WIZ-C

Rapid Application development for the PIC Microcontroller

Introductory Manual

WIZ-C is well featured development environment for Windows 9x, ME, 2000, XP and NT.
The program, and its support files and example programs are
© Copyright Robin Abbott, 2000,2010. <robin.abbott@fored.co.uk>.

The program may be installed onto the hard disk of only one Personal Computer, and must removed by deleting the executable file, and all the support files before installing onto a different computer.

Forest Electronic Developments
12 Buldowne Walk
Sway
HAMPSHIRE
SO41 6DU
Sales : +44 - (0)1590 - 681511

info@fored.co.uk

Or see the **Forest Electronic Developments** home page on the world wide web at the following URL:

http:\\www.fored.co.uk

Introduction

Welcome to WIZ-C, the second of Forest Electronic Developments' series of rapid application development environments following on from WIZ-ASM which is based on PIC Assembler. WIZ-C was previously known as PIXIE – the change in name is for copyright reasons.

It allows you to select software library components (*elements*) from a palette, set their parameters by drop down lists, check boxes and verified text entry. It will generate the main application, initialisation code and main loop automatically and considerably speeds the front end development of PIC projects.

WIZ-C includes the full functionality of the FED PIC C Compiler, and PICDESIM - the FED development and simulation environment. WIZ-C can be configured not to use the application designer in which case it behaves exactly like the FED PIC C Compiler. WIZ-C can also load FED PIC C Compiler projects and handles them identically to the compiler.

This document is split into three, this first part includes a simple tutorial which shows how easy the system is to use. The second part is a full PICDESIM tutorial, and the third part is the WIZ-C reference.

C

It is expected that users of WIZ-C will be familiar with the C programming language. The WIZ-C CD ROM is supplied with an introductory manual to C "Learn to Use C with FED", it is recommended that this is read if unfamiliar with C.

Installation

The program is installed from CD-ROM. For the CD-ROM insert into the CD drive and an opening menu should come up. Alternatively run the program "SETUP.EXE" from the CD.

It is strongly suggested that (at least initially) the program is installed in the default directory which will allow the example projects to operate correctly.

The manual is supplied as an Adobe Acrobat (PDF) Format file, a copy of Acrobat is supplied on CD-ROM and can be installed from the opening menu. The manual is duplicated in the help files which are accessible under the Help menu.

Running the program

Following installation there will be a new program group (or menu in Windows '95), called WIZ-C. Double click the icon titled "WIZ-C" to start the program. The program will start and open an example project. You may find that the project doesn't fit on the screen so you can use the menu option **Window | Arrange for Edit** to make it fit.

At this point you are recommended to run through the example projects below before migrating any of your existing programs to WIZ-C.

Using WIZ-C without the front end as a standard C Compiler

The C Compiler used by WIZ-C is identical to the standard FED PIC C compiler. To use it as a standard compiler then turn off the application designer using the *Project* | *Use Application Designer* menu option. It may be turned back on at will. The tutorials in the C Compiler manual may now be followed.

Once turned off the there will be no updates to any of the files included in the project. However BE WARNED that if the designer is turned back on and the application generated again then all changes to the _Auto.h, _Main.c, and _Lib.c files will be lost.

FED recommend that the C Compiler reference should be read in association with this manual.

Example Project #1 - Switches, LED's and a serial interface.

In the example project we will look at the development and simulation of a complete program using WIZ-C.

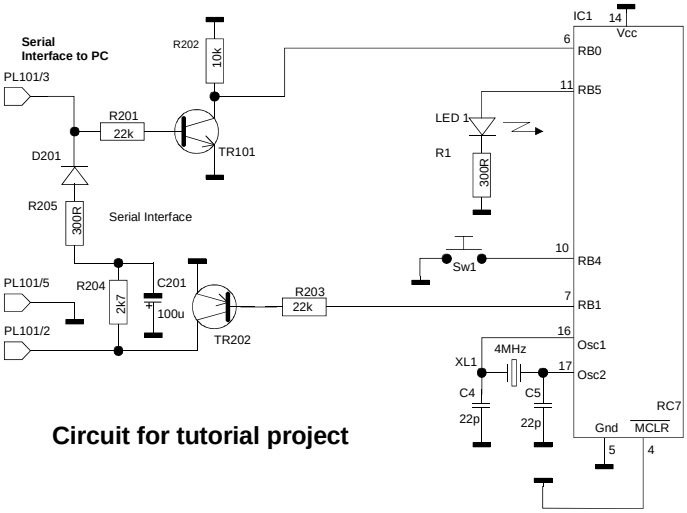
The program we will look at is designed for a 16F88 processor. The application will undertake the following simple functionality:

1) It will have a serial interface which may be connected to a standard PC using a 9 pin socket (PL101), this will use the 16F88 serial port.
2) It will have a push button switch
3) It will have an LED

When a byte is received on the serial interface the LED will illuminate. Now when the push button is pressed the received byte will be sent back on the serial interface and the LED will be extinguished.

We will simulate this device using the real device simulation capability of WIZ-C.

The circuit diagram is shown below:

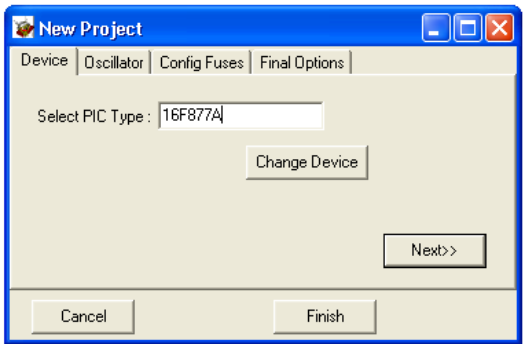


Circuit for tutorial project

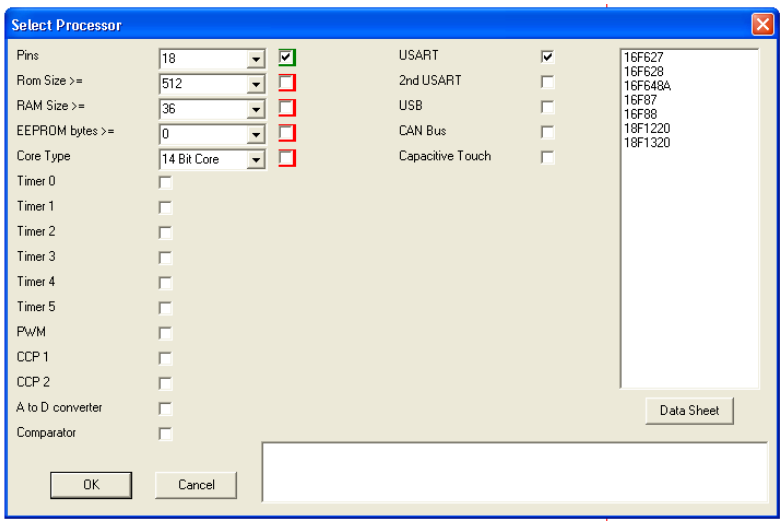
Note that by default WIZ-C will set Port B pull ups to enabled so there is no need to use a pull up on the switch input.

Opening a new project

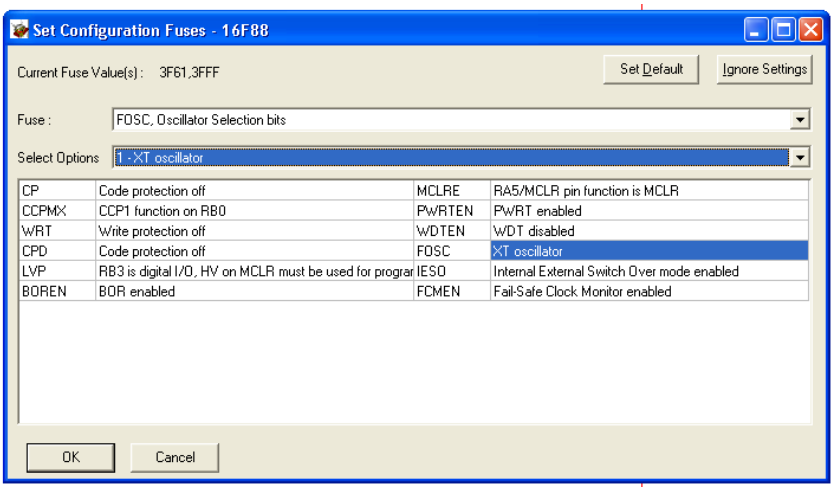
To open the new project then use the *Project | New Project or Project Group...* menu item. This brings up the new project wizard :



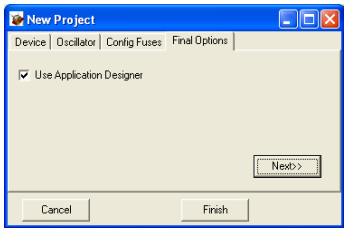
The default device is a 16F877, but we want to change that so click the Change Device button, a new dialog will be brought up – this is the Processor wizard, with this you can select the features you want – number of pins, ROM and RAM size and processor features. You can experiment with this – change the number of pins to 18, click the USART button and then you will have the following options :



Now select the 16F88 device. Click the OK and then the next button. This will bring up a number of standard oscillator frequencies – select 4MHz. Click Next, this brings up a button allowing the configuration fuses to be set, click the button to see the options. This is how we set it – note the watchdog is disabled, an XT oscialliator selected and the power up timer selected.



Click OK and then Next to display the final options. By default we would like to use the Application Designer so leave this option checked :



Finally press Next for the last time (or Finish).

A File dialog box will be brought up. Select a directory in which to save it for XP we used : “C:\Program Files\FED\PIXIE\Projects\Tutor”. For Vista you should use a directory in your documents structure. If the directory does not exist create it using the New Folder button on the Open Dialog. Enter the file name "Tutor", click Open. As this is a new project it may appear in minimal form, maximise the window by using the button in the top right of the window bar and then use the **Window | Arrange for Edit** option to position the new project on screen (you could also press ALT+E).

The top window will be the application designer. Check that the PIC type is 16F88 and ensure that the oscillator frequency shows 4000000 which is 4MHz. Alternate PIC types can be selected by clicking the button labelled "Change Device" (under the element groups), the oscillator frequency may be selected from a list, or the exact value may be typed into the box. The oscillator frequency is entered in Hertz.

Look at the application designer window. The application designer holds software elements in groups at the top of the window :

A software *Element* is a library subroutine, or software component, which may be used within an application. The application designer allows software elements to be selected for use within the current application. The software elements are grouped by type. The element is the small square icon.

The application designer can be shown or hidden using the Show button on the tool bar :

Using elements within the application

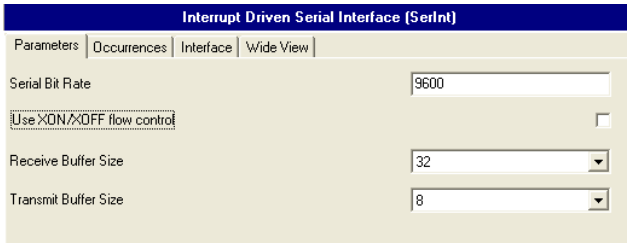
The first element that we will select is the serial interface. There are 3 asynchronous serial interface elements all under the Data tab. Select the Data tab and hover over an element with the mouse - a small help box will appear with the element name. Select the element called "Software serial interface - Interrupt driven" by clicking it. The element icon looks like this:

Now right click the element and a pop up menu will appear. Select the menu option "Help on selected element" and read through the help file entry for this element. It probably won't all make sense at the moment.

Now you can add this element to the project by double clicking it, or by dragging it on to the picture of the PIC. Do this and the element will appear in the element store at the bottom of the application designer. A picture of the element in a box will appear on the drawing of the PIC. When you select some elements another element will also be included this other element been *hooked* into the project by the serial interface. In this case no element is hooked by the serial interface.

Note that the element will have connected its pins to the PIC – Rx and Tx. This is because this element uses the USART hardware which is on fixed pins of the 16F88.

Now we must set the parameters of the serial interface. Click the parameters tab and the parameters for this element will be displayed. Set the serial bit rate to 9600, Clear the box labelled “Use XON/XOFF protocol” – we don’t need this, finally leave the Receive and Transmit buffer rates at 32 and 8 respectively. It will now look like this :



Some software elements including the Interrupt Driven Serial Interface element allow the user to define software functions to be called automatically when events occur. An event is described in the Applications designer as an *Occurrence*. In this case the Occurrence is that a byte has been received on the serial interface. When a byte has been received we would like to illuminate the LED. Click the Occurrences tab and a list of occurrences will be displayed, in this case there are two occurrences, one for when the transmit buffer is free, and one for when a byte is received. When a byte is received we would like to call a C function called "LEDOn".

Click the second occurrence “RxByte” to select it. In the "Calls for Occurrence" box type LEDOn and then click Add. The routine LEDOn will now be shown in the list of calls for this occurrence. Further functions could be added here if wished and they would be called in turn if a byte is received.

This completes the initialisation for the interrupt driven serial element.

Now we can add the switch which will be connected to pin B4. Select the Keys tab and double click the "Simple Switch" icon:

You will notice this time that two elements have been added, the first being the switch, the second being timer 0. The reason for this is that the simple switch includes debouncing and auto-repeat functions which need timer functions. The simple switch has automatically *hooked* in timer 0 as it uses it for the timer functions. Timer 0 can still be used by the application. You will notice that this element has the title "Simple Switch 0" in the blue title bar, this is because we can have multiple copies of the Simple Switch, the next would be called "Simple Switch 1".

Now connect the only pin on the element to pin RB4. The name of the pin will be SBIn0, this is not very meaningful so select the pin by clicking it, click pin RB4 to connect it. Now in the Pin Name box at the top of the application designer type the name "TxSwitch". Now when the application is generated there will be three symbols defined – a bit variable called TxSwitch and also two constants - TxSwitchPort and TxSwitchBit which represent the port and bit to which the switch is connected - we can test the switch directly by using a line such as:

```
if (TxSwitchPort&(1<<TxSwitchBit))
```

alternatively we can test it using the bit variable which is created with the same name :

```
if (TxSwitch)
```

Examine the parameters of the switch - we don't need to change any of the default values, so now click the Occurrences tab. When the switch is pressed we would like to transmit the last received value. So enter a function name "TxLastRx" and add it to the list of calls for the occurrence SBPress0.

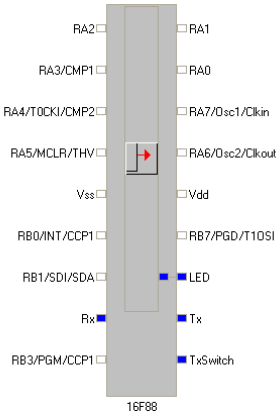
Now we can add the LED output which is on pin B6. Click the Ports Tab at the top and add the "Port Driver" element:

Connect the port output to pin RB6 and name it "LED". Initially the LED will be off so under the parameters tab select the initial value 0.

Finally we need to make all of the pins of 16F88 digital I/O pins instead of some being Analogue input pins. Add the last element from the Ports tab which is the All Digital element :

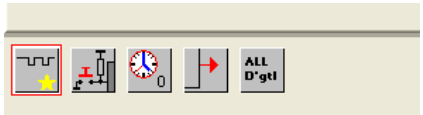
This has no parameters or pins or occurrences, it simply makes all analogue input pins set to digital I/O pins.

We have now completed the work with the application designer - our application will include initialisation, code and data for all the main functions of the application. Examine the PIC - it should look like the picture below, don't worry about the order of names on the pins of the PIC as it is dependent on the order of element selection.



You can print the PIC by right clicking the PIC graphic and using the *Print PIC* option of the pop menu. You can copy a picture of the PIC to the clipboard and paste into other applications by using the *Copy PIC* option of the same menu.

