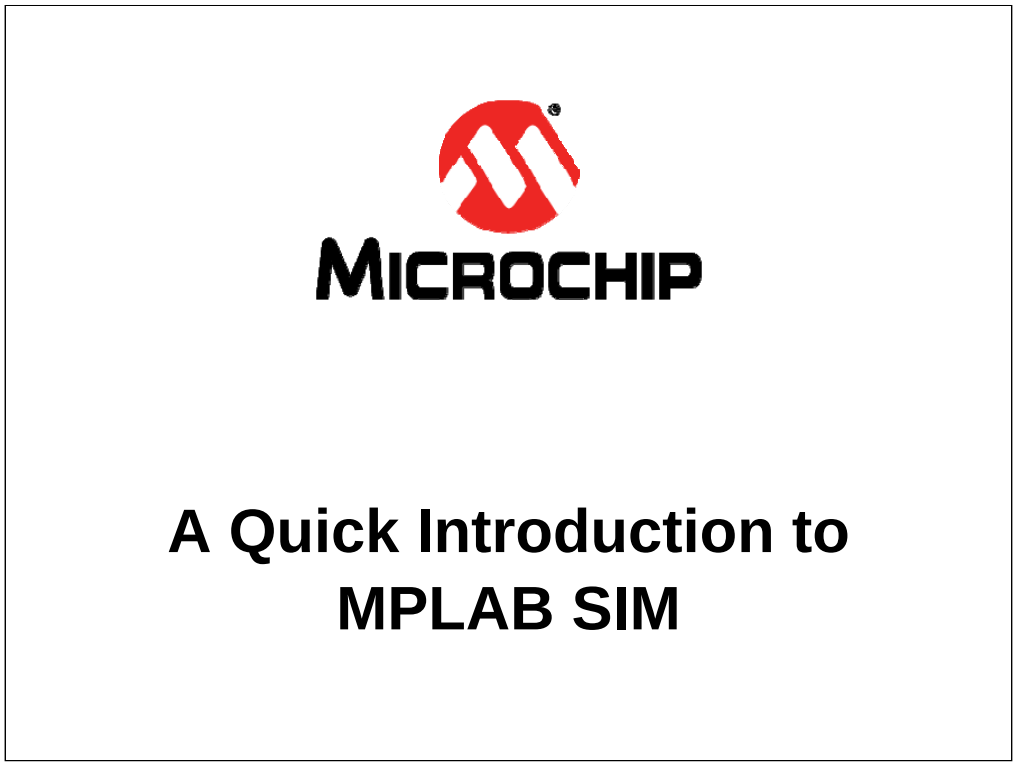
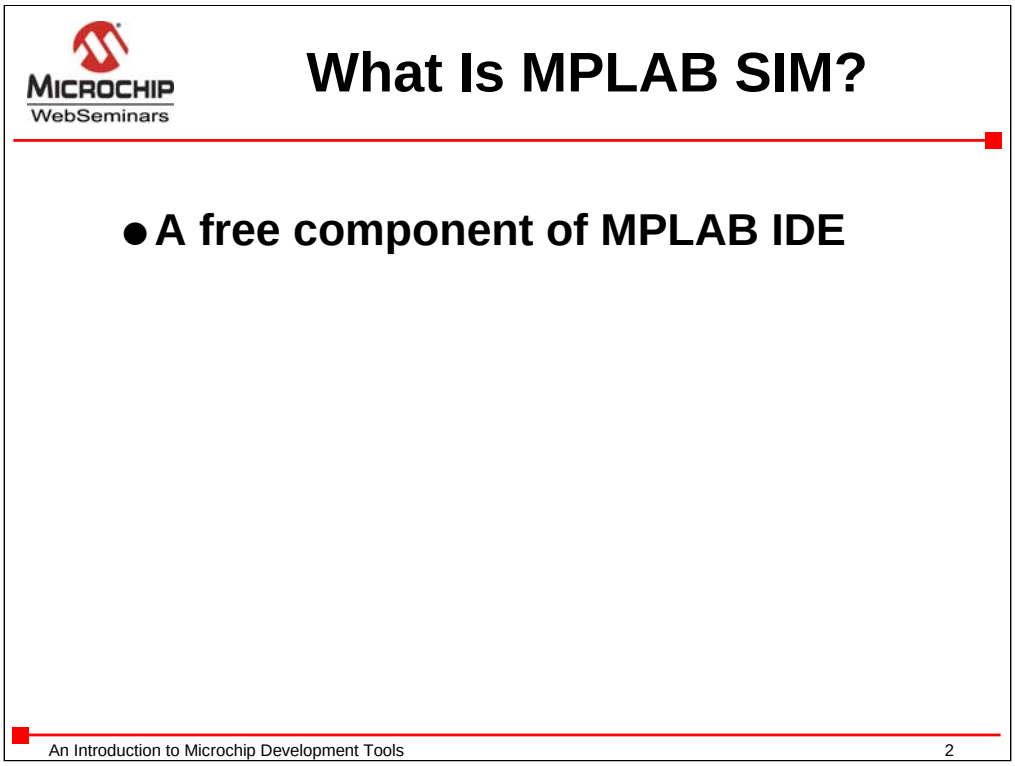


A Quick Introduction to MPLAB SIM

Welcome to this web seminar, “A Quick Introduction to MPLAB SIM.”

