MPLAB Tutorial

If you are new to MPLAB IDE and MPLAB SIM, this tutorial should help you understand the basics of using the integrated development environment to set up your application for development and of using the simulator to debug your application code. You will set up a simple application for the PIC16F84 device in a single project, single workspace environment.

MPLAB IDE Features and Installation

MPLAB IDE is a Windows OS based Integrated Development Environment for the PICmicro MCU families and the dsPIC Digital Signal Controllers. The MPLAB IDE provides the ability to:

- Create and edit source code using the built-in editor.
- Assemble, compile and link source code.
- Debug the executable logic by watching program flow with the built-in simulator or in real time with in-circuit emulators or in-circuit debuggers.
- Make timing measurements with the simulator or emulator.
- View variables in Watch windows.
- Program firmware into devices with device programmers (for details, consult the User's Guide for the specific device programmer).

Note: Selected third party tools are also supported by MPLAB IDE. Check the release notes or readme files for details.

System Requirements

The following minimum configuration is required to run MPLAB IDE (6.20):

- PC-compatible Pentium class system
- Microsoft Windows 98 SE, Windows 2000 SP2, Windows NT 4.0 SP6, Windows ME, Windows XP
- 64 MB memory (128MB recommended)
- 85 MB of hard disk space
- Internet Explorer 5.0 or greater

Install/Uninstall MPLAB IDE

To install MPLAB IDE on your system:

Note: For some Windows OS's, administrative access is required in order to install software on a PC.

• If installing from a CD-ROM, place the disk into a CD drive. Follow the onscreen menu to install MPLAB IDE. If no on-screen menu appears, use Windows Explorer to find and execute the CD-ROM menu, menu.exe. • If downloading MPLAB IDE from the Microchip web site (www.microchip.com), locate the download (.zip) file and click on it to save it to your PC. Then, unzip it and execute the resulting file to install.

To uninstall MPLAB IDE:

- Select <u>Start>Settings>Control Panel</u> to open the Control Panel.
- Double-click on Add/Remove Programs. Find MPLAB IDE on the list and click on it.
- Click **Change/Remove** to remove the program from your system.

Running MPLAB IDE

To start the IDE, double click on the icon installed on the desktop after installation or select <u>Start>Programs>Microchip MPLAB IDE vx.x>MPLAB IDE vx.x</u>. A screen will display the MPLAB IDE logo followed by the MPLAB IDE desktop.

Tutorial Overview

In order to create code that is executable by the target PICmicro MCU, source files need to be put into a project and then the code is built into executable code using selected language tools (assemblers, compilers, linkers, etc.). In MPLAB IDE, the project manager controls this process.

All projects will have these basic steps:

- Select Device. The capabilities of MPLAB IDE vary according to which device is selected. Device selection should be done before doing anything else on a project.
- Create Code. Then source code will be written to the file.
- Create Project. MPLAB Project Wizard will be used to Create a Project.
- Select Language Tools. In the Project Wizard the language tools will be selected. For this tutorial, the built-in assembler will be used. For other projects built-in linker or one of the Microchip compilers or other third party tools might be set.
- Put Files in Project. Only one file will be put into the project, a source file.
- **Build Project**. The project will be built causing our source files to be assembled into machine code that can run on the selected PICmicro MCU.
- **Test Code with Simulator**. And finally, the code will be tested with the simulator.

The Project Wizard will guide us through most of these steps and will make this series of tasks easy.

Note: Some aspects of the user interface will change in future releases and the screen shots in this tutorial may not exactly match the appearance of the MPLAB IDE desktop in later releases.

Select Device

To show menu selections in this document, the menu item from the top row in MPLAB IDE will be shown after the menu name like this <u>MenuName>MenuItem</u>. So to choose the <u>Select Device</u> entry in the <u>Configure</u> menu, it would be written as <u>Configure>Select Device</u>.

Choose <u>Configure>Select Device</u>.