Check the bottom of the application designer as well to make sure you have included all the elements :



Generating the application for the first time

We have now selected the initial set of elements for the application, and the parameters, inputs, outputs, and occurrence calls have been defined, the application may be generated. To do this right click the PIC graphic and use the menu option *Generate Application* or press Control and the F9 key, or use the small button at the top left of the application designer:

The project window on the top right will show three files and a box titled "Compiler Options" will be shown, this allows the C Compiler options to be set. This box will appear the first time that a project is compiled, but will not appear again. To set the options after the first time then use the *Project | Set options for this project menu* option.

As we don't need to alter the default options then this can be ignored so click OK and the project will compile and assemble. At present we still have some code to define.

Now we would like to add some application specific code to the project. We would like an application specific header for the project. Click *File | New* and a new file will appear. Use the menu option *File | Save As* and select the file type Header Files, enter the filename "Tutor" and the file will be saved as "Tutor.h".

This file will include the memory variables used by our application. In this case it will simply be a flag which is set to 1 when a byte is received. Enter the following into Tutor.h:

```
bit RxFlag;        // Flag when byte is received
```

Double click the file in the project window called Tutor_User.C to open it. This file is produced automatically the first time that an application is generated (and is not generated again after that). We need to add the header to the file and also need to write the sub-routines which we defined for the occurrences. Move to the top of the file, you will notice that the first line of the file includes a header file called Tutor_Auto.h, this is the header which includes all the application designer information as well as the processor header. It should be included at the top of any additional files you include in the project. Include our new header by entering the following on the second line of the file:

```
#include "Tutor.h"
```

Look down the file to the UserInitialise function which will be empty, add the following code to the function between the curly brackets:

```
RxFlag=0;
```

This clears the receive flag while the program initialises. This isn't strictly necessary as the compiler clears memory when it runs, but is good programming practice and makes it clears what we are doing.

Examine the file. There is a function called UserLoop. This label will be jumped to by WIZ-C as it runs round its own main loop checking for occurrences and calling associated sub-routines. In this case we do not need to do any processing in the main loop - it is all undertaken by WIZ-C, so leave this section as it is.

Now look at the bottom of this file, the two functions that should be called when an occurrence happens will have been entered as templates. The LEDOn function will turn the LED on when a byte is received and will set the RxFlag when a byte is received, edit the function to read:

```
void LEDOn()
{
 RxFlag=1;                // Show a byte has been received
 LED=1;    // Turn on the LED
}
```

Note that when we named the output connected to our LED (recall that we called it "LED"), then the application generator automatically created the bit variable LED for us.

For the second routine which transmits the byte received we need to know what are the calls and variables used by the serial interface element. Position the cursor on the blank line in the middle of the TxLastRx function and press ALT and Enter together. A sub menu will appear, select the element Calls/Vars option, now a list of available functions for the elements used will appear. Select the AddTx line and double click it (or press Enter). A blank call will appear - we could have typed this in by hand, but this feature shows the parameters and some information on the call.

Bring up the Application Designer by pressing the extreme right hand button on the tool bar:

or by using the *Window | Application Designer* menu option. Click the serial interface element in the element store at the bottom of the Application Designer and click the Interface tab to see a list of interfaces for this element. Note the WaitRx() function – this picks up the next byte from the receive buffer – we need this to detect the received byte. This information is used to help finish the second routine which will transmit the last received value when the button is pressed. Now go back to the edit window and change the function as follows:

```
void TxLastRx()
{
 if (!RxFlag) return;           // Return if no byte received
 RxFlag=0;
```

```
 // void pSerialOut(unsigned char Tx);
 // - Transmit value to port
 AddTx(WaitRx());

 LED=0;          // Turn off LED
}
```

The complete Tutor_User.ASM file should now look like the listing below (the include file may vary according to where you saved the file). Use the *File* | *Print* menu option to print the file if you wish. Check the file and edit any changes.

```
#include "D:\\Data\\CDev\\Projects\\CCompiler\\PIXIE\\Bugs\\Tutor_Auto.h"
#include "Tutor.h"

//
// This file includes all user definable routines. It may be changed at will as
// it will not be regenerated once the application has been generated for the
// first time.
//

//*******************************************************************************
//
// Insert your interrupt handling code if required here.
// Note quick interrupts are used so code must be simple
// See the manual for details of quick interrupts.
//

void UserInterrupt()
{
   // Insert your code here

 #asmline SETPCLATH UserIntReturn,-1  ; SETPCLATH for interrupt routine
 #asmline goto UserIntReturn           ; Assembler - go back to interrupt routine
}


//*******************************************************************************
//
// Insert your initialisation code if required here.
// Note that when this routine is called Interrupts will not be enabled - the
// Application Designer will enable them before the main loop
//

void UserInitialise()
{
 RxFlag=0;
}

//*******************************************************************************
//
// Insert your main loop code if required here. This routine will be called
// as part of the main loop code
//

void UserLoop()
{
}

//
// User occurrence code
//

//
// Occurrence - Switch Pressed
//

void TxLastRx()
{
 if (!RxFlag) return;            // Return if no byte received
 RxFlag=0;

 // void pSerialOut(unsigned char Tx);
 // - Transmit value to port
 AddTx(WaitRx());

 LED=0;           // Turn off LED
}

//
// Occurrence - Byte received on serial interface
//

void LEDOn()
{
 RxFlag=1;                // Show a byte has been received
 LED=1;    // Turn on the LED
}
```

Now generate the application again by using the Ctrl and F9 keys or the button as described above. If you get errors then double click them in the Information window and correct the line with the error - use the listing above to see how they should read.

This is now the completed application which can be programmed into a PIC16F88 and run directly – the configuration fuses have been set automatically. However we can also simulate it and look at the results on simulated devices and the waveform analyser.

Simulation

It is not the intention of the introductory manual for WIZ-C to cover all the simulation capabilities of the environment which is covered in the later section. However we can check the operation of the program. It is possible to simulate with a stimulus file or with direct simulation of the external devices. We'll start with simulation of the external devices.

Switching screen layouts

There is a large amount of information provided on the screen and to aid users there are 3 main views :

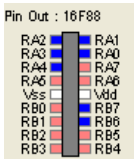
Compact	Press ALT+C keys
Debugging	Press ALT+D keys
Editing	Press ALT+C keys

In compact mode all windows (Debug, Project, Editting and Information) are shown on screen. In Editting mode the debug window is hidden and most screen space is given to the edit window. In Debugging mode most space is given to the debug window.

FED recommend that users get used to using the ALT and C, D or E keys to rapidly switch views. Normally only the ALT+D and ALT+E modes will be use, the compact mode is provided for existing users and is similar to previous versions of our environments.

Simulating with external devices

Press ALT+D to switch to the debugging layout. You will already see a “device picture”. This is a picture of the PIC with its pins shown. The pin colours show the state of ports. Red shows high and green low whilst the dark colours show the PIC is driving and light colours show the pins are operating as inputs, note the white pins which are not on I/O ports and the blue pins which on reset are Analogue inputs.



WIZ-C has the capability to simulate LED's, switches, LCD displays and a number of other devices which might be connected to the PIC.

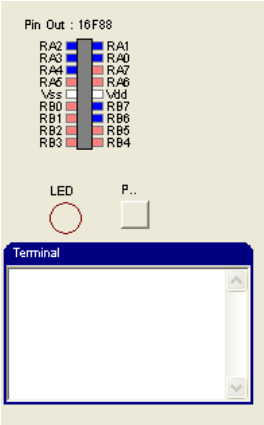
We'll start with the LED. Use the *Simulate* | *Add External Device* menu option. A dialog box will come up. In the External Device type box select LED. There are a number of parameters and values which may be selected for each device. For the LED most of these can be ignored apart from the connections. Under the Connections box there is a list box called Pins with two entries "Anode" and "Cathode". Select Anode and then use the Port box to select Port B, use the Bit box to select bit 6. Select Cathode and click the Connect Low option. This will connect the LED between Bit B6 and ground. Press the OK button. Note how the LED appears on the debugging window.

Next the push button. This time we’ll use a short cut.

Click the Show Application Designer button at the top of the screen ( ). The Application designer has a button on it which will automatically generate an external device to match the element. Click the Push Button element in the Element Store and then click the Create Device button which looks like this:

A push button will be created. Do the same for the serial element – click the element and then click the Create Device button.

Hide the Application Designer by pressing the button again. The new devices will be shown overlapped on the debugging window. It is possible to move them around by clicking on the title bar where the device name is shown and dragging the windows around. We adjusted them like this:



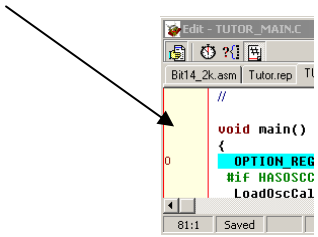
Now run the simulation by using the *Simulate | Run* menu option (you could also press F9, or use the button of the running man on the toolbar).

The LED will go out. Click the terminal box (the cursor will appear) and type the letter A. Watch the LED go on, press the Push Button and it will go out transmitting the character back to the terminal. Experiment with other characters. Note that you may have to hold the push button down for a few (simulation) milli-seconds before it registers. Note how the RB4 input changes to pale green as the push button is pressed and how the RB6 output changes from dark red to light red as the LED goes on and off. Change the TxLastRx() function as follows :

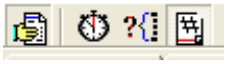
```
void TxLastRx()
{
 if (!RxFlag) return;            // Return if no byte received
 RxFlag=0;

 // void pSerialOut(unsigned char Tx);
 // - Transmit value to port
 while(GetRxSize())
 {
  AddTx(WaitRx());
 }

 LED=0;           // Turn off LED
}
```

Compile and run the program again, this time type several characters into the terminal window, press the button, note how they all the characters are output correctly in order, this shows the correct operation of the serial buffer.

Stop the program. Now look at the information bar – the yellow bar to the left of the edit window.



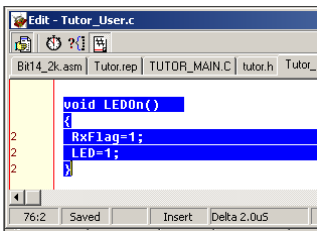
If the bar is not showing click the button to the top left of the edit window to turn it on (similarly click again to turn it off). Now the information bar can be set to show the address of each program line in the source, the time at which the line was last executed, or the number of times that it has been executed – click the buttons along the top of the edit window to select each option :



You can also determine how long a function or block of code takes to execute. Select a block of code – say the following block :

```
void LEDOn()
{
 RxFlag=1;                // Show a byte has been received
 LED=1;           // Turn on the LED
}
```

Now select the block of code with the mouse by clicking down before the void and dragging to the end of the function – you will see a box at the bottom of the edit window which will show the total time between the selected lines – in this case 2uS - the picture below shows the relevant part of the window:



You can drag over any area of code and the edit window will show the total time taken to execute that code provided that it has been simulated.

Simulating with a simulation file - using the waveform analyser

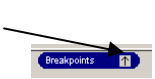
Next we will look at using a simulation file which will run alongside our external devices. Reset the processor by using the *Simulate* | *Reset* Processor menu option or press the reset button on the toolbar :

You may wish to switch back to the edit view using ALT+E. The first item is the simulation file - this will define the inputs to the program. Create a new blank file and save as type "Stimulus" with the file name "TutorInput.sti".

Select the project window and press Insert. A dialog box will open. Select Stimulus under the "Files of Type" box. Enter the filename "TutorInput" and press OK. Check that the file type is "Stimulus" in the Project File dialog box and press OK. Double click the file "TutorInput.STI" in the project window. We will generate a stimulus file which enters the character 41Hex at time 1mS, and then the key is pressed at time 5mS. To do this enter the following information (again the Simulator introductory manual covers this in more detail).

```
1m
serial9600-PORTB:0=41H    ; Enter character to Port B bit 0 at 1mS
5m
PORTB:4=0                 ; Press key at 5mS
```

Save the file by using the *File* | *Save All* menu option. Now we want to run the program until time 30mS and then stop to check the inputs and outputs. Press ALT+D to switch to debugging view. Bring up the breakpoint window by clicking the arrow on the minimised window:

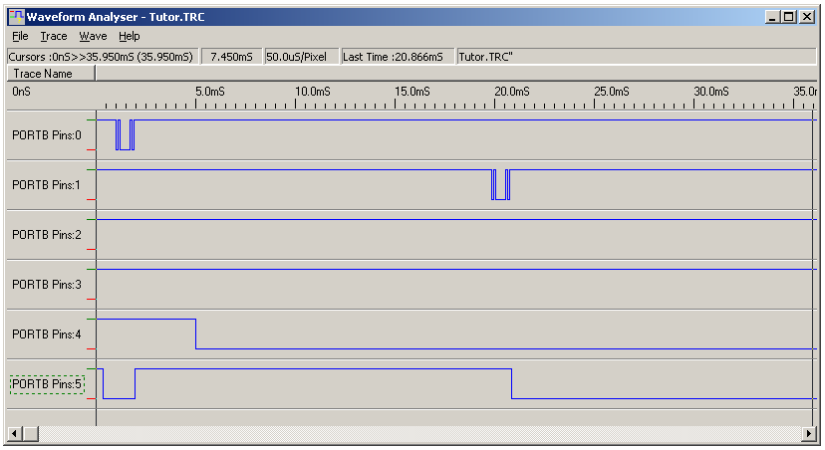


Right click the window and use the “Add Breakpoint” option. The Breakpoint Definition dialog box will be shown. Select "Break at Time", and then in the At Time box enter 30mS, click the OK button to define the breakpoint.

Now run the program by using the *Simulate* | *Run* menu option (you could also press F9, or use the button of the running man on the toolbar). Wait until the breakpoint is hit at 30mS. Use the *Tools* | *Examine Wave Window* menu option. The Wave Window "Define Trace Format" dialog box will be shown. In the Trace Name box select "Port B Pins", and then click the "Add as 8 line traces button". Resize the window to a comfortable size.

Finally press the F8 repeatedly button to zoom out and watch the received byte on Port B, bit 0, followed by Bit 5 going high (the LED turning on). Then at about 19mS the key press is detected (remember it is being debounced and takes a few milliseconds to detect it). You can see the received byte being transmitted on Port B, bit 1 and then the LED is turned off. The window can be copied to the clipboard or printed by using the options in the File menu for the Wave Window.

Here it is :



Run the simulation again - see how the use of the STI file operates with the external devices - the STI file has set the push button input low so until the button is pressed again the push button will automatically repeat every 250mS. Enter a character into the terminal and watch how it is repeated back automatically. Press the push button once to return to normal behaviour.

You are recommended to run through the simulator manual which follows this introductory manual.

Example 2

Digital clock operating to an LCD Display (in 10 lines of code)

In this example we shall show how to create a digital clock operating on an LCD display with less than 10 lines of code. Although in itself this project may not be that useful, it can operate in any project to provide subsidary timing functions.

Digital Clock Element

The digital clock element operates a full digital clock based on timer 0 which holds counters for seconds, minutes, hours and day of the week from 0 to 6. Either a 12 or 24 hour clock may be maintained.

It maintains five variables - Secs, Mins, Hours, pm and Day. Each of these is updated at the correct time and an occurrence happens when each item changes enabling a 7 segment or LCD display to be updated.

There are 3 functions for setting the time. IncMin(); IncHour(); and IncDay(); IncMin increments the minute by one, triggers the Minute passing occurrence and resets the seconds counters and internal counter chain to 0. It does not affect hours or days. IncHour() simply changes the hour without affecting seconds, minutes or days, but triggers the hour passing occurrence and IncDay() updates the day.