My name is Darrel Johansen and I’m a manager in the Development Tools group at Microchip.

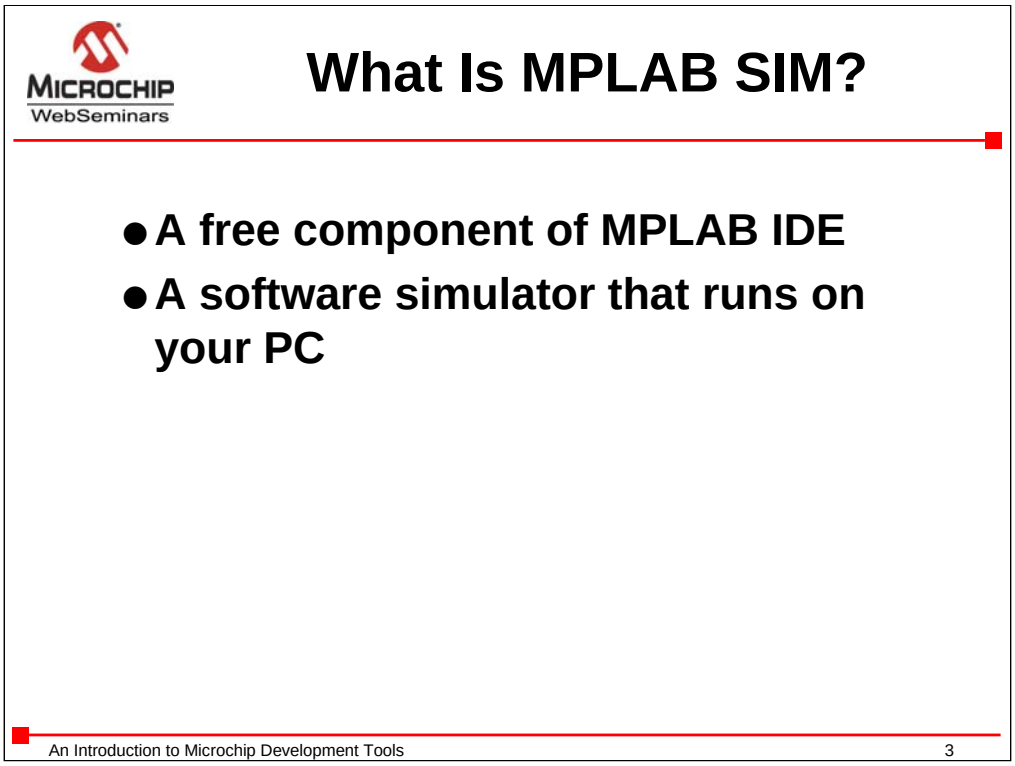


The centerpiece of our tool set is the software MPLAB Integrated Development Environment, or “IDE.”

