before process of the time, the time will be reseted.

Off-Delay

In the moment where the input sets high, the output sets high, also. If the input goes low, the predefined time starts to run.
After process of the time the output is getting low. If the input gets on High before process of time, the time is reseted.

Name

The name of the Mono-Flop must be clear. (max. 30 characters) and Parsic is allocating automatically.

Note

If you indicate a 8 bit variable for the pulse duration, then the pulse duration is calculated as follows:
(value of the variables (0 -255) x 250)+program transmission delay of the complete program.

Example:

You have entered the variable ZV1. It describes a counter (ZV1).
This counter has the value 16 at the moment.
Then the Mono-Flop got a pulse duration (at present) of:
 $16 \times 256 = 4096 \text{ times} = 4096 \mu\text{S at } 4 \text{ MHz} + \text{program transmission delay.}$

3.53 UART – Base



Universal Asynchronous Receiver Transmitter

With this object you define the settings of the serial interface. Not all microcontrollers have a built-in UART. If you use a PIC without hardware UART, an error message is created.

See **Connection example RS232**

See **Connection example RS485**

Baud rate

Here you can select the speed, with which the data shall be transferred.

Depending on the frequency of the PIC, the resulting Baudrate deviates more or less from the selected Baudrate. The indicated error should not exceed 3%. The format is always **8N1** (8-Bit data, no parity, 1 stop bit)

Tx-pin

You have to indicate at which Pin the data shall be distributed. If you use UART hardware it is a quite defined connection (type RC.6).

Rx-pin

Here you must say at which pins the data are fed. If you use UART hardware it is a quite defined connection (type RC.7).

DIR-Pin (only half-duplex)

If a RS485-interface is used, this connection switches the data direction. This output has low level (reception). If data shall be transferred, this output is set to high level (send) until the end of the transmission. This happens without consideration whether any object just shall receive data. With help of the active outputs (UART-Data-Objects), you can influence this process.

With checksum

Activated, a 1 byte checksum is always appended at the end of the transmitted data.

Reception only:

The received checksum is compared with the checksum internally calculated. Only at agreement, the data of the input buffer are transferred into the variables which you indicated in the right list.

The checksum is calculated as follows (CS = checksum):

1. CS = 170 (HEX = \$AA)
2. CS = CS + 1st byte
3. CS = CS + 2nd byte
4. etc..

Example

The 16-Bit counter presently got the value 1000. Then in the right list stands:

ZV5 (contains at present a 232 (\$ E8) = low-byte)

ZV5_HI (contains at present a 3) = high byte)

(3 x 256 + 232 = 1000)

CS = 170 + 232 + 3 = 405 = cut off to the lower 8-Bit = 149

The following byte sequence will be transferred:

<232><3><149>



WI-FI MRF series



MCP2200 USB-to UART serial converter



Commercial product RS232 serial TTL to Ethernet converter TCP IP module

3.54 UART – Data



With this object data are sent or received about the serial interface.

Active-Output (ACT)

For both function applies:

- Transferring = The ACT-Output remains on high level, until the data is transferred.
- Receiving = The ACT-Output remains on high level, until the data are correctly received or an error is occurred.
- Notes
- With assistance of this output you can prevent that two objects will receive resp. transmit data at the same time.
- Parsic uses this output to control the data flow. Well, minimum one interconnecting line must be connected, even if you don't want to use this function.

Function - transfer

All presently created variables are shown in the left list (in the RAM).

Slide all variables which shall be transferred into the right list. The succession in the right list also determines the succession at transmission. At a positive edge at the Enable-Input (EN) the output buffer is filled with the indicated variable (+ checksum) and authorized for transmission. Every further edges don't have any consequence, until the transmitting has been finished.

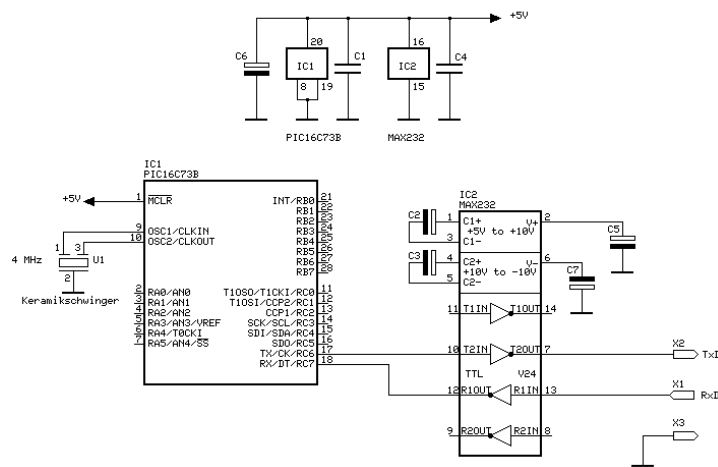
Function - receive

In the left list (in the RAM) all presently indicated variables are shown. Slide all variables into the right list, which shall get the received data. At a positive edge at the Enable-Input, the input buffer is initialized and authorized for reception. Only such data as indicated in the right list will be received (every line is 1-Byte). Every further edge doesn't have any consequence till the reception has been ended.

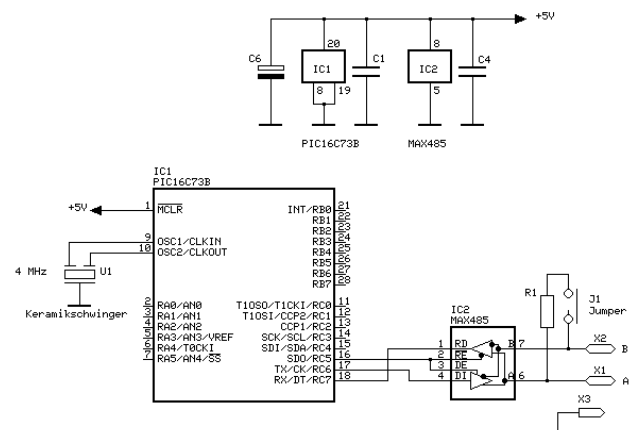
Note

If errors occurs during reception of data or checksum (if activated), then the reception gets interrupted and the ACT-Output is set low.

Connection diagram RS232



Connection diagram RS485



Bibliography

Arethusa Parsic Card Series
Hitachi HD44780 LCD Controller
Microchip Technology's Academic Program
Microchip Technical Training & Education
Microchip MPLAB – PICkit2 – PICkit3
Microchip MPLAB X Integrated Development Environment (IDE)
Microchip MCP2200
Microchip PICDevice Configuration
Microchip SPI™ Overview and Use
Philips I²C Protocol
Wikipedia free encyclopedia

Visual Parsic V4

Copyright

This user manual is a copyright **Swen Gosch** and **Parsic Italia**
Microchip, Windows, PICkit2, PICkit3, MPLAB, MCP2200, Windows, Visual Parsic V4, Parsic 4 and others are internationally registered Trademarks.