The accuracy of the clock is related to the overflow time of Timer 0. The faster that Timer 0 overflows the more accurate the clock. The internal counters are trimmed every minute and hour, and with a Timer 0 overflow of 1mS the accuracy is 1 part in $3.6x10^6$ which is an order of magnitude more accurate than most crystals.

For this example we shall operate a 24 hour clock showing hours, minutes and seconds, days will not be displayed. There will be two push buttons to set the time - set minutes and set hours.

LCD Displays

Introduction

The LCD element provides functions are to drive an LCD module based on the Hitachi chip set. The functions handle the 4 bit interface, and the device timing to the module. They also read the module busy flag and hold future transfers whilst the module is still performing the last operation. Functions are provided to initialise the module, to transfer single characters to the module, to transfer LCD module commands, and to write strings to the module.

Such modules are the LM020, LM016, LM018 and LM032, however there are a number of other modules based on this chip which is numbered HD44780. The module is driven from any port, however the data bits (D4 to D7) must be connected to bits 4 to 7 of the same PIC port. We'll connect the display to ports D and E as follows:

LCD Module	LCD Port number	Pin Number (2 line display LM016L)
RS	D2	4
R/W	D3	5
E	E1	6
D4	D4	11
D5	D5	12
D6	D6	13
D7	D7	14
Vss	-	1
Vdd	-	2
Vo (LCD Supply)	-	3

Functions

The LCDSTRING function sends the supplied string to the display. Thus to write “HELLO” to the display then the following can be used:

```
LCDSTRING("HELLO")
```

A number of macros and functions are provided to drive the LCD Display. These are as follows:

void LCD(unsigned int Data);

Write Data to LCD as a character for display.

void LCDPrintAt(unsigned char x,unsigned char y);

Macro to print at line y, column x

```
e.g.      LCDPrintAt(5,0); // Print at line 0, column 5
```

void LCDOnOff(unsigned char DisplayOn, unsigned char CursorOn, unsigned char Blink);

Macro to set up display. DisplayOn is 1 to enable the display, or 0 to turn it off, CursorOn is 1 to enable the display, or 0 to turn it off, Blink is 1 if the Cursor is to blink

```
e.g.      LCDOnOff(1,1,1); // Cursor on blinking
```

void LCDShift(unsigned char Cursor, unsigned char Right);

Macro to shift display Left or Right, Cursor is 1 or 0 to control shift with the cursor. Right is 1 for a right shift or 0 for a left shift.

```
e.g.      LCDShift(1,0); // Shift display left 1 character
```

void LCDClear();

Clear the display, return print position to 0,0.

Digital Clock Application

The application requires the digital clock element, the LCD element, and two push button elements. It will operate on the 16F877 - but in fact will run on most PIC's. Use the *File* | *Open/New Project* menu option to create a new project - or open this example which is included in the standard WIZ-C installation in the projects directory. Call the project DigClock.

Connect the elements as follows:

LCD

The LCD element is on the displays tab. It looks like this :

Once the LCD element is selected by double clicking it, it can be set up by connecting the pins to the pins of the F877. The only parameter for the LCD is the number of lines which needs setting to match the display. It does not need initialising.

Digital Clock

The digital clock element is on the Timers tab. It looks like this:

The only parameter is 12 or 24 hour selection. For this clock we'll use 24Hour. Now click on the Occurrences tab. Every time that the time changes (seconds, hours or minutes) then we'll update the display. So add a function call for SecPass, MinPass and HourPass. We'll use the same function for each - called UpDate. Ignore the Days for the moment.

Switches

Add two pushbutton elements for setting the hours and minutes. Connect one to RB0 and one to RB1. Now when a push is detected the minute (or hour) will be incremented. Right click on the digital clock element on the Timers tab. Select the "Help on Selected Element". This will bring up the help file. Look at the Public Calls and Variables section. Note that there is a function to increment minutes (which also clears the seconds counter) called IncMin, and one to increment the Hour. Return to the Application Designer. Add IncMin to the list of occurrences for RB0, and add IncHour to the list of occurrences for RB1. Now we need to write no code directly for the switch presses - the switch presses will translate directly to increment the minute and hour counters. On incrementing them the Digital Clock element will generate occurrences to update the display.

User Code

Generate the application (use Ctrl+F9).

Open the DigClock_User.C file. For the clock we can set up a welcome message on power up by including code in the UserInitialise() function as follows:

```
void UserInitialise()
{
 OPTION_REG&=0x7f;        // Port B pull ups
 // void LCDString(char *str);
 // - Write a string to the LCD
 LCDString("Digital Clock");     // Welcome message
 Wait(2000);                     // Wait for 2 seconds
 LCDClear();                     // Clear the display
}
```

Now to print the time to the display we need a function to convert an 8 bit number to a string. Use the *Help | Compiler Contents* menu option to open the C Compiler help file. Click Library Reference and then String Print Functions to bring up help on the functions which print numbers to strings. CPrintString is the function that we can use, however it does not print a leading zero if the number is less than 10. Add the following function to the bottom of the DigClock_User.c file which will print a number with a leading zero if necessary:

```
//
// Print a 2 digit number with a leading 0
//
void RJPrint(unsigned char v,char *s)
{
 if (v<10) { *s='0'; s++; }
 cPrtString(s,v);
}
```

Finally we need to write the UpDate() function which will print the time to the display. Add this function to the bottom of the file:

```
//
// Print the time on the display
//
void UpDate()
{
 char Ds[12];             // String to display

 RJPrint(Hours,Ds); Ds[2]=':';           // Hours
 RJPrint(Mins,Ds+3);  Ds[5]=':';         // Minutes
 RJPrint(Secs,Ds+6);                     // Seconds
 LCDPrintAt(0,0);
 LCDString(Ds);                          // Print time to LCD
}
```

This is quite straightforward, when the display is to be updated it prints the time to the first row, first column.

Final application

Generate the application (use Ctrl+F9). The project may be simulated using the external devices. Select the LCD on the application designer and press the light bulb to generate the LCD device. Set the display to 2 lines by 16 rows (this is owing to an anomaly with the method of operation of LCD displays with one row). Add push buttons for hours and minutes connected to RB1 and RB0 respectively. Run the program with update rate set to 20000. Watch the display and press the hours and minutes buttons. Note that the simulation is far behind real time owing to the use of a 20MHz clock. You can simulate faster than real time by using the application designer and setting the oscillator rate to 1MHz (actual simulation speed will depend on the speed of the PC - this manual was written around a simulation running on a 266MHz Pentium).

If the application is to be simulated without external devices then the input D7 for the display should be set to 0 which will make the LCD library routines believe that a display is acknowledging. For our example this could be achieved with the following line in a STI file:

```
PORTD:7=0  ; Set port D bit 7 to zero
```

The final application file DigClock_User.c looks like this:

```
#include "C:\\Program Files\\FED\\WIZ-C\\Projects\\DigClock\\DigClock_Auto.h"
#include <Delays.h>
#include <Strings.h>

//
// This file includes all user definable routines. It may be changed at will as
// it will not be regenerated once the application has been generated for the
// first time.
//

// ******************************************************************************
//
// Insert your interrupt handling code if required here.
// Note quick interrupts are used so code must be simple
// See the manual for details of quick interrupts.
//

void UserInterrupt()
{
   // Insert your code here

 #asmline goto UserIntReturn     ; PIC Assembler - go back to interrupt routine
}


// ******************************************************************************
//
// Insert your initialisation code if required here.
// Note that when this routine is called Interrupts will not be enabled - the
// Application Designer will enable them before the main loop
//

void UserInitialise()
{
 OPTION_REG&=0x7f;        // Port B pull ups
 // void LCDString(char *str);
 // - Write a string to the LCD
 LCDString("Digital Clock");
 Wait(2000);
 LCDClear();
}

// ******************************************************************************
//
// Insert your main loop code if required here. This routine will be called
// as part of the main loop code
//

void UserLoop()
{
}

//
// User occurrence code
//

//
// Print a 2 digit number with a leading 0
//
void RJPrint(unsigned char v,char *s)
{
 if (v<10) { *s='0'; s++;}
 cPrtString(s,v);
}

//
// Print the time on the display
//
void UpDate()
{
 char Ds[12];             // String to display

 RJPrint(Hours,Ds); Ds[2]=':';           // Hours
 RJPrint(Mins,Ds+3);  Ds[5]=':'; // Minutes
 RJPrint(Secs,Ds+6);                     // Seconds
 LCDPrintAt(0,0);
 LCDString(Ds);                          // Print time to LCD
```

```
}
```

Including the day

Adding the day is very straightforward. A new push button element is required - connect to RB2 and couple the occurrence to IncDay to set the day. Add UpDate to the occurrence for DayPass for the Digital Clock element. Change the UpDate function as follows:

```
//
// Print the time on the display
//
const char *DayStr[]=
{
 "Sun ",
 "Mon ",
 "Tue ",
 "Wed ",
 "Thu ",
 "Fri ",
 "Sat "
};

void UpDate()
{
 char Ds[12];            // String to display

 RJPrint(Hours,Ds);   Ds[2]=':';         // Hours
 RJPrint(Mins,Ds+3);  Ds[5]=':';         // Minutes
 RJPrint(Secs,Ds+6);                     // Seconds
 LCDPrintAt(0,0);
 LCDString(DayStr[Day]);                 // Print day to LCD
 LCDString(Ds);                          // Print time to LCD
}
```

The DayStr array is set up in ROM to minimise RAM usage, the correct item from the array is printed before the rest of the time string.

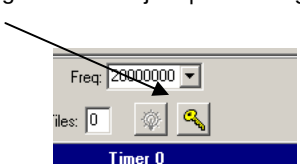
Taking it further

Try changing the clock to operate on a 12 hour clock with a pm indicator.
Make it into an alarm clock.
Extend into a multi-function timer.

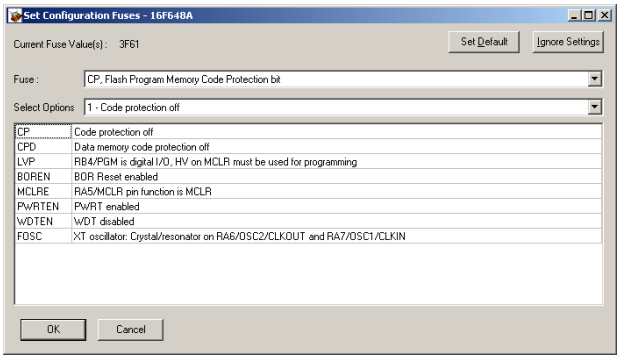
Setting the configuration fuses :

The C Compiler provides support for simple set up of the configuration fuses which is essential to do before running the program.

To bring up the Configuration fuses dialog use the Project | Set configuration fuses menu option, or in WIZ-C use the button on the application designer :



This will bring up the Set Configuration fuses dialog box :



This is very straightforward to use, and is described in section 3.10 of the C Compiler manual.

FED PIC & AVR Integrated Development Environments

(WIZ-C, AVIDICY, AVRDE, PICDE & C Compilers)

Simulation System

Introductory Manual

The simulation system provided with FED's development environment is flexible and comprehensive.
The program, and its support files and example programs are
© Copyright Robin Abbott, 1996-2006. <robin.abbott@fored.co.uk>.

The program may be installed onto the hard disk of only one Personal Computer, and must removed by deleting the executable file, and all the support files before installing onto a different computer.

Forest Electronic Developments
12 Buldowne Walk
Sway
HAMPSHIRE
SO41 6DU
Sales : +44 - (0)1590 - 681511

info@fored.co.uk

Or see the **Forest Electronic Developments** home page on the world wide web at the following URL:

http:\\www.fored.co.uk

Simulator

Contents

Simulation System

Introduction

Debugging and editing views

The Debugging Window

Debugging Windows

Working with layouts

Working with external devices

Working with Variables

How to change the program counter

Breakpoints

Running the waveform analyser

Waveform Generator

Working with Stimulus and Injection Files

Simulation Accuracy

Introduction

The simulation system for PIC and AVR devices is very powerful, and yet is intended for use by beginners.

Debugging and editing views

There is a large amount of information provided on the screen and to aid users there are 3 main views :

Compact	Press ALT+C keys
Debugging	Press ALT+D keys
Editting	Press ALT+E keys

In compact mode all windows (Debug, Project, Editting and Information) are shown on screen. In Editting mode the debug window is hidden and most screen space is given to the edit window. In Debugging mode most space is given to the debug window.

FED recommend that users get used to using the ALT and C, D or E keys to rapidly switch views. Normally only the ALT+D and ALT+E modes will be use, the compact mode is provided for existing users and is similar to previous versions of our environments.

The debugging window

The debugging window is very flexible and allows external devices and windows which show the state of the PIC or AVR during simulation to moved and resized freely and to appear on one or more of the tabs shown at the top of the window.

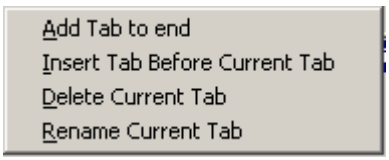
Throughout this manual we'll assume the system is set up as the default for new projects. If you wish to switch to the default the press ALT+D and then click the button shown below at the top of the debugging window which will load the default layout for the debugging window:

Tabs

The debugging window has a number of tabs at the top :

All of the sub-windows on the debugging window may be assigned to one or more of the tabs on the debugging window. By default new external devices will appear on every tab, whilst the information sub-windows (such as breakpoints) will appear only on one or two tabs.

To add or delete a tab then right click one of the tabs, the tabs menu will appear :



The menu options are self-explanatory – there are a maximum of 32 tabs, you cannot delete the last tab, and if you insert a tab it will appear before the current tab (the tab which was right clicked).

Sub-Windows

The Debugging Window has a series of sub-windows, these may be selected and moved at will. Each window may be minimised and will appear as a bar until selected again. Here are the complete set of windows :

These are described one by one in the section Debugging Windows.

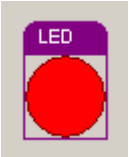
Every external device is also a window in its own right. Windows can be setup with a border and a caption bar. Right click the top bar of any window to show a menu where the border and caption can be selected. To minimise a window which has no caption bar then right click at the top (or just above the external device) and use the minimise menu option.

To resize a window move a mouse over the bottom or right of the window and click and drag it to the new size. Some external devices can be resized (e.g. LED's) and some cannot (e.g. LCD displays).

To move a window move the mouse over the top bar of the window (or just above an external device if it has no top bar), click the mouse and drag the window.

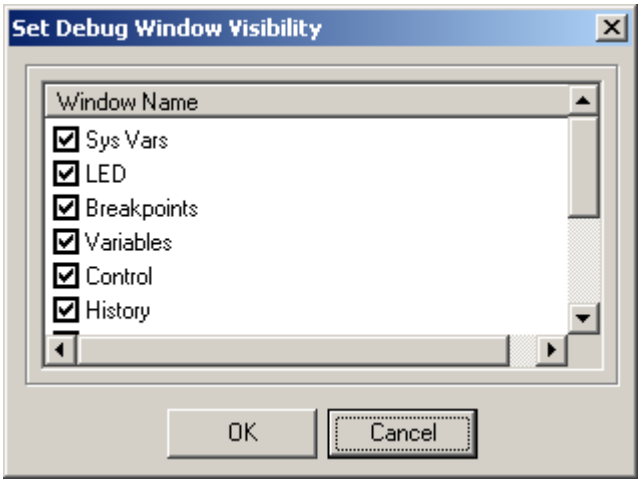
The little arrow in the top right of the window bar minimises or grows the window.

Here is a slightly unusual view of an LED which has been set up with a border and caption bar and also resized.