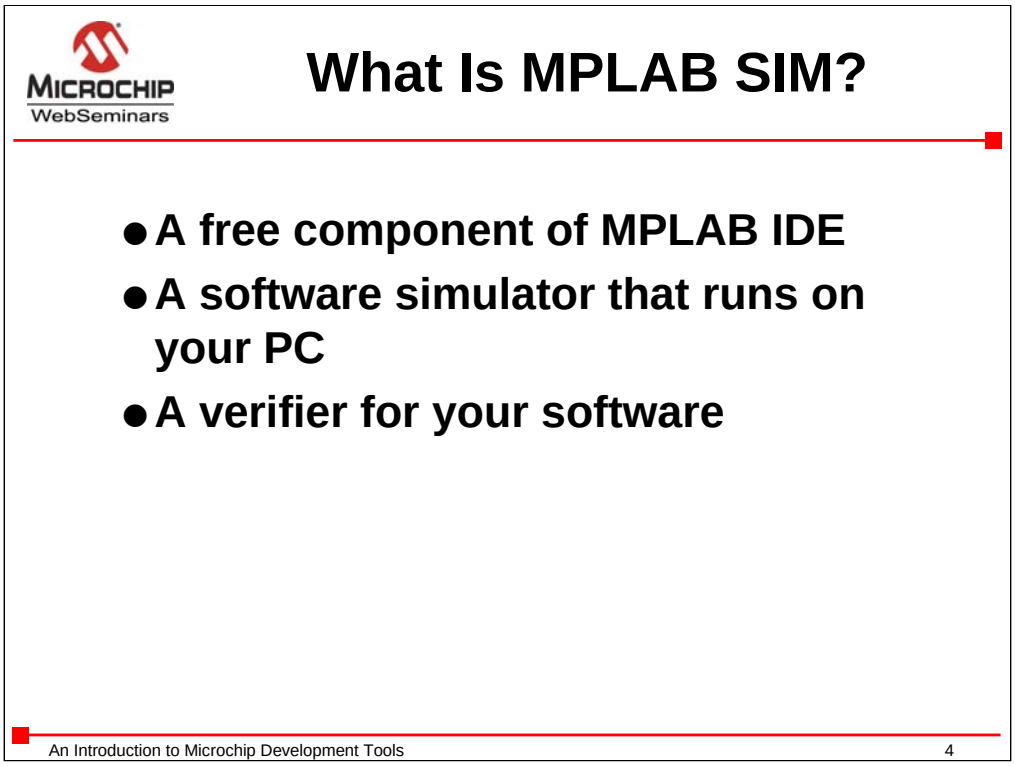
MPLAB IDE has enjoyed many years of use and evolution, tracking Microchip’s expanding catalog of microcontrollers and digital signal controllers.

One of the most popular elements of MPLAB is the software simulator, MPLAB SIM.

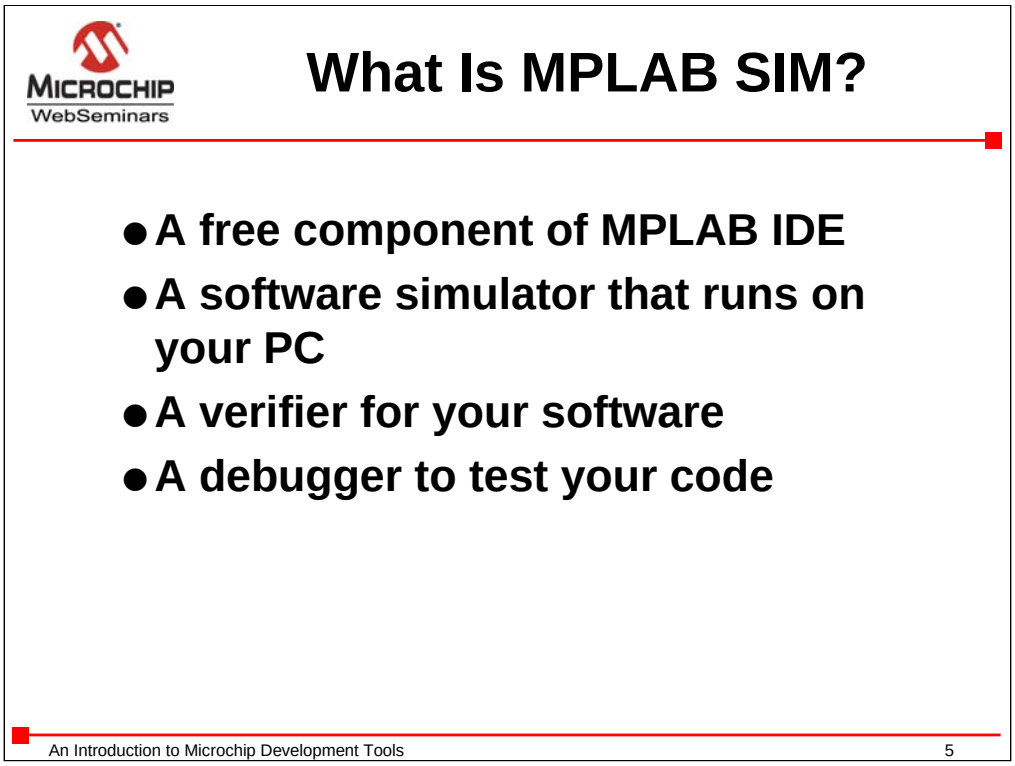
This component, like the MPLAB IDE is free.