Figure: Configure>Select Device

| ** | MPLAB | IDE ve | 5.62 | | | | | | | | | |
|-----------|--------------|--------------|-----------|---------------|-------------|---------|---------------|-----------------------------|-----------------------|------|------|--------|
| Eile | <u>E</u> dit | <u>V</u> iew | Project | <u>D</u> ebug | jger Pro | grammer | <u>T</u> ools | <u>C</u> onfigure | <u>W</u> indow | Help | | |
| | 🗅 🖻 | | Х 🕨 | | 8 A | ? | d* 🖻 | Select <u>D</u> Configur | evice ration Bits. | | sum: | 0×3bff |
| E | Untitl | ed Wo | rkspace | | IX | | | Externa | Memory., | , | | |
| IC | | utput | | | | | | ID Memo | ory | | | |
| | Buil | d Ve | rsion Con | trol Fir | nd in Files | | | <u>S</u> ettings | | | | |
| | | | | | | | | | | | | |
| | | | | | PIC16F84 | F [| | W:0 | z dc | c | | |

In the Device dialog, select the **PIC16F84** from the list if it's not already selected.

Figure: Select Device Dialog

| Select Device | × |
|-----------------------------|--------------|
| De <u>v</u> ice: | |
| PIC16F84 | • |
| Microchip Programmer Tool S | Support |
| 🥝 PICSTART Plus 🧉 🥝 | MPLAB ICD 2 |
| 🥥 PRO MATE II 🛛 🥥 | PICkit 1 |
| MPLAB PM3 | |
| | |
| Microchip Debugger Tool Sup | oport |
| 🥥 MPLAB SIM 🛛 🥥 | MPLAB ICD 2 |
| MPLABICE 2000 M | PLABICE 4000 |
| РСМ16ХН0 | No Module |
| PCM16XH1 | |
| | |
| O <u>K</u> Cancel | Help |

The "lights" indicate which MPLAB IDE components support this device.

- A green light indicates full support.
- A yellow light indicates minimal support for an upcoming part that might not be fully supported in this release by the particular MPLAB IDE component. Usage of components with a yellow light instead of a green light is often intended for early adopters of new parts who need quick support and understand that some operations or functions may not be available.
- A red light indicates no support for this device. Support may be forthcoming or inappropriate for the tool, e.g., dsPIC devices cannot be supported on MPLAB ICE 2000.

Creating Source Code with the Editor

Select <u>*File*>New</u> to open an empty editor window in which to type your source code. For more on using the editor, see MPLAB Editor Help.

Enter the following, or copy and paste:

| | list | p=16f84 | 4 | inc |
|-------|----------------|--------------|---------|--|
| c1 | equ | 0x0c | ±. ; | Set temp variable counter c1 at address 0x0c |
| reset | org | 0x00 | ; | Set program memory base at reset vector 0x00 |
| | goto | start | ; | Go to start of the main program |
| atoxt | org | 0x04 | ; | Set program memory base to beginning of user code |
| loop | movlw movwf | 0x09 c1 | ; ; | Initialize counter to arbitrary value greater than zero Store value in temp variable a defined above |
| 1005 | incfsz goto | cl,F loop | ; ; | Increment counter, place results in file register Loop until counter overflows |
| | goto end | bug | ; | When counter overflows, got to start to re-initialize |

This code is a very simple program that increments a counter and resets to a predetermined value when the counter rolls over to zero.

Once you have completed entering the code, select *File>Save* and save the file as tutor84.asm. You may save it to any directory; you will move it, if necessary, into the project directory in the next step.

Create Project

Next, we'll create a project using the Project Wizard. A project is the way your files are organized to be compiled and assembled. We will use a single assembly file for this project. Choose the <u>Project>Project Wizard</u>.

From the Welcome dialog, click on Next> to advance in the Project Wizard.

The next dialog (Step One) allows you to change the device, which we've already done. Make sure that it says PIC16F84. If it does not, select the PIC16F84 with the menu.

Select Language Tools

Step Two of the Project Wizard sets up the language tools that are used with this project. Select Microchip **MPASM Toolsuite** in the top pulldown. Then you should see **MPASM**, **MPLINK** and **MPLIB** show up in the Toolsuite Contents box. You can click on each one to see its location. If you installed MPLAB IDE into the default directory, the MPASM assembler executable will be:

C:\Program Files\MPLAB IDE\MCHIP_Tools\mpasmwin.exe

the MPLINK linker executable will be:

C:\Program Files\MPLAB IDE\MCHIP_Tools\mplink.exe

and the MPLIB librarian executable will be:

C:\Program Files\MPLAB IDE\MCHIP_Tools\mplib.exe

If these do not show up correctly, use the browse button to set them to the proper files in the MPLAB IDE subfolders.