Hiding Windows

It is possible to hide sub-windows which are not required. To do this right click on a blank part of the debug window and click the **Window Visibility** menu option. The following box will appear :

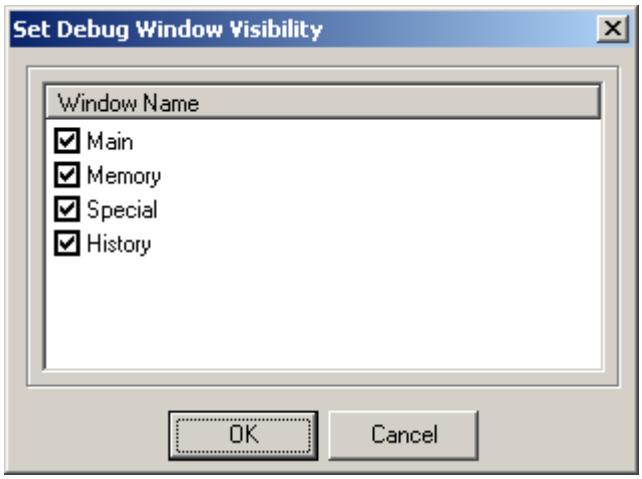


This shows the complete set of windows including any external devices. Here we can see one external device called LED. Remove a check box by any window which is not required and click OK. Invisible windows will be completely hidden from the user.

Moving Windows from tab to tab

It is possible to move windows from tab to tab, or to show windows on more than one tab. By default all external devices appear on all tabs. Right click just above an external device, or right click the caption bar of a window to bring up the options available.

The **Change Page** menu option brings up the following box :



A list of tabs will appear – the tabs on which the window will appear may be selected and cleared. It is not possible to remove a window from every tab (in this case it will stay on the first page). If you wish to make a window invisible use the **Window Visibility** menu option (right click a blank part of the debugging window).

Debugging Windows

The Variables window [Default – Main Tab]

The Variables window shows a list of variables which may be added by the user. Right click the window and use Insert to include a new variable or value or to change the way a variable is examined.

Double click a variable to bring up the Modify Memory Dialog Box which allows the value to be changed, similarly double click any item in the watch window to bring up the dialog box and change its value.

The Sys Vars window [Default – Main Tab]

The Sys Vars window shows the registers, the ports, and a list of memory locations which are essential to the operation of the C Compiler. These items are automatically included after a compilation and are updated whenever they change. Double click a register to bring up the Modify Memory Dialog Box which allows the value to be changed.

The Stack Window [Default – Main Tab]

The stack window shows a listing of the stack. Double clicking a stack value will show the line in the main edit window to which the program will jump back when a return is executed at that location.

The Control Window [Default – On all Tabs]

The control window has various controls on it which affect the simulation. For WIZ and AVIDICY users the processor clock frequency is set in the Application Designer. The Change PC button allows the value of the PC to be set.

The Breakpoint Window [Default – Main, Memory and Special Tabs]

The breakpoint window shows the breakpoint list. Double click a breakpoint to edit it, press insert to create a new breakpoint, press delete to remove the selected breakpoint. Note that all breakpoints included by setting breakpoints on boxes are shown here as well. See also Breakpoints.

The Terminal window [Default – Main and Memory Tabs]

The terminal is a standard RS232 terminal which allows bytes to be sent to and received from application programs connected to the serial port. Right click the terminal window to bring up the options for it, or use the Module menu. The terminal window is selected by clicking the Terminal tab in the debugging window.

The options under the Module menu which refer to the terminal window are :

Communications. This option brings up the Set Serial Port Parameters

Send File. This allows a file to be transmitted over the serial port.

The Special window [Default – Special Tab]

The Special window examines a number of special internal device registers – these should be self explanatory.

Controlling the Simulation

The update rate is a slider which is shown at the top of the IDE :

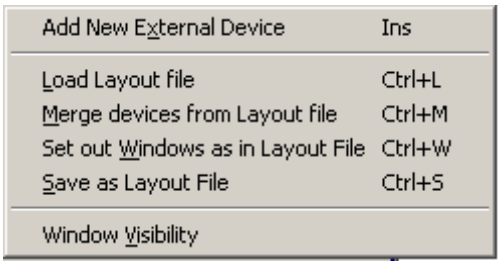
At the left the value is 1. Almost at the right the value is 64000. This represents the number of cycles that the processor should execute before updating the watch variables. The higher the number then the faster the simulation. If the slider is set to the extreme right then the variables are not updated whilst running unless the "Update Now" button is pressed, or a simulation breakpoint is hit.

Also at the top of the IDE is the Stop on Errors Check Box. If enabled this will stop the simulation should an error occur during the simulation Run. If this Check Box is not selected, and if an error occurs, then errors will be printed in the Information Window, but the simulation will continue.

Working with layouts

Clearly it is possible to configure the debugging window in a number of different ways, and it may be desirable to have different layouts for different projects. For example a general purpose development board may have a standard layout with a picture of the board and external devices for LED's and LCD's and push buttons which may be used for a number of different projects.

To this end it is possible to save the layout of the window including all the external devices. To do this right click on a blank part of the debugging window to bring up the layout menu :



Use the **Save as Layout File** option to save the current layout, the layout will be saved with the current tabs, external devices, window positions and visibility.

Use the **Load Layout File** option to delete all current devices and load up the devices and window layout in the external file. Note that it is possible to load a layout from any project file, not just those saved as layout files.

Use the **Set out Windows** as in Layout File option to rearrange the windows but ignores any external devices.

Use the **Merge devices from Layout** file option to bring in any external devices in the layout file whilst leaving all current devices and window layout intact. This is a very useful option. For example use this menu option and load the file LEDBarPortB. This is a group of 8 LED's attached to port B bits 0 to 7 and gives a quick visual indication of the state of the pins. There are a number of predefined layouts in the Layouts sub-directory of the IDE.

Note that it is possible to merge devices from any project file, not just those saved as layout files.

There are two pre-defined layouts – Classic and Default. The classic layout is loaded automatically for projects created before version 11. The default layout is loaded for all new projects. You can quickly switch the window layout to either of these versions using the two buttons at the top left of the debugging window:

The left button switches to default view, the right button (the 'C' button) switches to classic view which has most of the debugging windows on their own tab.

Working with external devices

This section describes the external devices which may be simulated. External devices are simulated on the debugging window immediately above the debugging control tabs. Each device has a number of inputs and outputs which may be connected to the device pins. It is also possible to write models for new types of device.

There are a number of elements such as Graphic LCD and touch sensitive buttons which are described with the relevant library. The inbuilt devices are shown here.

Introduction to External Devices

Generating devices automatically from the application designer

External Debugging Device Dialog Box

Handling External Devices

The list of external devices is as follows:

7 Segment Displays

Audio - Buzzer

Audio - Piezo

Bitmap

Clock Generator

Device Picture

Encoder - Incremental

Hex Keypad

I2C Devices

LCD Module

LED

Logic Analyser

OneWire – Bus

OneWire – DS18S20/DS1920

OneWire – DS1990A/DS2401/DS2411

Push Button

Potentiometer

PWM Voltmeter

Shape

Serial Terminal

SPI - Eeprom

SPI - Monitor

SPI - Terminal

Wire - Digital

Writing external device models

Introduction to External Devices

Any number of external devices may be simulated, and each type may be simulated any number of times - it is quite permissible to simulate 3 potentiometers and two LCD display modules for example.

To add an external device then use the **Simulate | Add External Device** menu option. This will bring up the External Debugging Device Dialog Box.

To delete the device then click on the device name above it (the name will then be underlined), use the right mouse button to bring up the menu and use the delete option.

To edit the device (to change parameters or pin connections) then click on the device name above it (the name will then be underlined), use the right mouse button to bring up the menu and use the edit option.

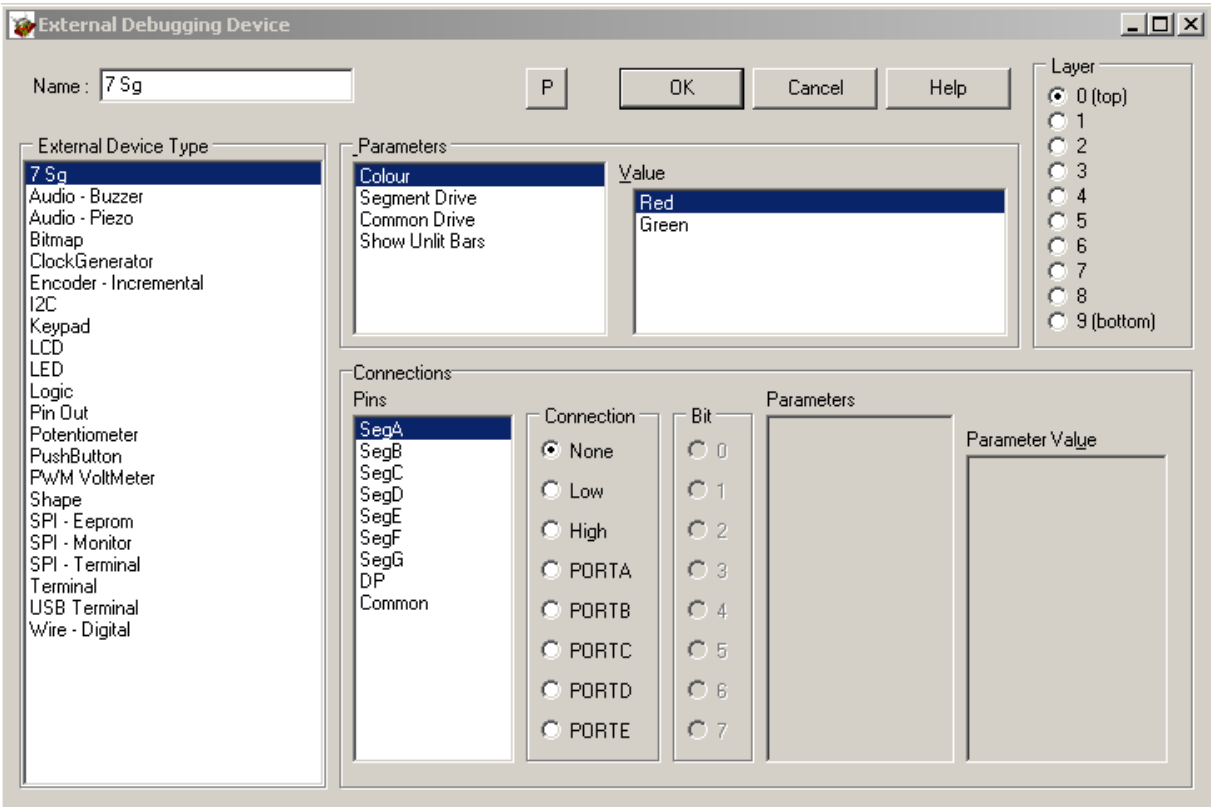
Generating devices automatically from the application designer

The easiest method to generate a device is to use the application designer. Click the element you wish to use as a template and press the create device button :

A device will be created with the same pin connections and parameters as the application designer. Note that not all application designer elements have associated devices and the button will only be enabled if there is an associated device.

External Debugging Device Dialog Box

This box looks like this :



This dialog box allows external devices to be added or edited.

The list of devices is shown in the "External Device Type" box. A device being edited cannot be changed. Select the device type in this box.

The parameters box shows a list of parameters for the device. For example "Colour" or "Bit Rate". Click on a parameter to select it and the Value box will show a list of available options. Click on the option to be used for the device.

The connections box shows a list of pins for the device. Each pin may be left unconnected, connected low (to 0V), high (to +5V), or connected to any of the pins of the device. In the connection list choose Low or High for the relevant logic level, use the Port options to connect to a pin on that port..

Some devices allow each pin to have their own parameter. In this case the Parameter and Value boxes may be selected for each pin.

Please note that the external devices have no concept of drive impedance. The PIC or AVR pins when set to inputs default in the simulator to a high state. Thus if an LED anode is connected to a PIC or AVR pin defined as an input it will see a high state and will illuminate if the cathode is low, in a real device this would not happen. This is unlikely to be a problem for the great majority of applications which will set outputs to drive during initialisation.

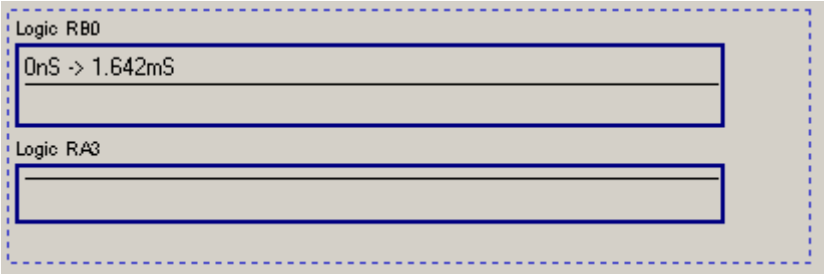
The layer shows how the device will be drawn. Devices on layer 0 will always appear over devices on layer 1 and so on. Some devices such as bitmaps and shapes appear on lower levels by default.

Finally the name of the device (shown on the debugging window above the device) may be typed into the "Name" box. Clear this box if the device does not need a name.

Handling external devices

It is possible to undertake a number of actions with external devices. It is important to understand that the menu for a device will only appear if the caption is clicked. The caption is the name which appears just above the device. If there is no caption because the name is blank then simply right click just above the device.

To select one or more devices. Click the device in the caption area, to select more devices use the shift key to select others, to select all devices in an area then click and hold the mouse outside the area and drag a rectangle round all devices to be selected :



To group devices so that moving or copying one device will move or copy all of the group together. Then select all the devices to be grouped, right click just above one of them (in the caption of the device) and use the group option.

To ungroup devices. Then select one of the devices in the group, right click on the caption area and use the ungroup menu option.

To delete a device. Then select a device, right click the caption area and use the Delete menu option, alternatively select it and use the delete key..

To duplicate a device. Then select a device, right click the caption area and use the Duplicate menu option, this will create an exact copy including all the pin connections.

To align devices so that they are all lined up on screen then select the devices, right click them and use the Align menu option, it is possible to line them up horizontally, vertically or space them on screen equally.

7 Segment Displays

The 7 segment display device emulates a 7 segment display. The display is shown at full brightness when the common pin is enabled. However when the common pin is disabled the display shows the last pattern at half brightness. This enables multiplexed displays to be simulated.

Parameters	**Options**	**Notes**
Colour	Red, Green	The colour of the display
Segment Drive	Low, High	Selects whether the PIC or AVR pin needs to be low to turn on the segment or high.
Common Drive	Low, High	Selects whether the Common pin needs to be low to turn on the display or high. Note that the AVIDICY, PIXIE and WIZPIC 7 segment elements assume that a drive transistor will be used to turn on a display. So these elements drive the Segment and Common pins in the same sense.
Show Unlit Bars	No, Yes	Unlit bars may be displayed (as dark gray bars) if desired.
Connections	**Parameters**	**Notes**
SegA to SegG	None	The 7 segments of the display.
DP	None	Decimal point, leave unconnected if not required.
Common	None	The common pin for the display.

Audio - Buzzer

This device mimics a simple on/off buzzer with a fixed frequency. It is displayed using the schematic symbol that is normally used for a buzzer. The terminals are drawn using red (for high), green (for low) and black (for unknown). Each terminal has the name of the PIC pin (or Vcc, Vss, NC) it is connected to drawn next to it.

When the buzzer's body is coloured red this implies an error - an unconnected or reversed terminal has been detected or if the buzzer is permanently connected between Vcc and Vss".

The buzzer has an audible and a visual feedback of it's "buzzing state". Audible feedback is accomplished through the use of the PC's soundcard/speakers. The frequency is configurable. The visual feedback consists of displaying several "sound waves" in the device picture.