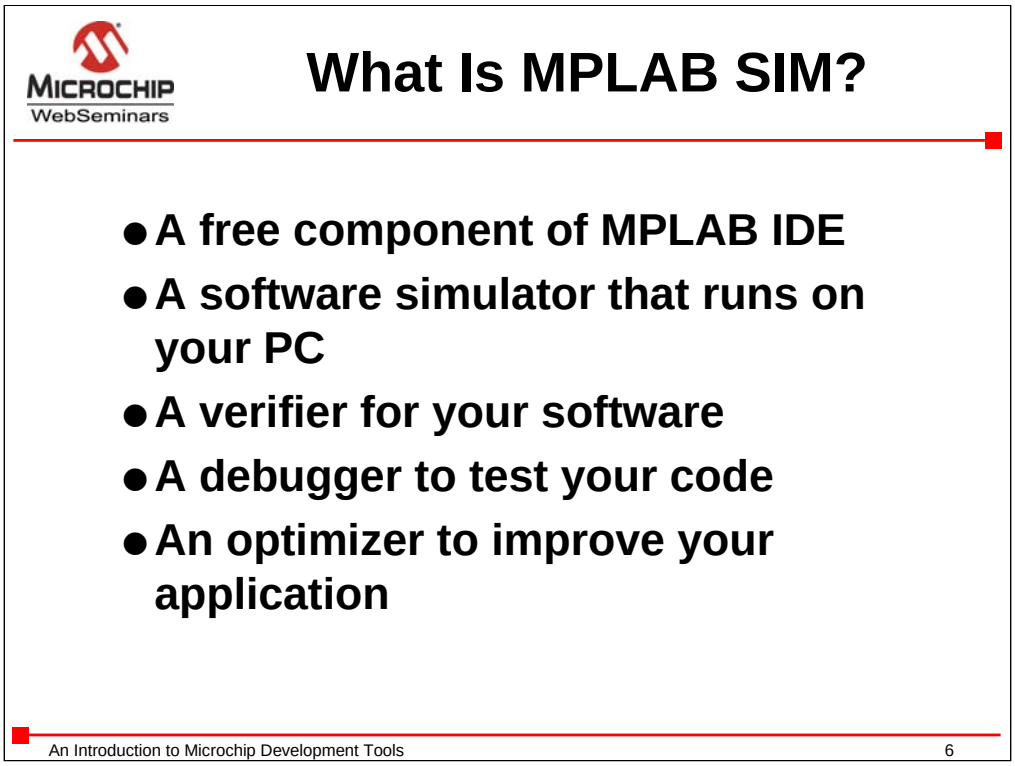
The simulator runs on your PC to simulate the actions of the various PIC and dsPIC microcontrollers.



You can verify your software routines execute as designed…



…and debug, test and inspect your code.

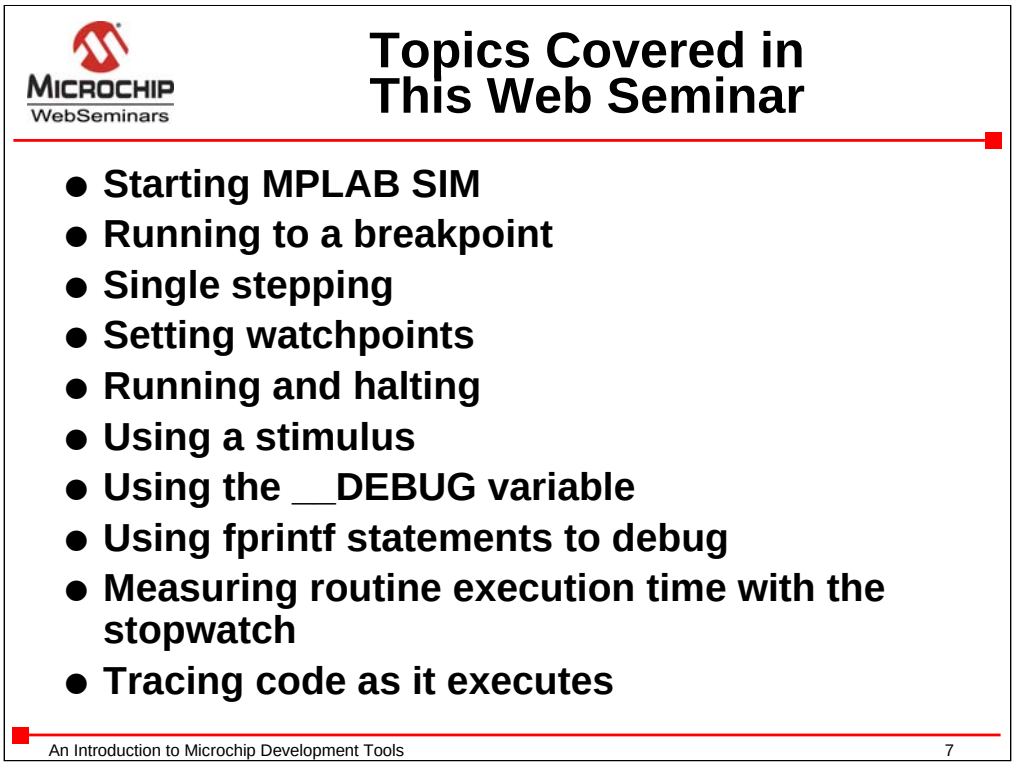


MPLAB SIM helps you optimize your application with

•timing tools,

•trace tools, and

•various graphical analyzers.