Figure: Project Wizard - Select Language Tools

| Project Wizard | × |
|--|------|
| Step Two: Select a language toolsuite | چ |
| Active Toolsuite: Microchip MPASM Toolsuite | • |
| Toolsuite Contents MPASM Assembler (mpasmwin.exe) MPLINK Object Linker (mplink.exe) MPLIB Librarian (mplib.exe) | |
| | 1 |
| | |
| Help! My Suite Isn't Listed! | ites |
| < <u>B</u> ack <u>N</u> ext > Cancel Help | Þ |

Put Files in Project

Step Three of the wizard allows you to name the project and put it into a folder. This sample project will be called tutor84, and using the Browse button, the project will be placed in a folder named My Documents.

| Project Wizard | | | | X |
|----------------------------------|--------|----------------|--------|-----------|
| Step Three: Name your project | | | | <u>چر</u> |
| | | | | |
| - · · · · | | | | |
| tutor84 | | | | |
| Project Directory | | | | |
| C:\My Documents | | | | Browse |
| . <u>.</u> | | | | |
| | | | | |
| | 4 Back | Nouts | Canaal | |
| | | <u>N</u> ext > | Lancel | |

Figure: Project Wizard - Name Project

Step Four of the Project Wizard allows us to select files for the project. Choose the file named tutor84.asm. The full path to the file will be:

C:\My Documents\tutor84.asm

Figure: Project Wizard - Select File

| Project Wizard |
|--|
| Step Four: Add any existing files to your project |
| Add >> tutor84.asm tutor84.CDD tutor84.err tutor84.HEX tutor84.HEX Tutor84.mcs tutor84.mcs tutor84.mcs tutor84.mcs Tutor84 |
| < <u>B</u> ack <u>N</u> ext > Cancel Help |

Press the Add>> button to move the file name to the right panel, and click on the check box at the start of the line with the file name to enable this file to be copied to our project directory.

Make sure that your dialog looks like the picture above, with check box checked, then press the **Next>** button to finish the Project Wizard.

The final screen of the Project Wizard is a summary showing the selected device, the toolsuite, and the new project file name.

After pressing the **Finish** button, look at the Project Window on the MPLAB IDE desktop. It should look like this. If the Project Window is not open, select *View>Project*.

Figure: Project Window

| 🗖 tutor84.mcw | |
|---|--|
| tutor84.mcw tutor84.mcp Source Files Header Files Object Files Library Files Library Files Other Files | |
| | |
| | |

Files can be added and projects saved by using the right mouse button in the project window. In case of error, files can be manually deleted by selecting them and using the right mouse click menu.

Build the Project

Assembling the file can be accomplished in several ways. The method described here uses the <u>*Project>Build All*</u> menu item. This will execute the MPASM assembler in the background. Once the assembly process is complete, the Output Window will appear.

Figure: output window – Build Failed

You have intentionally entered at least one error if you entered the code as written in previously. The last goto in the program references a nonexistent label called bug. Since this label has not been defined before, the assembler reports an error. You may have other errors as well.

Using the mouse, double click on the error message. This will bring the cursor to the line in the source code that contains the error. Change bug to start. Use the Output window to help find the errors, and repair any other bugs in your source code. Reassemble by executing the *Project>Build All* menu function. This process may take a couple of iterations.

To build the project, select either:

- <u>Project>Build All</u>
- Right-click on the project name in the project window and select Build All
- Click the Build All icon on the Project toolbar. Hover the mouse over icons to see pop-up text of what they represent.

The Output window shows the result of the build process. When you've fixed all errors in the source code, the Output window will display "BUILD SUCCEEDED".

Figure: output window – Build Succeeded



You now have a complete project that can be executed using the simulator.

Upon a successful build, the output file generated by the language tool will be loaded. This file contains the object code that can be programmed into a PICmicro MCU and debugging information so that source code can be debugged and source variables can be viewed symbolically in Watch windows.

Note: The real power of projects is evident when there are many files to be compiled/assembled and linked to form the final executable application - as in a real application. Projects keep track of all of this. Build options can be set for each file that access other features of the language tools, such as report outputs and compiler optimizations.