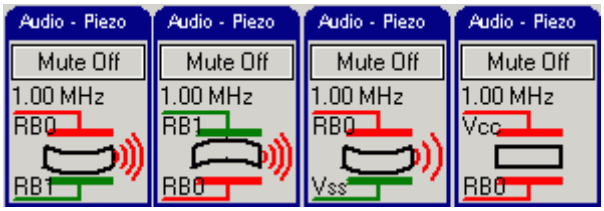
The device contains a single button. This button is used to mute the sound. When the sound is on, the button displays the text "Mute". When sound is off, the text "Mute off" is displayed. The button thus displays the action that will be executed when the button is clicked.

The device is fully resizable.

Parameters	**Options**	**Notes**
Frequency	Entered value	The frequency (in Hz) of the audible feedback.
Mute	Sound is on Sound is off	Selects whether the sound (audible feedback) is on or off at startup.

Connections	**Parameters**	**Notes**
Positive	None	The positive terminal of the buzzer.
Negative	None	The negative terminal of the buzzer..

Audio - Piezo



This device mimics a PWM-driven piezo. It is displayed using the schematic symbol that is normally used for a piezo transducer. The terminals are drawn using red (for high), green (for low) and black (for unknown). Each terminal has the name of the PIC pin (or Vcc, Vss, NC) it is connected to drawn next to it.

When the transducer's body is coloured red, an error has been detected. Errors are unconnected or reversed terminals or permanent connection between Vcc and Gnd.

The piezo has an audible and a visual feedback of it's "buzzing state". Audible feedback is accomplished by calculating the frequency of the input-PWM. If this frequency is within the human audible range (20 Hz – 20,000 Hz), it is heard through the PC's soundcard/speakers. The visual feedback consists of several "sound waves" in the device picture as well as the measured frequency. When both terminals of the piezo are PWM-driven, the emitted volume and the displayed number of sound waves will be higher than when a single-side PWM is used.

The device contains a single button. This button is used to mute the sound. When the sound is on, the button displays the text "Mute". When sound is off, the text "Mute off" is displayed. The button thus displays the action that will occur when the button is clicked.

The device is fully resizable.

Parameters	**Options**	**Notes**
Mute	Sound is on Sound is off	Selects whether the sound (audible feedback) is on or off on startup.

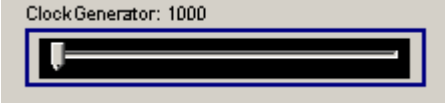
Connections	**Parameters**	**Notes**
Terminal 1	None	The two non-polarised terminals of the
Terminal 2	None	piezo transducer.

Bitmap

The Bitmap device allows a bitmap to be loaded and shown on the debugging window as a background. For example a picture of a development board can be loaded and devices placed on top of the board such as LED's or LCD's

There are no parameters or pins for this device – just click the FileName button and load the correct file which may be in bmp or jpg format.

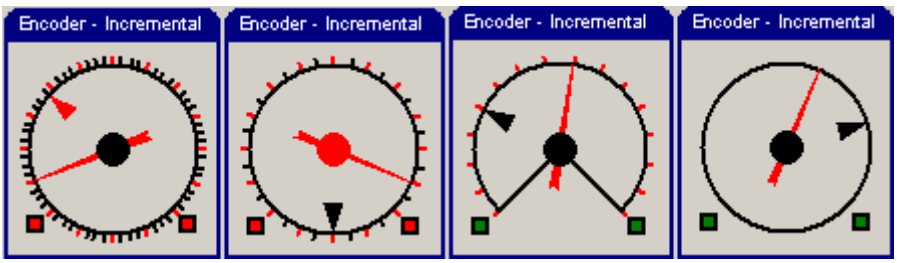
Clock Generator



The clock generator will send a series of pulses into the device on the connected pin. It is possible to set the minimum and maximum frequency of the generator and then adjust the frequency using the slider.

Parameters	**Options**	**Notes**
Minimum Frequency	Entered value	The frequency of the clock when the slider is at the extreme left.
Maximum Frequency	Entered value	The frequency of the clock when the slider is at the extreme right.
Connections	**Parameters**	**Notes**
Clock	None	The clock output of the generator.

Encoder - Incremental



The encoder has a quarter, half or full quadrature output. A detent free option is also available. All rotations and switch-operations are buffered and sent to the MCU in a timely way. The large red pointer indicates the user input while the small triangle displays the progress of sending these actions to the MCU. Clicking on the centre hub presses the switch. A red hub indicates closed switch-contacts, a black hub indicates open contacts. When switch actions are sent to the MCU, the triangle will also show red for closed contacts and black for open contacts. A visual feedback of the quadrature outputs A and B is available as two square boxes at the bottom of the picture (red = high, green = low).

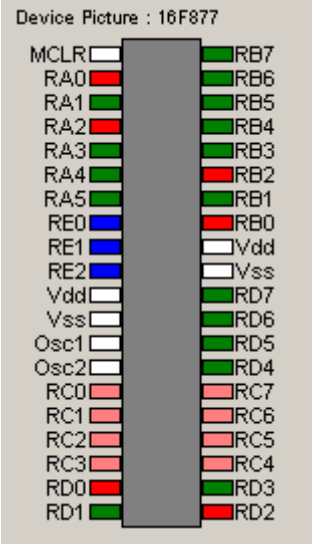
You can choose how to control the encoder using either the Mouse Wheel or buttons (or both). Mouse wheel action can be accelerated by simultaneously pressing Shift (5 times faster) or Ctrl (10 times).

The device is fully resizable.

Parameters	**Options**	**Notes**
Pulses per Revolution	Entered value	The number of full pulses per rotation

Detent Factor	0 (No Detents) 1 (Quarter Quadrature) 2 (Half Quadrature) 3 (Full Quadrature)	One to four detents per pulse, or no detents at all.
Travel	Continuous 90 degrees 180 degrees 270 degrees 300 degrees 360 degrees	The maximum angle that is used. Please note that “continuous” (without stop) is not the same as “360 degrees” (with stop).
Initial Position	Random Left Middle Right	The position of the encoder at start up or after a reset.
Maximum RPM	Entered value	The maximum revolutions per minute supported by the device. It is used to calculate the shortest pulse widths that can be send to the MCU.
Show Detents	No Yes	Should the detents be visible?
Switch Type	No Switch Momentary, Normally Open Momentary, Normally Closed Alternate (sticky)	The pushbutton definition
GUI Mode	Use Mouse Wheel Use Buttons Use Both	Default operation is by Mouse Wheel. Upon request, buttons are displayed.
Connections	**Parameters**	**Notes**
Terminal A	None	Quadrature Output A.
Terminal B	None	Quadrature Output B.
Common C	None	Common.
Switch D	None	Switch terminal.
Switch E	None	Switch terminal.

Device Picture



The Device pictrure shows an outline view of the device together with the state of all of the pins. The colour coding of the pins is as follows :

Red	Device Output driving high
Pink	Device Input being driven high
Dark Green	Device Output driving low

Bright Green	Device Input being driven low
White	Not a device I/O pin
Blue	Analog input

Parameters	Options	Notes
Show Pin Text	Port, All, None	Determines whether the text written by the pin should show the Port and pin (e.g. RA2), the full name (RA2/AN2/VRef-), or no text at all (the last being useful if the device is sized to a small scale).

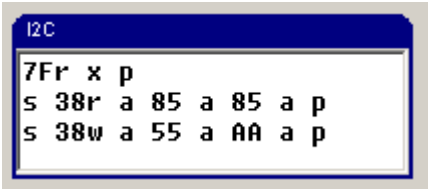
Hex Keypad

The hex keypad simulates a 4x4 hex keypad with 4 rows and 4 columns. Key 0 is row 0, column 0, Key 1 is row 0, column 1 and so on up to Key F which is row 3, column 3. When a key is pressed with a mouse the device connects the input to the output. If the key is held down with the mouse then it will remain pressed.

Parameters	Options	Notes
PIC Drives (AVR Drives)	Rows, Columns	Selects whether the PIC or AVR is driving rows or columns. For the FED PIXIE elements the PIC drives the keypad rows.
Pull Ups On PIC (AVR) Inputs	Pull Up, Pull Down	Selects whether the PIC or AVR inputs connected to the keypad are pulled up to +5V with resistors or down to ground with resistors. For the AVIDICY, PIXIE and WIZPIC elements the device is connected to +5V with pull ups.

Connections	Parameters	Notes
Row 1 to Row 4	None	The 4 rows of the display.
Col 1 to Col 4	None	The 4 columns of the keypad.

I2C Devices



The I2C device provides a simulation of a I2C device on a bus through a terminal window which shows the state and values presented on the bus together with the capability to specify values to be transmitted to the PIC or AVR when the bus is read.

The terminal will display the following to show the state of the bus :

Character	Meaning
s	Start state
p	Stop state
r	Read bit detected at end of address
w	Write bit detected at end of address
a	Acknowledge by addressed device
x	No acknowledgement received
XX	Hex value transmitted or received on bus

Timing Violations

The device model measures clock width for timing violations. This follows the I2C specification, in practice many devices better this specification so if timing violations are reported then it is worth checking that clock width is within spec, and then switching the device to a higher rate to prevent these warnings.

Parameters	Options	Notes
Device Rate	High Speed (3.4MHz) Standard (400KHz) Low Speed (100KHz)	Selects the device rate for timing checks. Switch to a higher rate if timing violations are received, but the Device itself is within specification.
Address Length	7 Bits, 10 Bits	Selects whether the device accepts 7 or 10 bit addressing.
Address Range Low	Entered Value	Lower value of the range of addresses which will be accepted by the device. The range of addresses acknowledged by the device will run from this address to the Address Range High value.
Address Range Upper	Entered Value	Upper value of the range of addresses which will be accepted by the device.
Byte To Read	Entered Value	Byte value which will be read from the device on the bus should the address be within range.

Connections	Parameters	Notes
SDA	None	The IIC SDA connection.
SCL	None	The IIC SCL connection.

LCD Module

The LCD module simulates LCD modules of up to 80 characters based on the Hitachi chipset. The module simulation includes reading of busy flag (and the timing delays in the real module are simulated accurately). Full initialisation and 4 bit and 8 bit transfer mode are supported. CG RAM is simulated for read and write, but the user defined characters are not simulated, similarly the character set shown is ASCII rather than an accurate representation of the ROM based character set. The cursor is not simulated, however shift and display commands are all accurately simulated.

Parameters	Options	Notes
Rows	1, 2 or 4	The number of rows on the display. Note that 4x20 row displays behave like 2 rows of 40 characters with a line break.
Chars Per Row	8, 16, 20, 32, 40	The number of characters per row on the display.

Connections	Parameters	Notes
E	None	The E pin of the display
RS	None	The RS pin of the display
R/W	None	The Read/Write pin for the display. Note that some display drivers only write to the display - for this type of drive the R/W pin can be connected low.
DB0-DB7	None	The data pins of the display. For a 4 bit interface only DB4-DB7 are used, the other pins can be left unconnected.

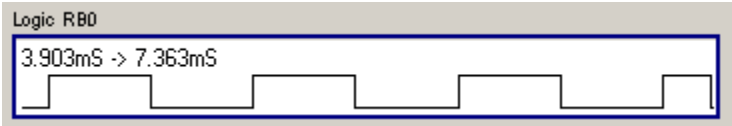
LED

The LED device simulates a simple LED which may be resized. The LED will light when the Anode is positive and the Cathode is negative. To Simulate drive through a transistor each pin may be specified as connected through a drive transistor in which case the sense of the pin drive will be inverted.

Parameters	Options	Notes
Colour	Red, Green, Yellow,	

	White or Blue	The colour of the LED..
Shape	Round, Rectangle, Arrow right, Arrow left, Triangle	The shape of the LED..
Connections	**Parameters**	**Notes**
Anode	Transistor Drive	The Anode of the LED - must be high to illuminate the LED. If transistor drive is set to yes then the LED will illuminate when the PIC or AVR Anode pin is low.
Cathode	Transistor Drive	The Cathode of the LED - must be low to illuminate the LED. If transistor drive is set to yes then the LED will illuminate when the PIC or AVR Cathode pin is high.

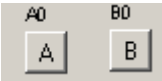
Logic Analyser



The Logic analyser shows the past history of a pin in digital format as a line trace.

Parameters	**Options**	**Notes**
Time Per Pixel	100nS to 10S	The sampling rate of the analyser, every sample is displayed as one pixel.
Display times on window	Yes/No	Selects the option to display the time on the logic analyser window showing the range of the displayed trace.
Connections	**Parameters**	**Notes**
Sample	None	The sample pin of the logic analyser which should be connected to an output pin of the device.

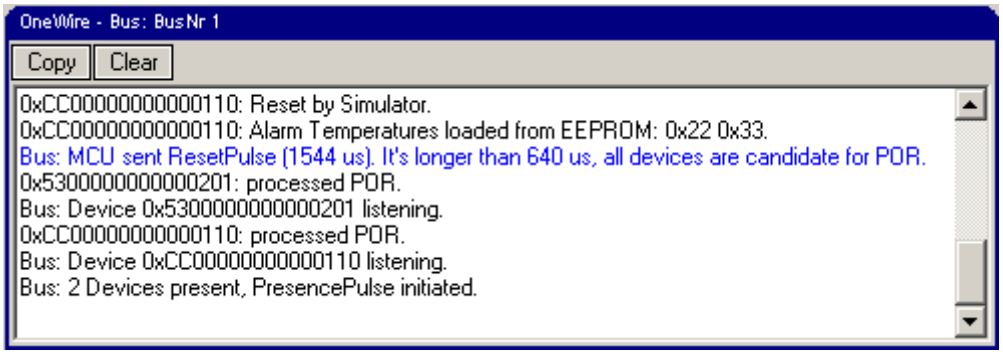
Push Button



The push button models a simple push to make switch. The switch has an input (the common pin) and an output (the contact pin). The switch may be configured to operate as a push button (the button only stays down whilst the mouse is held down over the switch). It may also be set to operate as a push-push switch (the mouse is clicked to push the switch, and clicked again to release it).

Parameters	**Options**	**Notes**
Caption	None, A to Z	The letter which appears on the push button (or a blank may be selected).
Sticky	No, Yes	Set to no for a push button switch, and set to Yes for a push-push switch.
Connections	**Parameters**	**Notes**
Contact	Resistor On Pin	The switch output - this should be connected to a PIC or AVR input. The resistor on pin parameter selects whether the switch output is pulled up or pulled down with a resistor. The switch will default to the pull up or pull down value when released.
Common	None	The input to the switch. It should be connected to a PIC or AVR output, or to high or to low.

OneWire – Bus

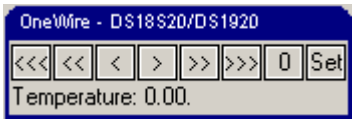


OneWire (also known as 1wire or Microlan) is a bus network system developed by Dallas Semiconductor (Dallas semiconductor has been aquired by Maxim Integrated Products in 2001).

The OneWire bus is the heart of the OneWire simulation system. It serves as the main interface between the MCU and all OneWire devices on this bus. The need for a separate busdevice in the simulator was caused by the inherent structure of the FED simulation system which has no concept of impedance. Next to serving as the only MCU-coupled device, it also serves as central logging system for all devices on the bus. The pinnames are derived from Maxim's application note 244: "Advanced 1-Wire Network Driver".