We'll cover the basic elements of the simulator in this web seminar, including:

•How to configure MPLAB IDE to use MPLAB SIM as the current debugger

•How to run your code in MPLAB SIM halting at a breakpoint

•How to single step through your code

•How to set watchpoints on data variables in your code

•How to run and halt your code

•How to use a stimulus to simulate external inputs to your hardware

•How to use the __DEBUG variable to make conditional code execution while debugging

•How to insert fprintf statements in your code to monitor the executing routines

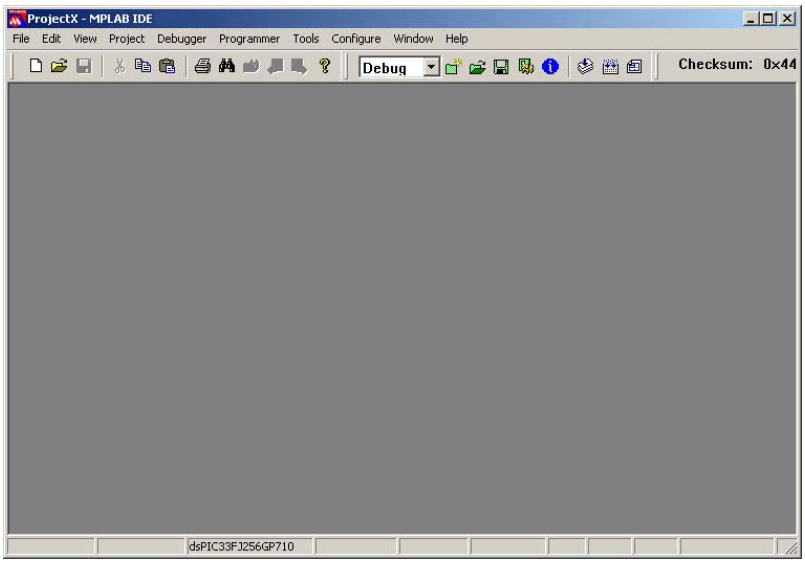
•How to measure the execution time of your code, and

•How to record code as it executes, capturing its history in a buffer to be reviewed.

Topics Covered in This Web Seminar

- **Starting MPLAB SIM**
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

Starting MPLAB SIM
MPLAB Desktop

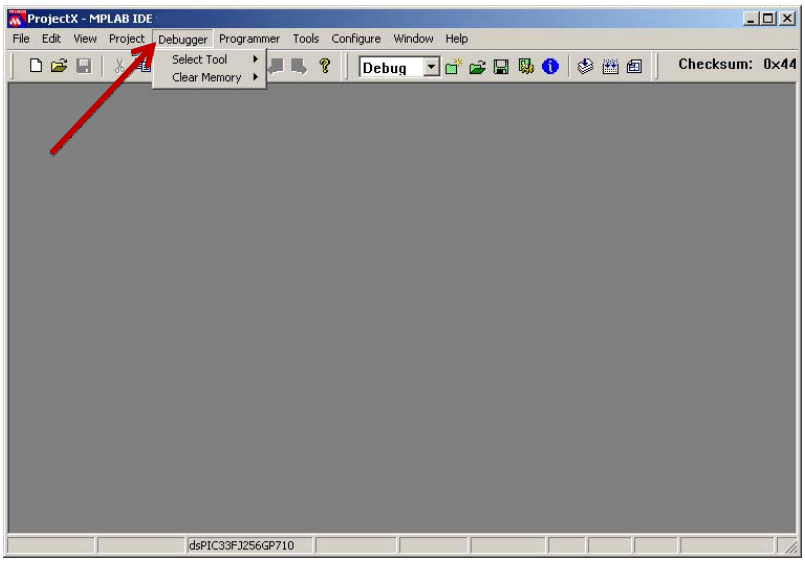


You should already be familiar with MPLAB IDE before going through this seminar. You might want to watch the “Introduction to MPLAB” seminar first.