Test Code with Simulator

In order to test the code, we need some kind of software or hardware that will execute the PICmicro instructions. A debug execution tool is a hardware or software tool that is used to inspect code as it executes a program (in this case tutor84.asm). Hardware tools such as MPLAB ICE or MPLAB ICD 2 can execute code in real devices, but if we don't have hardware yet, the MPLAB simulator can be used to test the code. For this tutorial use MPLAB SIM simulator.

The simulator is a software program that runs on the PC to *simulate* the instructions of the PICmicro MCU. It does not run in "real time," since the simulator program is dependent upon the speed of the PC, the complexity of the code, overhead from the operating system and how many other tasks are running. However, the simulator accurately *measures* the time it would take to execute the code if it were operating in real time in an application.

Note: Other debug execution tools include MPLAB ICE 2000, MPLAB ICE 4000 and MPLAB ICD 2. These are optional hardware tools to test code on the application PC board. Most of the MPLAB IDE debugging operations are the same as the simulator, but unlike the simulator, these tools allow the target PICmicro MCU to run at full speed in the actual target application.

Select the simulator as the debug execution tool. This is done from the <u>Debugger>Select Tool</u> pull down menu. After selecting MPLAB SIM, the following changes should be seen.

- 1. The status bar on the bottom of the MPLAB IDE window should change to MPLAB SIM.
- 2. Additional menu items should now appear in the Debugger menu.
- 3. Additional toolbar icons should appear in the Debug Tool Bar.
- **TIP:** Position the mouse cursor over a toolbar button to see a brief description of the button's function.

| MPLAB IDE v6.62 | | | | | | | | | × |
|---|--|----------------------------|---------------|-------------|-----------|--------|---------|---------------|----|
| File Edit View Project | Debugger Programm | ner Too | ols Configure | Window Help | | | | | |
| 🗅 😅 🖬 🐰 🖿 | Select <u>T</u> ool Clear <u>M</u> emory | + | 🛩 🖬 🤴 | 🛥 🅸 🛗 📋 | Checksum: | 0×7434 | D III D | 에 (1) 46 45 4 | £ |
| Lutor84.mcw Lutor84.mcp Source Files Utor84.asm Header Files Ubrary Files Uther Scripts Other Files | Run Animate Halt Step Into Step Over Step Out Reset Breakpoints StopWatch Stimulus Controller Refresh PM Settings | F9 F5 F7 F8 F2 | | | | | | | |
| MPLAB SIM | PIC16F84 | pc:0 | W:0 | z dc c | | bank 0 | | | // |

Figure: MPLAB IDE DESKTOP WITH MPLAB SIM AS DEBUGGER

Next we select <u>Debugger>Reset</u> and a green arrow shows us where our program will begin. The first instruction in memory jumps to the label called *start*, where we put our code. This instruction jumps over the PIC16F84 vector areas in lower memory.

Figure: Debug>Reset

```
C:\My Documents\tutor84.asm
                                                                              - 🗆 ×
                 p=16f84
           list
           include <pl6f84.inc>
       cl equ 0x0c
                         ; Set temp variable counter cl at address 0x0c
           org 0x00
                         ; Set program memory base at reset vector 0x00
       reset
    4
           goto start
                          ; Go to start of the main program
                           ; Set program memory base to beginning of user code
           org 0x04
       start
           movlw 0x09
                           ; Initialize counter to arbitrary value greater than zero
           movwf cl
                          ; Store value in temp variable a defined above
       100p
           incfsz cl,F ; Increment counter, place results in file register
                  loop
                          ; Loop until counter overflows
           goto
           goto
                           ; When counter overflows, got to start to re-initialize
                   start
           end
                                                                                ►
```

To single step through the application program, select <u>*Debugger*>Step Into</u>. This will execute the currently indicated line of code and move the arrow to the next line of code to be executed.

There are shortcuts for these commonly used functions in the Debug Tool Bar.

TABLE: DEBUG SHORT CUT ICONS

| Debugger Menu | Toolbar Button | Hot Key |
|---------------|-----------------------|---------|
| Run | D | F9 |
| Halt | 00 | F5 |
| Animate | DD | |
| Step Into | [} | F7 |
| Step Over | 0 + | F8 |
| Step Out Of | በት | |
| Reset | | F6 |

Click on the appropriate icon on the toolbar or use the hot key shown next to the menu item. This is usually the best approach for repeated stepping.

Now press the Step Into icon or select <u>Debugger>Step Into</u> to single step to our code at Start.