Parameters	**Options**	**Notes**
BusNr	0-7	The number of the OneWire bus. It is used to connect actual OneWire devices to the correct bus. Up to 8 OneWire buses (= 8 independent networks) can be defined in a single simulation.
Log Level	Disabled Errors Errors + Warnings Errors + Warnings + Info	Select the messagelevels to be displayed in the Log window. Errors are displayed in red, warnings in blue and informational messages in black.
Log Size	From 16 to 1024	The size of the log, expressed in lines.
Connections	**Parameters**	**Notes**
DQ / Sense	None	The main OneWire network connection. It is used to sense the bus by the MCU. When the Drive-pin is not connected, this pin is also to be used to drive the bus.
Drive	Polarity	When it is desirable to deparate the Sense-pin from the Drive-pin, connect this pin to the appropiate MCU-pin. For a true OneWire network, this pin can be left unconnected.
Dynamic PullUp	Polarity	For large networks, a simple resistor pullup may not suffice. Use the Dynamic PullUp (DPU) pin to provide a better alternative. Leave it unconnected if not needed. Of course, the simulator has no knowledge of the actual circuitry that is going to be attached. But the simulator will check signal timing on this pin. Usage of this pin is optional.
Strong PullUp	Polarity	Several OneWire devices from time to time need more power then can be delivered through the OneWire pullup resistor. This additional power is delivered by a special Strong PullUp circuit, activated by this pin. Of course, the simulator has no knowledge of the actual circuitry that is going to be attached. But the simulator will check signal timing on this pin. Usage of this pin is optional.
Program Pulse	Polarity	Activating this pin will pull the bus to a 12V level needed by several devices to program their EEPROM's. Of course, the simulator has no knowledge of the actual circuitry that is going to be attached. But the simulator will check signal timing on this pin. Usage of this pin is optional.

OneWire – DS18S20/DS1920



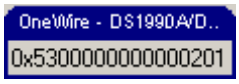
OneWire (also known as 1wire or Microlan) is a bus network system developed by Dallas Semiconductor (Dallas semiconductor has been aquired by Maxim Integrated Products in 2001).

This OneWire device implements all temperature measuring devices that have a familycode 0x10. Buttons are provided to set the temperature. In-/decreases can be done in ticks (16 ticks fit into a centigrade degree), degrees or 10 degrees. The 0-button sets the temperature to zero. When the correct temperature is entered, clicking the Set-button makes the temperature available at the next temperatureconversion initiated by the MCU.

Parameters	Options	Notes
BusNr	0-7	The number of the OneWire bus this device should be connected to.
48-bit serialnr	Entered Value	Every OneWire device has an unique 64-bit factory-lasered ROMcode. It consists of a familycode (0x10 for this device), a 48 bit serialnumber and a CRC. Only the 48 bit serialnumber needs to be entered here.
Power Source	OneWire bus (parasite) External (Self-powered)	The way the device is powered.

Connections	Parameters	Notes
None		The device is connected to the MCU through the OneWire bus.

OneWire – DS1990A/DS2401/DS2411



OneWire (also known as 1wire or Microlan) is a bus network system developed by Dallas Semiconductor (Dallas semiconductor has been aquired by Maxim Integrated Products in 2001).

This OneWire device implements the silicon serial numbers that are identified by familycode 0x01.

Parameters	Options	Notes
BusNr	0-7	The number of the OneWire bus this device should be connected to.
48-bit serialnr	Entered Value	Every OneWire device has a unique 64-bit factory-lasered ROMcode. It consists of a familycode (0x01 for this device), a 48 bit serialnumber and a CRC. Only the 48 bit serialnumber needs to be entered here.
Power Source	OneWire bus (parasite) External (Self-powered)	The way the device is powered.

Connections	Parameters	Notes
None		The device is connected to the MCU through the OneWire bus.

Potentiometer

The potentiometer models a potentiometer connected between 0 and +5V. The output is a number from to the maximum value. The maximum value may be set as between 7 bits (127) and 16 bits (16383). The output of the potentiometer should be connected to a PIC or AVR analogue input and will then be read by the simulator when examining an A/D converter input. The slider value may be changed by dragging it with a mouse, or by using the arrow buttons to set the value accurately.

Parameters	Options	Notes
Maximum Value	From 31 to 16383	The maximum value of the potentiometer which will drive the PIC or AVR input. For an 8 bit A/D the maximum value should be 255, for 10 bit it should be 1023 etc..

Connections	Parameters	Notes
Slider	None	The potentiometer output which should be connected to a PIC or AVR analogue input.

PWM Voltmeter

The PWM voltmeter measures the voltage produced by a PWM output by dividing the high time by the low time over a period of several PWM cycles. The result is displayed on a digital display with a resolution of 1mV.

Parameters	Options	Notes
High Voltage	From 0.5V to 5.0V	The voltage represented by high. For a standard logic output on a standard power supply this would be 5.0V.
Integration Time	From 10us to 1S	The time over which the voltage is measured and therefore the frequency with which the display is updated.

Connections	Parameters	Notes
PWM Input	None	The PIC or AVR output pin which should be connected to the voltmeter.

Shape

The Shape device has no connections, it allows a shape to be drawn on screen, and also allows text comments to be written..

Parameters	Options	Notes
Shape	5 shapes	Select the shape type here. The last shape which is Text Only allows text comments to be written on screen.
Colour	11 colours	Select the shape colour here.
Fill	Yes/No	Should the shape be filled ?
Angle	Rotation	Amount by which to rotate the shape, not all shapes can be rotated.

Serial Terminal

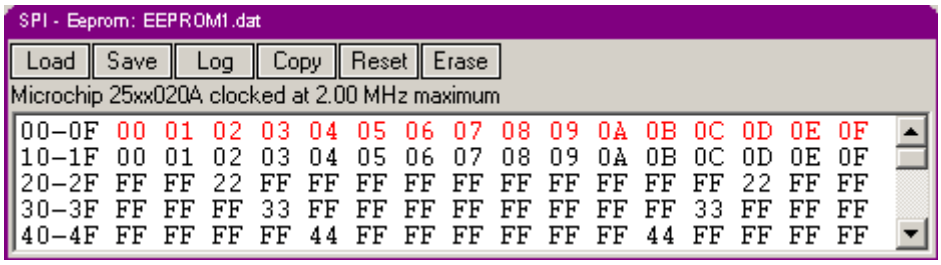
The serial terminal simulates a 3 wire asynchronous serial terminal using RS232 style 8 bit signalling with one start bit and one stop bit. The idle state is high. No flow control is simulated.

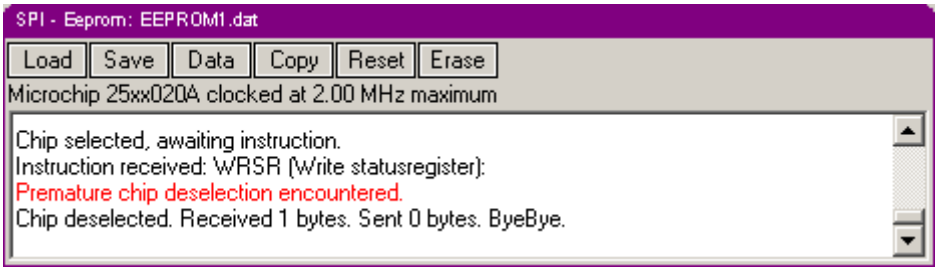
The terminal has 24 lines - although only 2 lines are shown - to scroll up and down click on the terminal window and use the up and down arrow keys. To send a character the terminal is clicked with the mouse and any key presses will be transmitted from the terminal. Similarly any keypresses received by the terminal are shown on the display.

Characters out of normal printing range are shown as a '\' character followed by the decimal value of the character. However Carriage Return (value 13) is shown as '\r' and line feed (value 10) is shown as '\n'. The terminal requires a line feed (Decimal 10) to select the next display line.

Parameters	**Options**	**Notes**
Bit Rate	From 75 to 115000	The serial bit rate of the terminal from 75 bps up to 11500bps. Only standard rates may be selected.
Send CR as CR-LF	Yes, No	If Yes is selected then when the Enter key is pressed a character 13 followed by a character 10 will be sent. Otherwise just a 13 is sent.
2 char delay between	No, Yes	When selected the terminal will send characters separated by a delay equivalent to two characters. This allows time for processing in those programs which do not expect consecutive serial characters to be received.
Display Mode	ASCII, Hex	Received characters are displayed either in ASCII form or as hex number.
Connections	**Parameters**	**Notes**
Terminal Rx	None	The input to the terminal. Note that this should be driven by a PIC or AVR output. (PIC or AVR Tx)
Terminal Tx	None	The output from the terminal, note that this should be connected to a PIC or AVR input (PIC or AVR Rx).

SPI - Eeprom



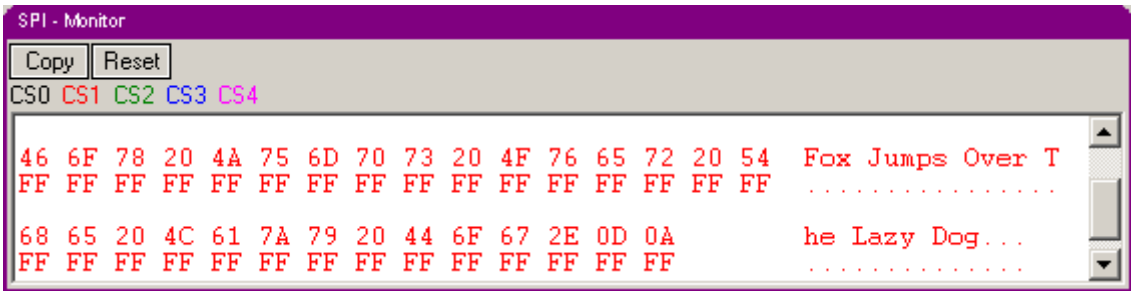


The SPI EEPROM external device is capable of simulating a wide range of SPI serial EEPROM, serial flash and serial FRAM devices. Thru the use of the Load/Save buttons, the EEPROM-data (including the non-volatile bits in the status register) can be loaded from/saved to a disk file. The path to the last loaded/saved file is retained in the WIZC project file to allow automatic reload when re-opening the project. The contents of the EEPROM is displayed in the data view, 1 line per page, in hex. Changed bytes are marked in red. The log view displays all messages generated by the device. These include e.g. command interpretation and execution results. Clicking the Log/Data button toggles between the log view and the data view. The “byte changed” indication can be reset by clicking the Reset button. All bytes of the EEPROM are initialised to 0xFF by clicking Erase.

Parameters	**Options**	**Notes**

Chip Type	List of supported EEPROM, Flash and FRAM chips	The EEPROM chip to be simulated. Additional chips can be defined in the SPIeeprom.ini file.
Log Level	Disabled Errors Errors + Warnings Errors + Warnings + Info	Select the message levels to be displayed in the Log window. Errors are displayed in red, warnings in blue and informational messages in black.
Log Size	From 16 to 1024	The size of the log, expressed in lines.
Connections	**Parameters**	**Notes**
CS	None	Chip Select, low is active.
SCK	None	The SPI clock connection.
SI	None	Serial data input of the EEPROM.
SO	None	Serial data output of the EEPROM.
WP	None	Write Protect pin, low is active.
HOLD	None	Hold pin, low is active

SPI - Monitor

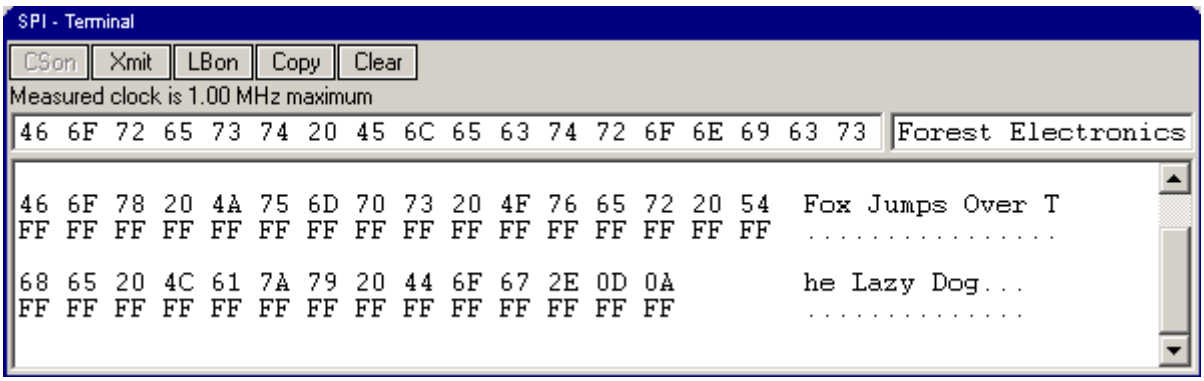


The SPI monitor is intended to show activity on an SPI bus and implements up to 5 slave select signals and two data in signals. Each byte received is displayed as a hex value on the top line or the bottom line depending on which data line the byte was received (The top line shows DataIn0, and the bottom line DataIn1). The colour indicates which slave select signal is active (CS1 in the above picture). Non-ASCII characters are replaced by a period.

The reset button is used to set the monitor back to searching for the first bit of a signal. The Copy button will copy the contents of the monitor log window to the clip board.

Parameters	**Options**	**Notes**
Display Width in characters	4,8,16,32,64	The number of bytes displayed per row on the monitor.
Buffer Size in characters	From 32 to 16384	The number of bytes held and displayed, the default of 256 is normally sufficient for most purposes.
Connections	**Parameters**	**Notes**
ChipSelect 0-4	SPI mode Polarity	The communication between the master and up to five slaves can be monitored. Each slave has it's own SPI mode setting. Refer to the peripheral datasheet for the mode to use. Polarity (normally high) defines the logical level that signals an unselected peripheral.
Clock	None	The SPI clock connection.
DataIn 0	Bit order Polarity	The data which will be displayed on the top row of the monitor.
DataIn 1	Bit order Polarity	The data which will be displayed on the bottom row of the monitor.

SPI - Terminal



The SPI terminal is a complete SPI-device in itself. It is intended to be of help while debugging MCU-software when no external device for your SPI-peripheral is available (yet). Data to transmit can be entered in Hex and/or ASCII. When hex data is typed, the ASCII equivalence is shown v.v. Clicking the Xmit-button transfers the entered data to the transmit buffer. Data will then be transmitted at the first available occasion: When the terminal is operating as a SPI-slave, this moment will be the next byte-boundary while the chip select is made active by the SPI-master. When master-operation is chosen, the buffered data will be transmitted as soon as the chip select is made active by clicking the CSon-button. By clicking LBon, loopback mode is activated: No data can be manually entered in this mode. The Copy-button copies the trace to the clip board. Clearing the buffers and log can be done by clicking the Clear-button. The maximum measured clock speed is displayed.

Parameters	**Options**	**Notes**
SPI Mode	0,0 (CKP = 0, CKE = 1) 0,1 (CKP = 0, CKE = 0) 1,0 (CKP = 1, CKE = 1) 1,1 (CKP = 1, CKE = 0)	The SPI mode to be used by the terminal. Please, be aware that the needed settings for the PIC MSSP do not follow standard SPI practice. (See data sheet)
ClockMode & Speed	Slave Master@Freq.	Define the terminal as a SPI slave or SPI master. Several frequencies can be chosen when master operation is selected.
Input buffersize in characters	From 8 to 1024	The number of characters that will fit into the transmit buffer.
Log buffersize in characters	From 32 to 16384	The number of bytes held and displayed in the log window. The default of 256 is normally sufficient for most purposes.
Log displaywidth in characters	From 4 to 64	The number of bytes displayed per row on the log part of the terminal.
Connections	**Parameters**	**Notes**
ChipSelect	Polarity	As all SPI-devices, the terminal has it's own chip select. Polarity (normally high) defines the logical level that unselects the terminal.
Clock	None	The SPI clock connection.
DataIn	Polarity Bit order	Data receive pin of the terminal. The data will be displayed on the top row of the log window. Polarity and bit order can be selected.
DataOut	Polarity Bit order	Data transmit pin of the terminal. Data is sent from the transmit buffer through this pin. Upon transmission, the data is displayed on the bottom row of the log window. Polarity and bit order can be chosen.