The MPLAB IDE desktop has all the standard Windows features:

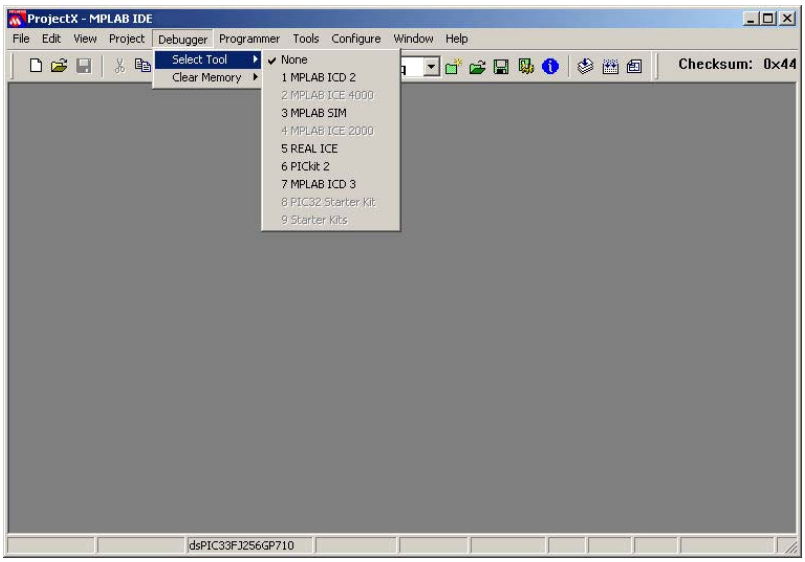
- menus,
- toolbars with icons, and
- a status bar at the bottom.

Starting MPLAB SIM
Debugger>Select Tool



To select the simulator, select the Debugger pull down menu, and scroll to “Select Tool”…

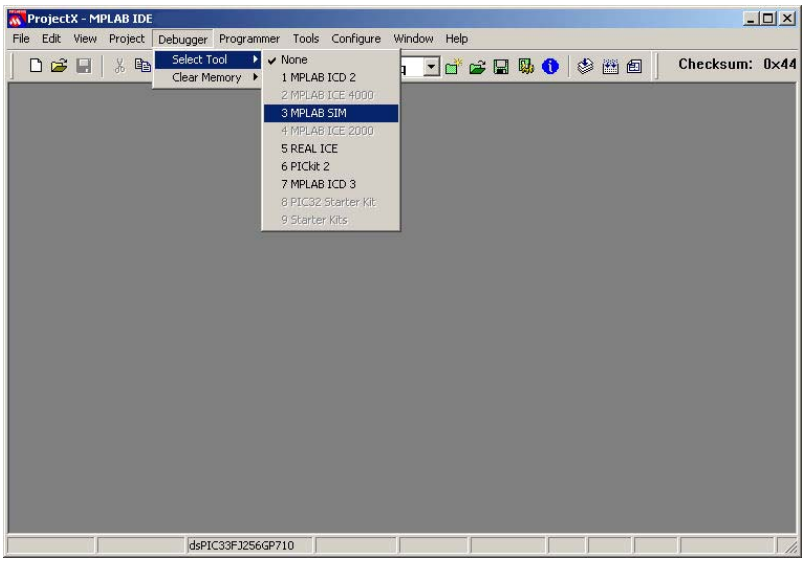
Starting MPLAB SIM
Debugger>Select Tool



A sub menu pops up…

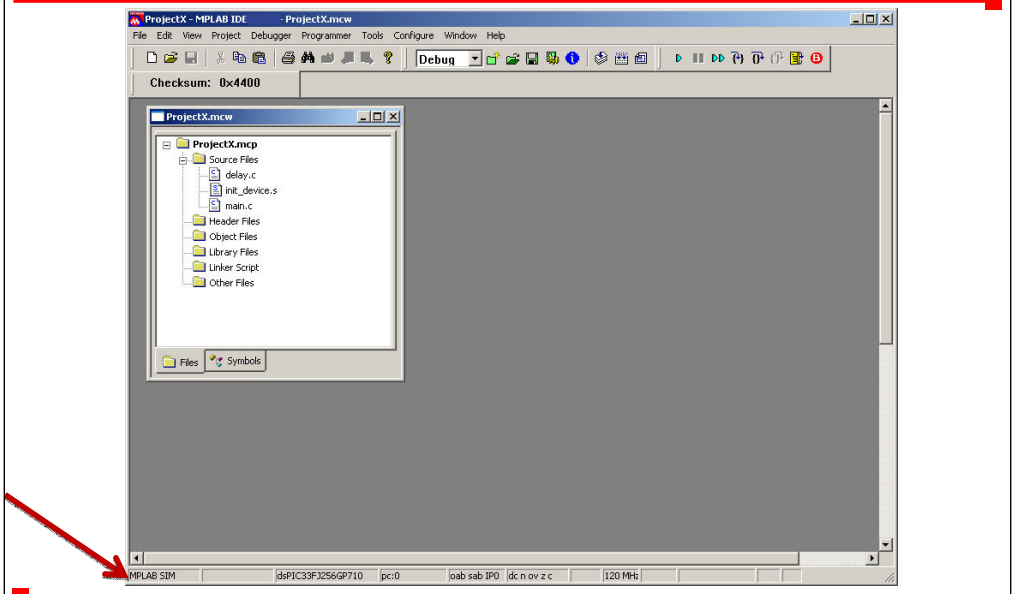
Starting MPLAB SIM

Debugger>Select Tool>MPLAB SIM



Scroll down to select MPLAB SIM.