Figure: DEbug>Step Into

| 🔲 C:\My I | Docur | nents\tut | or84.asm | | | |
|-----------|-------|--------------------|--|------|--|-------|
| | | list include | p=16f84 <p16f84< th=""><th>. i:</th><th>nc></th><th>-</th></p16f84<> | . i: | nc> | - |
| | cl | equ 0x0 | c | ż | Set temp variable counter cl at address 0x0c | |
| | | org 0x0 | 0 | ż | Set program memory base at reset vector 0x00 | |
| | res | et | | | | |
| | | goto | start | Ż | Go to start of the main program | |
| | | org 0x0 | 4 | ż | Set program memory base to beginning of user code | |
| | sta | rt | | | | |
| | | movlw | 0x09 | 2 | Initialize counter to arbitrary value greater than | zero |
| | | mowwf | cl | 2 | Store value in temp variable a defined above | |
| | 100 | р | | | | |
| | | incfsz | cl,F | 2 | Increment counter, place results in file register | |
| | | goto | loop | ż | Loop until counter overflows | |
| | | goto end | start | 2 | When counter overflows, got to start to re-initial | ize 💌 |
| | | | | | | • |

Opening Other Windows for Debugging

There are many ways to look at your program and its execution using MPLAB IDE. For example, this program is intended to increment a temporary counter, but how do you know for sure that is happening? One way is to open and inspect the file register window. Do this by executing the <u>View>File Registers</u> menu item. A small window with all of the file registers, or RAM, of the PIC16F84 will appear. All special registers can be inspected by selecting <u>View>Special Function Registers</u> menu item.

Press <F7> (Execute Single-Step) a few times and watch the values update in the file register window. We put the counter variable at address location 0x0C. As the temporary counter is incremented, this is reflected in the file register window. File registers change colors when their value changes so that they can easily be noticed on inspection. However, in very complex programs, many values may change, making it difficult to focus on one or two variables. This problem can be solved by using a Watch window.

Select <u>View>Watch</u> to bring up an empty Watch Window. There are two pull downs on the top of the Watch Window. The one on the left labelled "Add SFR" can be used to add the Special Function Register, wREG, into the watch. Select WREG from the list and then click **Add SFR** to add it to the window.

Figure: Watch - Select WREG

| 🔲 Watch | | | | | × |
|---------|--|--|-----------|-------|---|
| Add SER | WREG | - Add Sym | nbol16F84 | • | |
| Add | PCLATH PORTA PORTB STATUS TMR0 TRISA TRISB WREG | <pre>with the second se</pre> | ol Name | Value | |
| Watch 1 | Watch 2 | Watch 3 W | /atch 4 | | |

The pull down on the right, allows us to add symbols from our program. Use that pull down to add the cl variable into the Watch Window. Select cl from the list and then click **Add Symbol** to add it to the window.

Figure: Watch - Select variable "c1"

| 🔲 Watch | | | |
|-----------------|---------------|--|----------|
| Add SER WREG | ▼ Add Symbol | c1 | • |
| Address | Symbol : | | _ |
| | WREG | DC EEIE EEIF F GIE INTE | • |
| Watch 1 Watch 2 | Watch 3 Watch | 14 | |

The watch window should now show the address, value and name of the two registers.

Note: You also may add items to the watch window by either dragging them from the SFR, File Register or Editor window or directly in the window under symbol name and typing in the item.

We could continue single stepping through this code, but instead we'll set a breakpoint just before the first increment of the temporary counter c1. Set a breakpoint by putting the cursor on the line and clicking the right mouse button.

| E:\My Documents\tutor84.asm | | |
|----------------------------------|--|-----------------------------------|
| loop incfsz cl,F goto loop | Add Filter-in Trace Remove All Filter Traces | lace results in file register |
| goto start end | Remove Filter Trace Add Filter-out Trace Close Set Breakpoint Breakpoints Run to Cursor Set PC at Cursor Cut Copy Paste Delete Add To Project Advanced Bookmarks | ws, got to start to re-initialize |
| | Properties | |

Figure: Debug Context Menu (Right Mouse Click on Line)

Select Set Breakpoint from the context menu. A red "B" will show on the line.