Wire - Digital

The digital wire simply emulates a wire connected between two PIC pins. The colour of the text on the component shows the state of the wire – High red, Low Green or Black Unknown. Any change on one pin will reflect the input on the other. If both are driving then the wire appears as a resistor – each pin will have its own output state and the colour of the text will reflect the most recent change.

Connections	Parameters	Notes
End 0	None	One end of the wire.
End 1	None	The other end of the wire.

Writing external device models

It is possible to create new external devices in DLL files. This requires a C or C++ Compiler, but new devices may be added simply by adding the new libraries to the directory structure. The logic and clock generator devices are supplied in DLL files.

This is an advanced topic which is covered in detail in the DLL authoring manual which is supplied with source files in the Device DLLs sub-directory of the distribution CD.

Working with Variables

The variables window shows the variables which can be examined whilst the program is running. Most actions concerned with watch variables are on the pop up menu of the watch list. Right click on the list to view the menu options.

To add a Watch Variable, then either use the Add Watch menu option under the Simulate Menu, or press the Insert key whilst the window is highlighted. This will bring up the Insert/Add Memory Watch Dialog Box (see below). One or more Watch Variables may be added in the Dialog Box by typing their addresses in label or numeric form separated by commas. The Browse button will display a list from which known variables may be selected. Alternatively right click on a variable name in the edit window and use the menu option to add a watch variable (or press F2).

To trace the watch variable (which allows it to be examined by the waveform analyser), then click the Trace button in the dialog box. The variable will be displayed with a * or a # to show that it is being traced.

To modify a variable in the Variable Window then select it and press enter.

To change the value of a variable in the Variable Window then double click it to bring up the Modify Memory Dialog Box.

To remove a Watch Variable, select it in the Variable Window and press the Delete key.

Insert/Add Memory Watch Dialog Box

The Insert/Add Memory Watch Dialog Box allows the user to enter one or more variables (memory locations) to the Variable Window. If an item is selected in the Variable Window, then the variables will be inserted before the selected item. The Dialog Box contains the following items:

Use C Definition. Professional version only. This check box should be clicked with the C Compiler (or WIZ or AVIDICY programs) if the variable is to examined using the C source as a guide to its content. The scope is also significant so within a block local variables will be shown ahead of global variables with the same name.

Trace. This check box should be clicked if the File Registers which are entered are to be traced during the run for examination in the waveform window (See Running the waveform analyser).

Names or addresses. This edit box should contain one or more expressions for File Registers. If more than one expression is to be entered, then separate items with commas e.g. SPL,SPH,temp,temp+1 will insert four variables into the Variable Window.

Display As. This allows the display shown on the Variable Window to be configured. The display may be set to Bit, Byte, Word, or Long to display the value as a bit, as 1 byte, as 2 byte, or as a 4 byte value stored in memory. The Hex Dump options shows 16 bytes starting at the defined address. The bit option simply shows the value of a single bit of the address, the bit number to be examined should be entered in the edit box.

Display Format. This controls the format of the displayed value. Values may be displayed in binary, decimal or hex. Float and Zero terminated String format may also be used The signed button may be used to display decimal values in signed notation.

Pointer. If this box is clicked then the variable is taken to be a 2 byte pointer to an item defined by the other boxes. The value of the pointer, and the contents of RAM at that location are shown. If the pointer has its top bit set (bit 15) then the pointer is assumed to point to ROM, the top bit is reset to get the ROM address.

Browse. The Browse Button brings up a list of all File Register labels defined for this Program. One or more labels may be selected which will be added to the list of addresses in the Insert File Watch Dialog Box separated by commas when the >> Button is pressed.

How to change the program counter

To change the value of the PC click the PC box shown at the top of the Debug Window.

Alternatively use the **Simulate | Set PC to current line** menu option. This will find the address of the line that the cursor is on within the edit window and set the PC to that line.

When one of the step or run instructions is used the program will now run from the new value of the PC.

Breakpoints

Working with Breakpoints

The *Simulate* | *Add Breakpoints* menu option brings up the Breakpoints Dialog Box which allows the user to configure Program Breakpoints, or click the breakpoint window and press insert, or right click the breakpoint window and use the Add Breakpoint menu option:

Breakpoints Dialog Box

The Breakpoints Dialog Box allows the user to add or change breakpoints in the program. The Dialog Box contains the following items:

Type. There are four types of breakpoint which may be selected
Unconditional Addressed : Enter an address and this breakpoint stops whenever the Program Counter reaches that address.
Conditional Addressed : Enter an address and a condition and this breakpoint stops whenever the Program Counter reaches that address, and the condition is non-zero.
Global Conditional : Enter an address and this breakpoint stops whenever the condition is true, regardless of the value of the Program Counter. This may be used to look for specific faults or conditions. For example enter PIR1&0x40 to stop when bit 6 of the PIR1 register is set to 1.
On Time : Enter a time (e.g. 1.0mS, 250uS, or 2.5S), and the program will stop when the simulation time equals the breakpoint time.

Address. This edit box allows the user to define the address for an unconditional breakpoint or an addressed conditional breakpoint. The address can be any valid PICDE expression, which must be in the range of the processor's program memory space. When the Break on Time button is clicked this box will be re-titled "Time"

Conditional Expression.. This edit box allows the user to define the Conditional Expression for a global conditional breakpoint or an addressed conditional breakpoint. The expression can be any valid PICDE expression which will stop the Program when it evaluates to a Non-Zero result, eg. test ==9. This conditional expression will stop the Program when the value of the File Register test is 9.

Browse. The Browse button brings up a list of program labels. To add a label to the address edit box then select the label and double-click it, or select a label and press the >> button.

Animate. If this checkbox is clicked then the breakpoint is an animation breakpoint. An animation breakpoint does not stop the program, it simply causes the program to update all the watch variables when it is hit. This is most useful if the update slider is at the extreme right, in this event variables are only updated when an animation breakpoint is hit (or when the Update Now button is pressed). This allows the user to follow the value of watch variables when the program passes specific points.

OK. When the OK button is pressed, then all the breakpoints in the breakpoints list, and any breakpoint which is currently being edited, are set on the current Program.

Running the waveform analyser

The waveform analyser operates as a separate program, and allows the user to examine file registers which have been specially marked as for debugging. All of the ports are traced automatically. To set a watch variable as a traced variable then right click it and and use the *Modify Watch* menu option. Click the trace box, reset the simulation and run it - it will then be available in the waveform analyser.

Once the variables have been added to the Debug Window with the trace option set, and the simulation has been reset and then run, then the waveform analyser window may be used to examine them. Use the Tools | Waveform Trace menu option to bring up the analyser, which has its own help file and manual.

Waveform Generator

The Waveform Generator is intended to allow users to design complex data and analogue patterns for injection to the pins of the device under simulation. It is a front end for the FED PIC and AVR development tools.

The wizard allows complex data patterns to be input to the PIC or AVR, clocks to be generated, or analogue waveforms to be generated for injection into the A/D converter inputs of the PIC or AVR. The waveform wizard allows a number of stimulus' to be stored together in one file. One of more of these files may be added to the list of project files and will then be included as simulation input when the program is simulated.

Within any of the FED development environments use the **Tools | Run Wave Generator** menu option to start the program. A new wave generator project will be created with the same name as the current project. The wave generator file will be added to the project window which will automatically include the stimulus when the program is simulated.

The Waveform Generator is described in more detail in the introduction to the professional version supplied with the software.

Working with Stimulus and Injection Files

To add or remove a Stimulus File (which contains Commands to set Port Inputs or File Variables to specific values) or Injection Files (which insert Hex Values in the Program), then use the project window.

To define a stimulus file for use in a program then add it to the project. Click the project window and press insert (or right click the window for a menu). Select one or more files in the file open dialog window (they normally have an extension of .STI). Now when the project type dialog box appears select Stimulus as the type of file.

See :

Stimulus Files

Injection Files

Stimulus Files

Stimulus files allow the user to define data which appears as if it is present on the external pins of the Processor, or to change File Registers at specific times in the Program. Up to 16 Stimulus files may be included for any one simulation.

Stimulus files have the format described below:

A time should be given, which may be in the form of a number followed by an optional letter. If only a number is given, then the time is taken to be in nanoseconds. Alternatively, the letters U, M, or S may be placed directly after the number, in which case the time will be taken in microseconds, milliseconds or seconds, respectively. The time may be given at the start of a line with a command, or on a line on its own. Times do not have to be sequential and one file may define time and stimulus information which ends after another file. The time may be given in decimal format.

The following lines all specify the same time - 1mS:

```
0.001S
1m
1000u
1000000n
1000000
```

It is possible to specify a delta value – a number which will be added to the previous time, for example :

```
1m
+100U
```

The time for the next event will now be set to 1.1mS (1100uS).

To trigger an event on a key press the command onkey is given with a letter or digit :

```
onkey 0
onkey A
```

Use onkey instead of a time, now when that key is pressed the following event will be triggered.

The commands available in a Stimulus file are described individually below, they allow a value to be applied to a Register, to a bit of a Register, or clocks or asynchronous data to be injected to a bit of a Register.
Each line contains a command as follows:

time Set the time for an event, follow with n,u,m or S for units all following events will occur at that time. If no units are provided then time is assumed in nS

```
e.g.    1000m
        200n
        1S
        35u
        1.050m
```

Note that times do not have to be in order - see the example at the end.

onkey Set the keyboard key to trigger a following event, note this must be on a line of its own

```
e.g.    onkey 0
        onkey 1
```

name=value Set specified file register defined by name to value. Note if = is given on its own then the last file register is assumed**.**

```
e.g.  1000m PORTA=76h        ; At 1000mS set file PORTA
                             ; to value 76Hex
1500m =0  ; At 1500mS set file PORTA
          ; to 0

onkey a
PORTA:0=1 ; Set PORT A bit 0 to 1 when key 'a' is pressed
```

name:bit=value Set specified bit of file register defined by name to value. Note if = is given on its own then the last file register and bit set is assumed. (Note if the last assignment was to a file register then the complete file register is assumed).

```
e.g.
          1000m PORTA:0=1    ; At 1000mS set file PORTA, bit 0
                             ; to 1
          1.5mS =0           ; At 1500mS set file PORTA, bit 0
                             ; to 0
          STATUS:3=0         ; At 1500mS set file status, bit 3
                             ; to 0
```

serialrate-name:bit=value

Inject value as an 8 bit asynchronous serial stream to the specified file (name), and bit. Note, if a name and bit value are not given then the last port and bit value are assumed. The rate is given in decimal immediately after the word serial.

```
e.g.
          10m serial9600-PORTC:0=65 ; At 10mS start the
                                    ; injection in
                                    ; serial format of the
                                    ; letter A
                                    ; to Port C, bit 0 at
                                    ; 9600bps
          200m serial2400-=66       ; At time 200mS start the
                                    ; injection of letter B to
                                    ; Port C bit 0, note the
                                    ; same port as
                                    ; that used last time is
                                    ; used.
                                    ; bit rate is 2400
```

clock-name:bit=low,high,cycles

Inject a clock to the specified file register and bit on that file register. low and high are the low time of the clock and the high time of the clock. cycles are the number of repeats of the clock. If cycles is 0, then the clock runs forever. The clock stays high when the number of cycles is complete

```
e.g.
        PORTB=0                       ; At time 0 set PORT B to
0

        100u                          ; Next event will occur at
                                      ; time 100uS
        clock-PORTB:0=25u,75u,10      ; This line generates 10
                                      ; clocks, which
                                      ; are low for 25uS, and
                                      ; high for 75uS
                                      ; the clocks are driven to
                                      ; PORT B, bit; 0
        clock-PORTB:1=25u,75u,0       ; This clock runs forever
```

RTCC and Reset pins

To apply an input to the RTCC pin (the timer 0 input pin), or to apply a value to the reset pin, then instead of a port value use the labels PIN_RTCC or PIN_RESET. The lines below apply 100 20uS clocks to the RTCC (external timer 0) input pin, and then reset the processor at time 10mS, at time 12mS the reset is removed.

```
6m clock-PIN_RTCC=10u,10u,100
10m PIN_RESET=0
12m PIN_RESET=1
```

Analogue values

To apply an analogue input to any of the A/D input pins, then the same format may be used, however the value supplied is from 0 to 0FFFH, and represents the result of the A/D conversion. The higher values are only available for the 10 and 12 bit converters.

```
1m PORTD:0=080H   ; equivalent to 2.5V
2m PORTD:1=0C0H   ; equivalent to 3.75V
```

Injection Files

Injection Files allow the user to define hex data which is applied to a specified Register when the Program counter reaches a specified address. This is particularly useful in the situation where PICDESIM does not fully emulate the behaviour of the real device, and a value can be injected as if the hardware of the real device was operating correctly.

To define an injection file for use in a program then add it to the project. Click the project window and press insert (or right click the window for a menu). Select one or more files in the file open dialog window (they normally have an extension of .HEX). Now when the project type dialog box appears select Injection as the type of file. Now it is possible to define the program address, and the memory address at which the data will be injected. Any normally valid PICDE expression may be used.

The format of an Injection File is a number of hex bytes, each of which is on a line of its own, the hex data should be defined using digits 0 to 9 and letters A - F. Following the number then a comment may follow provided that it is separated from the number with a space. When the end of the file is reached then the file will be reset and data will be read from the start again. To aid layout of the file it is possible to insert blank lines where required.

For example:

```
00
FF
FE
00
```

This will insert the four bytes 00, FF, FE and 00 repeatedly and in turn.

Simulation accuracy

The instruction set and architecture for the basic PIC and AVR is accurately simulated.

A large number of peripherals are simulated as shown in the table below. Where devices are not simulated their registers are available and are reset to the correct values. The injection files technique may be used for these devices to emulate operation of real hardware.

Peripheral	PIC	AVR	Notes
Watchdog	Y	Y	
Timer 0	Y	Y	
Timer 1	Y	Y	
Timer 2	Y	Y	
Timer 3	Y	Y	
Timer 4	Y	Y	
CCP (up to 4 channels)	Y	Y	
PWM	Y	Y	
Enhanced PWM	N	N/A	Normal PWM modes for the Enhanced PWM hardware are supported
External Interrupts	Y	Y	
Internal Interrupts	Y	Y	
USART	Y	Y	2nd USART is also simulated
EUSART	N	N/A	
Analogue Comparators	N	Y	
A/D Converters	Y	Y	
Data EEPROM	Y	Y	
Program Memory EEPROM (read and write)	Y	Y	
Peripheral Service Port	Y	N/A	
Remappable Port Pins	Y	N/A	

Sleep	Y	Y	
I2C	N	N	This is known as TWI on the AVR
SPI (MSSP for PIC)	Y	N	
CAN	N	N/A	All CAN registers are correctly reset and may be written or read.
USB	Y	N/A	All USB registers are correctly reset and may be written or read, the simulation is covered fully in the USB library optional purchase.

WIZ-C

PIC Microcontroller
C based visual development for Windows
Reference

WIZ-C is a well featured visual development environment for Windows '95, '98 and NT4.0.
The program, and its support files and example programs are
© Copyright Robin Abbott, 2000. <robin.abbott@fored.co.uk>.
The program may be installed onto the hard disk of only one Personal Computer, and must removed by deleting the executable file, and all the support files before installing onto a different computer.