Starting MPLAB SIM Projects



An Introduction to Microchip Development Tools 13

Now the status bar shows MPLAB SIM as the current debugging tool.

-- MPLAB deals with applications as “projects.” --

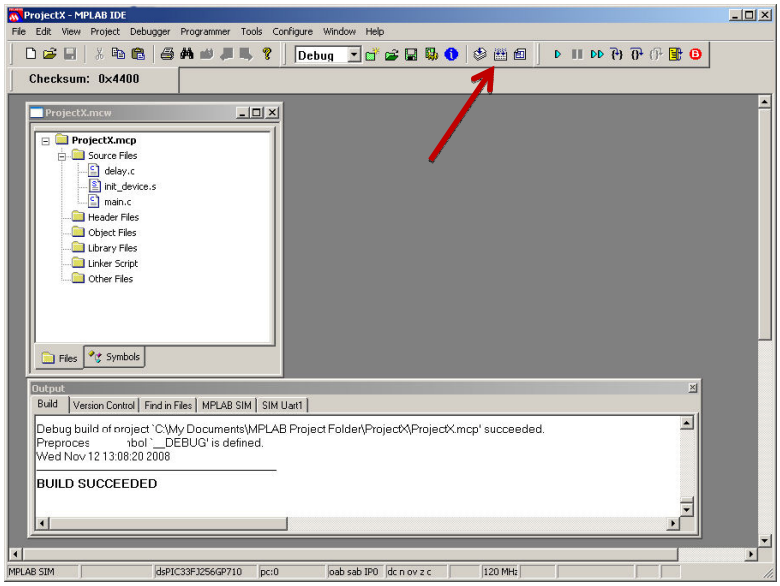
Projects are the source files that are used to build the application along with compilers, assemblers and linkers to “build” the firmware for the application.

In this project are three files, “main.c.” “delay.c,” and an assembly file, “int_devices.s.”

All will be compiled using the MPLAB C compiler and its tool suite for the dsPIC family of digital signal controllers, but this process is the same for the other families of Microchip microprocessors.

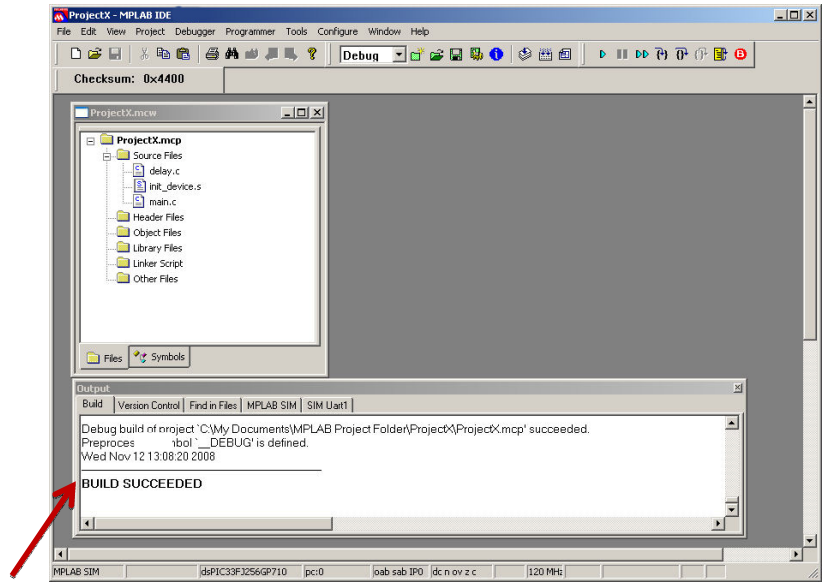
For more information on setting up a project, see the “Introduction to MPLAB” seminar referenced earlier.

Starting MPLAB SIM
Build Project



When all the source code is correctly written and the project configured properly, click the build icon on the toolbar to compile the files.