Figure: Editor Window - Set Breakpoint

| 🔲 C:\My I | ocuments\t | utor84.asm | | |
|-----------|--------------------|---|--|----------|
| | list inclu | p=16f84 de <p16f84< th=""><th>inc></th><th>-</th></p16f84<> | inc> | - |
| | cl equ O | x0c | ; Set temp variable counter cl at address 0x0c | |
| | org O | x00 | ; Set program memory base at reset vector 0x00 | |
| | reset goto | start | ; Go to start of the main program | |
| | org O | x04 | ; Set program memory base to beginning of user co | ode |
| | start | 0 | · Initialize counter to exhit your value greater t | hon govo |
| ⇒ | movwf | cl | ; Store value in temp variable a defined above | Man zero |
| | loop | | | |
| B | incfs | z cl,F | ; Increment counter, place results in file regist | er |
| | goto | loop | ; Loop until counter overflows | |
| | goto end | start | ; When counter overflows, got to start to re-init | ialize |
| | | | | ▶ |

Select <u>*Debugger*>*Run*</u> to run the application. A text message "Running..." will briefly appear on the status bar before the application halts at this first breakpoint.

The watch window should now show that the variable c1 was incremented by one. This would seem to indicate that the program is working as designed. You can single step through the code, or run more times to verify that things are acting properly.

It would be interesting to calculate time spent in loop. We could use the data book to determine how long each instruction would take in our loop and come up with a pretty accurate number. Or we could use the MPLAB StopWatch to measure the time. We're interested in the time our whole loop takes to execute, so if we set another breakpoint on the instruction goto Start, we can measure the time between breakpoints.

Use <u>Debugger>StopWatch</u> to bring up the StopWatch dialog. Make sure that a two breakpoints are set at the incfsz c1, F and at the goto Start instruction, and then press <u>Debug>Reset</u> and then <u>Debug>Run</u> to halt at the incfsz c1, F instruction. With the default processor frequency of 4 MHz, the StopWatch should show that it took 4 microseconds to reach the first breakpoint.

Figure: Stopwatch - at first breakpoint

| Stopwatch | | |
|--|-----------------|--|
| Stonwatch | Total Simulated | |
| Synch Instruction Cycles 4 | 4 | |
| ero Time (uSecs) 4.000000 | 4.000000 | |
| Processor Frequency (MHz) 4.000000 | | |
| Clear Simulation Time On <u>R</u> eset | | |

Now press Zero button in StopWatch window and disable or remove breakpoint at incfsz c1, F instruction. Execute Run again to go around the loop, and note that the StopWatch shows that it took about 740 milliseconds. To change this, you can change the initial value in the c1.

Figure: Stopwatch - after delay

| Stopwatch | |
|--|-----------------------|
| | |
| Stop | watch Total Simulated |
| Synch Instruction Cycles | 740 744 |
| Zero Time (uSecs) 74 | 0.000000 744.000000 |
| Processor Frequency (MHz) | 4.000000 |
| Clear Simulation Time On <u>R</u> eset | |

Using Trace

To view which instructions are executed from a Run to a Halt or Break, use the trace function.

- Select <u>Debugger>Settings</u> and click the **Trace/Pins** tab. "Trace Enable" should be checked by default. If it is not checked, click to check it now. Click **OK**.
- Select <u>View>Simulator Trace</u> to open the Trace window.
- If "Trace Enable" had been checked by default, there will be data in this window when you open it. These are the instructions that executed when you ran to the breakpoint in the previous topic.
- If "Trace Enable" had not been checked by default, there will be no data in this window when you open it. To fill the window with data, repeat the "Using Breakpoints" procedure from the previous topic. When the breakpoint halts the program, click on the Trace window to make it active and view the instructions that executed when you ran to the breakpoint.

For more on the meaning of the data in this window, go to the topic on the Trace window.

Close the Trace window when you are done.

Using Pin Stimulus

To better simulate real-world conditions, such as high/low input to a pin, the simulator provides a mechanism known as stimulus. The first type of stimulus is called pin stimulus which provides synchronous or asynchronous stimulus on I/O pins.

- Select <u>Debugger>Stimulus Controller</u> and click the **Pin Stimulus** tab.
- Click Add Row to add a row for data entry.
- Click in the added row to select it for data entry. Then select "Asynch" for "Type", "RB7" for "Pin" and "Toggle" for "Action".
- Click **Fire** under Enable. Then perform a "Step Into" and see the value of the PORTB change to 0x80 in the Special Function Registers (SFR) window.
- Click **Fire** under Enable again. Then perform a "Step Into" and see the value of the PORTB change to back to 0x00 in the SFR window.