Forest Electronic Developments
12 Buldowne Walk
Sway
HAMPSHIRE
SO41 6DU
Sales : +44 - (0)1590 - 681511

info@fored.co.uk

Or see the **Forest Electronic Developments** home page on the world wide web at the following URL:

http://www.fored.co.uk

Contents

Introduction
The Application Designer within WIZ-C
The Application Designer - concepts
Using the Application Designer with existing FED PIC C projects
Starting an application
Selecting PIC and oscillator frequency
Selecting and deleting elements
Setting Element parameters
Defining Occurrences
Connecting pins to the PIC
Listing public calls and variables
Generating the application for the first time
User code and the main loop - in brief
Structure of generated application
User defined code
Processors with multiple ROM and/or RAM pages
Macro Reference
Including the definitions header in your own source files

Introduction

Welcome to WIZ-C, FED's unique drag & drop, point and click development of our popular PIC C development program. Please run through our tutorial first to see some of the capabilities of the program, this manual describes the operation of WIZ-C, and explains some of the procedures that you will need to follow to make best use of the program.

The Application Designer within WIZ-C

The Application Designer is the centre for selecting software library functions, connecting them to the PIC, and defining user defined functions which are called when events occur within the library function. The Application Designer has its own window within WIZ-C. To bring this window to the front use the **Window | Application Designer** menu option, or use the button which is at the extreme right hand side of the tool bar.

The application designer may be enabled or disabled by using the **Project | Use Application Designer** menu option. By default it will be enabled with a new project.

The Application Designer - concepts

The application designer holds software elements in groups at the top of the window. A software *Element* is a library subroutine, or software component, which may be used within an application. The application designer allows software elements to be selected for use within the current application.

Elements are extremely easy to generate using the *Element Editor*. Existing software components or libraries may be converted into software elements using the Editor which is described in its own manual.

When an element is selected for use in the application, the application designer will automatically hook it into the application, and set up the element according to the parameters selected within the designer. The application designer allows an element to be connected to any of the inputs or outputs of the PIC. Some elements have no connections to the pins of the PIC, others have a number of connections.

Some software elements also allow the user to define software functions to be called automatically when events occur. An event is described in the Applications designer as an *Occurrence*.

Examples of occurrences are the internal timers overflowing, a byte being received on a serial interface, for a seven segment multiplexed display changing digit.

Some elements include other elements automatically, and hook their functions automatically into occurrences of the element which they include. An example of this is the HMS element, which generates an occurrence every second, minutes, and hour that passes. This element automatically includes the Timer 0 element which drives the hardware for timer 0. The HMS element automatically hooks its functions into the overflow occurrence of Timer 0.

Using the Application Designer with existing FED PIC C projects

It is not recommended to use the application designer with existing projects until the user is very familiar with the operation of the designer.

By default the application designer will be enabled when an existing project is opened. It is recommended that it is disabled by using the **Project | Use Application designer** menu option when an existing project is opened.

Starting an application

A new project is started in WIZ-C by using the **Project | Open/New Project** menu option. A file open dialog box will shown, it is recommended that a new project is started in its own directory. As soon as the file name is selected and the OK button is pressed, the blank project will be created and the application designer is presented as the top window.

It is recommended that before any user specific code is written, that the application designer is used to create the minimum application. Then user code files can be inserted into the project window.

Selecting PIC and oscillator frequency

The first actions with a new project are to set up the PIC type and the oscillator frequency. The PIC type is selected with the "Change PIC" button under the element groups, the oscillator frequency may be selected from a list, or the exact value may be typed into the box. The oscillator frequency is entered in Hertz. Thus a 4MHz crystal should be entered as 4000000, however as this is a standard value it may be selected from the drop down box.

Selecting and deleting elements

To select an element then select the tab which holds the group from which you wish to make a selection. When the mouse is held over an element a help box will appear showing the name of the element and some information about it. The elements may be double clicked, dragged on to the PIC or the right button may be used to bring up a pop up menu which will allow the selected elements to be inserted.

Some elements are the unique, in that only one copy of the element may be used at any time. Other elements, such as the level detection elements which operate on a single pin of the PIC, may be selected as many times as necessary for the application. If an element is selected which may only be used once, then attempts to select additional copies of the element will fail. Elements which may only connect themselves to a certain pins of the PIC, such as the PWM element will automatically connect themselves to the PIC when they are selected.

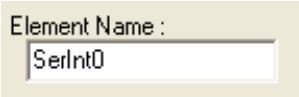
The elements are all shown in the element store at the bottom of the Application Designer window. Elements may be selected from the element store by clicking it with the mouse, a small red box will be shown around the selected element. The selected element has its parameters, Occurrences, its list of public calls and variables, and the connections to the PIC shown in the main application designer window. The element may be defined within the application as described below.

Setting Element parameters

The parameters tab will display a list of parameters which may be changed for the selected element. Parameters may be selected as a number, or as an item from a list, or as a simple on or off selection by a check box. The full meaning of the parameters are described in the help file which may be shown for an element by right clicking it with a mouse, and using the help menu option.

Name Elements

To name an element then use the ElementName box at the top of the application designer:



Any unique name can be entered, this name will now be defined in the header file to a unique number for each element and can be used and tested – particularly in the occurrence functions.

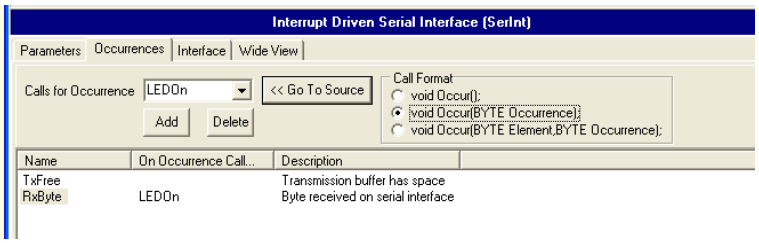
Defining Occurrences

An *Occurrence* is an event which happens, which is detected by an element. It is possible to define user functions which will be called automatically when an occurrence happens.

An Occurrence may be defined as one of:

```
void Occur();
void Occur(BYTE Occurrence);
void Occur(BYTE Element,BYTE Occurrence);
```

The type of call is selected in the Application Designer when the Add button is pressed :



In this example the function LEDOn is defined as being of the second form where the function will be called with the Occurrence Number.

The first form (void Occur();) is the most common and used for the situation where only one function is assigned per occurrence.

The second form (void Occur(BYTE Occurrence);) is really useful for when an element has multiple occurrences which may be handled by a single function an example is the Touch element which has up to 16 occurrences. Each occurrence can call the same function for efficiency. The occurrence parameter to the function will be set to the number of the occurrence. In this example Number 0 will be key 0, number 1 will be key 1 and so on.

The final version (void Occur(BYTE Element,BYTE Occurrence);) is used when the same function can be called my more than one element, such as for push button elements. The Element parameter is set to the element number which triggered the function, the occurrence is the occurrence number. The Element can be tested using the Element name – recall that if an element is named as SwitchUp for example then this will be define to the unique element number. E.g.

```
void SwitchPress(BYTE Element,BYTE Occur)
{
 unsigned char Switch;

 switch(Element)
 {
  case SwitchUp : Switch=1; break;
  case SwitchDown : Switch=2; break;
  case SwitchEnter : Switch=3; break;
 }
 //
 // More code here
 //
}
```

To do this select the occurrence tab on the application designer for the selected element. The main list box will show a list of occurrences for the current element, click one of these occurrences to select it. Type the name for the assembler label at the start of the function into the box, and click the add button to include that function in the list of functions which will be called automatically when the occurrence happens.

To delete a call from the list of occurrences select the Occurrence in the main list box, then select the call to be deleted from the drop down box at the top of the window, and click the Remove button.

When an element, such as the HMS element, automatically loads another element, it may include its own functions within the list of calls which are to be executed when the occurrence. It is important that the user should not delete any of these calls, as they will prevent the element which hooks in the selected element from operating correctly.

Connecting pins to the PIC

Many elements may be connected to pins of the PIC. When an element is selected from the element store, it will be shown on the PIC outline, together with its inputs and outputs shown as pins alongside the elements within the main outline. To connect a pin from an element to a pin of the PIC, then click the pin of the PIC, and then the click the pin of the element to which it is to be connected. If the pin can be connected to that pin of the PIC then the outline will be re-drawn and the connection shown on the main PIC outline.

It is possible to connect more than one element to the same pin of the PIC, for example the LCD display driver allows the data lines to be shared with others inputs. If it is not possible to connect the element pin it to the PIC them an error box will be displayed explaining the reasons why the connection is not possible. For example the seven segment display driver element requires all segment drivers of the display to be connected to the same port, although any port may be selected for this functions.

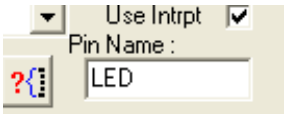
To break a link then the connected pin of the PIC, or the element should be clicked to select the connection. The right mouse button may be used to bring up the menu, the break link option selected, or the break the link button on the Connect Pins tab used.

The PIC is shown in diagrammatic form within the application designer window. If the Connect Pins tab of the Application Designer is selected, then the outline of the PIC will be drawn in a wide form, allowing the pin names of the element, and the PIC to be read. If any of the other tabs is selected, then the outline will only show pin names on the PIC itself. The pin names shown on the PIC are those defined by Microchip, or when connected to an element, show the names of the element inputs/outputs.

The pins of the PIC will take on the name of the element pin to which they are connected. It is possible to use a customised name - for example the PortOut element sets a PIC pin to be an output and names it Out0, the second PortOut element output will be called Out1 etc.

It may be better and easier to understand to give them customised names - "StatusLED" and "MotorDrive" for example. The application designer will now create defines for these pins called StatusLED – which is a bit variable which can be assigned and tested directly and also constant integers - StatusLEDPort and StatusLEDBit which can be used within the program.

To name the pins, then select it by clicking the pin on the PIC or the element and then use the Pin Name box on the top of the Application Designer window - it will be shown on the PIC graphic as it is typed :



When an application is generated, the application designer will check that all element pins which must be connected are correctly connected to the PIC, if this is not the case then it will not be possible to go ahead and assemble the application.

Listing public calls and variables

The interface tab of the application designer shows all the public calls and variables which may be used for the selected element. This information is of use to the programmer of the main code.

To automatically insert a blank call template for a C function for an element then press Alt and Enter whilst the cursor is in the Edit window, use the element Calls/Var option to bring up a list. Double click the required call.

Generating the application for the first time

Once the initial set of elements for the application have been selected, and the parameters, inputs, outputs, and occurrence calls have been defined, the application may be generated. To do this use the pop-up menu option, **Generate Application** within the application designer window, or use the main menu option **Assemble | Generate application**, or press Ctrl+F9, or use the tool bar button.

The application designer will automatically generate three or four files and then go on to compile the application (this is described in the Compiler reference manual).

The first of these is the application designer header file. It has the name ProjectName_Auto.h where ProjectName is the name selected for the project when it was first opened. the second is the main application file named ProjectName_Main.c, which includes the initialisation code, and the main interrupt routines. It should be included in all files added to the project.

The next file is the user file. It has the name ProjectName_User.Inc. Initially this simply contains blank functions for initialisation, interrupts and main loop. Blank functions are created for occurrence calls. This is the only automatically generated file which may be modified as it will only be generated once. The functions are described in detail below.

The final file is the library file which includes all the library routines from the various library files, they are all gathered together here. It has the name ProjectName_Lib.Inc. It will not be generated if the elements use no code for library files.

The include, main and library files should not be altered at all, as they will be completely rewritten every time that the Generate Application option is used.

These files are automatically included within the project window so they will be assembled in order. Additional C files may be added or inserted freely.

After the application designer has generated the main files, and included them within the project window, the Compiler Options dialog box will be bought up allowing the project to be compiled. All of the C options may be set with this box as described in the main C reference, or simply click the Help button.

User code and the main loop - in brief

All user code should be written in the ProjectName_User file, or additional files. Additional code may be added into the project in c files. To add additional files to the project window then click on a file and press insert, select a file and it will then be included before the file which was selected. Alternatively de-select all files and press insert to add a new file at the end. It is recommended that files be inserted between the _User and _Lib files. If the file is not found then it will be created. Further details are in the PICDESIM example and on-line reference.

The user file contains user generated code for initialisation, interrupts, and the main loop. The main loop is code which the PIC executes in turn - testing for occurrences, calling any attached sub-routines, and calling any library routines which need to be called regularly. User defined code may be attached to the main loop, but should return to the main loop to continue the main functions of the program.

Attaching user defined code to the ProjectName_User file is straightforward. Double click the file in the Project Window and it will open in the Edit window. The various are described in comments in the file - and in detail below.

Structure of generated application

The Application Designer generated program follows the flow shown below:

```
void main()
{
                    Initialise essential registers
                                v
                                v
                        Initialise Elements
                                v
                                v
    Set I/O pins to correct state using TRIS statements
                                v
                                v
Initialise Elements - code added after TRIS registers are set up
                                v
                                v
                        Enable Interrupts
                                v
                                v
                    Call function UserInitialise()          // ANY USER INITIALISATION CODE
                                v
                                v
 while(1)                                   // MainLoop
 {                              v
                      For Each Occurrence
                                  Test Occurrence Flag
                                v
                                v
          Call functions associated with occurrence
                                v
                                v
Call any Element functions which are to be called regularly
                                v
                                v
                     Call function UserLoop()           // USER CODE HERE
                                v
                                v
                                v
 }
}

void Interrupt()                                        // Interrupt Routine
{
                 Priority Element interrupt code
                                v
                                v
                  Other Element interrupt code
                                v
                                v
                   Goto label - UserInterrupt           // Assembler level USER INTERRUPT CODE
                                v
                                v
}
```

User defined code

The Application Designer automatically generates a file called ProjectName_User.ASM. This file is only generated once when the Generate Application option is first used, and thereafter may be modified at will. The blank generated file is shown below:

```
//
// This file includes all user definable routines. It may be changed at will as
// it will not be regenerated once the application has been generated for the
// first time.
//

// ******************************************************************************
//
// Insert your interrupt handling code if required here.
// Note quick interrupts are used so code must be simple
// See the manual for details of quick interrupts.
//

void UserInterrupt()
```

```
{
   // Insert your code here

 #asmline goto UserIntReturn     ; PIC Assembler - go back to interrupt routine
}


// ******************************************************************************
//
// Insert your initialisation code if required here.
// Note that when this routine is called Interrupts will not be enabled - the
// Application Designer will enable them before the main loop
//

void UserInitialise()
{
}

// ******************************************************************************
//
// Insert your main loop code if required here. This routine will be called
// as part of the main loop code
//

void UserLoop()
{
}

//
// User occurrence code
//
```

There are three functions here :

void UserInitialise()

Any user defined initialisation code should be written in here, it will be called immediately before the main loop.

void UserInterrupt()

Control jumps to this label during an interrupt. Any code needed in the interrupt can be written here following which control should return to the main interrupt routine. Note the return which is handled in assembler. Please read the C Compiler manual with particular reference to the allowable code for Quick Interrupts which are used for WIZ-C.

void UserLoop()

The main loop undertakes any processing necessary for occurrences and then jumps to UserLoop. Any user defined routines which need to be called regularly can be included here.

Including the definitions header in your own source files

The application designer defines a macro which has the name of the Automatically generated header file with all the element definitions within it.

```
_APPWIZ_AUTO
```

This value is included and set to the name of the header file which defines the values and elements in a WIZ-C project. To include the header file in your own source file then simply use # include :

```
#include _APPWIZ_AUTO     // Include App Designer header
```

This header file is included automatically in the _main.c, _user.c and _lib.c files.