Although this tutorial program does not require pin I/O to function, your own application may require, for example, the pushing of a button to activate a procedure. By using pin stimulus, you can simulate a button push to test your code.

• Save your pin stimulus to a file by selecting Save under "Pin Stimulus file".

You may load this pin stimulus file into another project by using **Load** under "Pin Stimulus file".

Note: Despite its listing in the drop-down list, you cannot load old clock stimulus files (*.sti).

Using File Stimulus

The second type of stimulus is called file stimulus which provides triggered stimulus on I/O pins or file registers, set up from files. There are three types of files used in file stimulus. The hierarchy of these files is as follows:

- File Stimulus file (*.fsti) composed of one or more Synchronous Stimulus files.
- Synchronous Stimulus file (*.ssti) contains information on triggers used for applying stimulus to either a pin or register.
- Register Stimulus file (*.rsti) specifies register stimulus.

When you are creating file stimulus, you should proceed as follows:

Creating a Register Stimulus File

To create a Register Stimulus file:

- Select *<u>File>New</u>* to open a new edit window.
- In the window, enter the following: B0.
- Select *File>Save* and save the file as reg1.rsti.
- Select *<u>File>Close</u>* to close the file.

You will next create a Synchronous Stimulus file which will make use of this Register Stimulus file.

Creating a Synchronous Stimulus File

To create a Synchronous Stimulus file:

- Click the File Stimulus tab of the Simulator Stimulus dialog (*Debugger>Stimulus*).
- Click Add under "Input Files". Enter stim1 for "File Name" and click OK in the Open dialog. The file name should appear in the file list box, i.e., C:\My Documents\stim1.ssti.
- Click on the file name to select the file. Click **Edit** under "Input Files". This will disable some "Input Files" buttons and enable the "Edit Controls" buttons.
- Click Add Row under "Edit Controls". A row will appear for entering data.
- Set up the row for file injection into a register as follows:
 - Select "PC" under "Trigger On".

Note: You must use PC with register stimulus.

- Enter 0x1a under "Trig Value".
- Select TRISB under "Pin/Register".

Note: You cannot inject file values into port registers, e.g., PORTB. Use pin values (explained next.) Also, you cannot inject file values into the W register.

- Select "File" under "Value". This will open a dialog to select a register stimulus file. Select regl.rsti and click **Open** to enter this file under "Value".
- Click Add Row under "Edit Controls". Another row will appear for entering data.
- Set up the row for pin injection as follows:
 - Select "Cycle" under "Trigger On".

Note: You must use Cycle with pin stimulus.

- Enter 14 under "Trig Value".
- Select "Pin:RB4" under "Pin/Register".
- Select "1" under "Value".
- Click **Save** under "Input Files" to save your work. This will disable the "Edit Controls" buttons and enable the "Input Files" buttons.

You will now create a File Stimulus file including just this Synchronous Stimulus file.

Creating a File Stimulus File

To create a File Stimulus file:

• Click **Save Setup** under "File Stimulus". This will save all the files in the file list box as part of a master stimulus file. In this case, there is only one file. Enter file1 for "File name" and click **Save**.

You will now run your program again and observe the affects of the stimulus.

Note: Do Not Close the Simulator Stimulus dialog. You may move it almost off the screen if you need to view other dialogs.

Running with Stimulus

First, you will need to add TRISB to the watch window. Do this by clicking on the watch window, selecting TRISB from the SFR list, and then clicking **Add SFR**.

To run your program and inject the stimulus:

- Select <u>Debugger>Reset</u> to reset your application.
- Select <u>Debugger>Step Into</u> (or click the equivalent toolbar icon) until the PC on the status bar shows a value of 0x1a.
- Step once more to execute the instruction at pc:0x1a and trigger the register stimulus. The value of TRISB should now be 0xB0 instead of 0xF0.
- Step once more to execute the instruction at cycle=14 and trigger the pin stimulus. The value of PORTB should now be 0x11 instead of 0x01.

You could add another row to the ssti file that would toggle the RB4 pin, i.e., "Trigger On" = "Cycles", "Trig Value" = 15, "Pin/Register" = "Pin:RB4", "Value" = 0.