Starting MPLAB SIM
Build Project

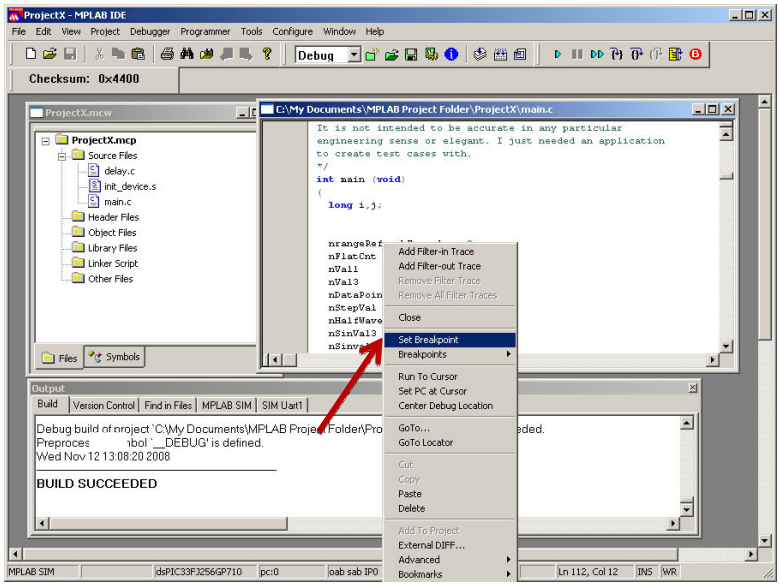


If the project builds with no errors, the output window will show “build succeeded.”

Topics Covered in This Web Seminar

- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

Breakpoint
Set Breakpoint

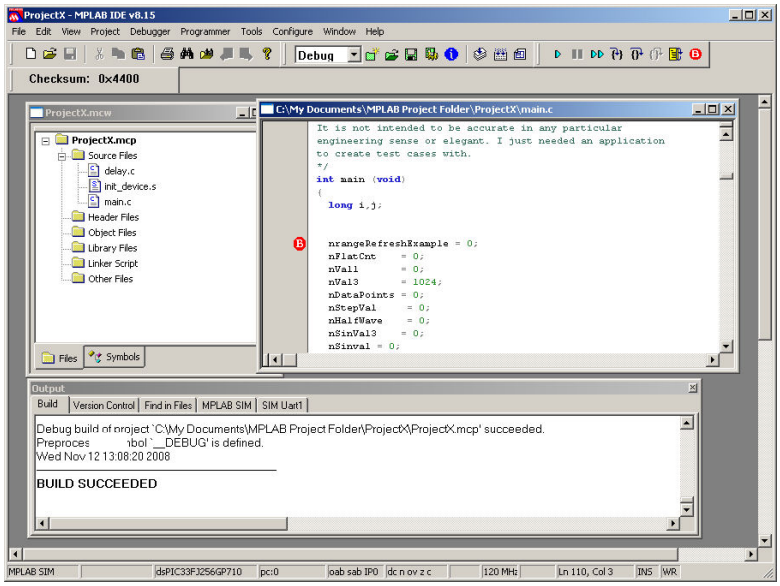


Open a source file for editing or debugging by double clicking on the file name in the project window,

After a project successfully builds, position the cursor on the desired line,

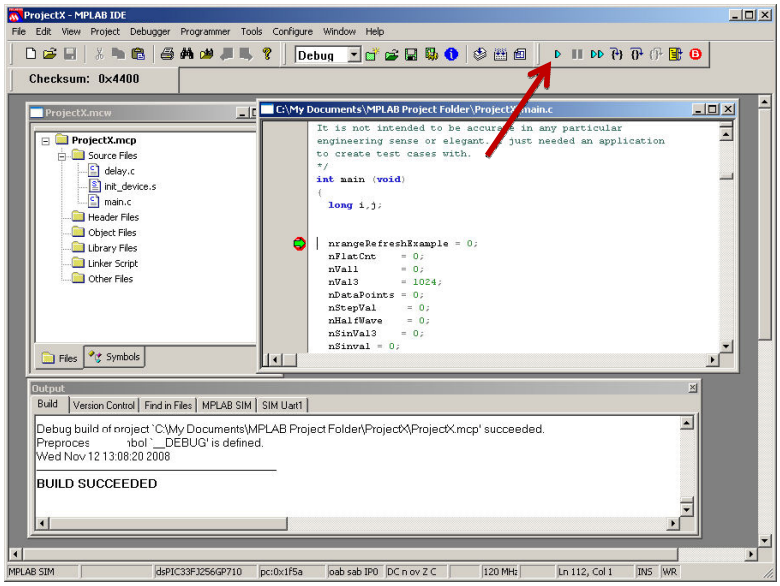
And click the right mouse button to bring up a menu to set a breakpoint.

Breakpoint



In the source file the breakpoint is shown by the red symbol with the letter “B”

Run to Breakpoint



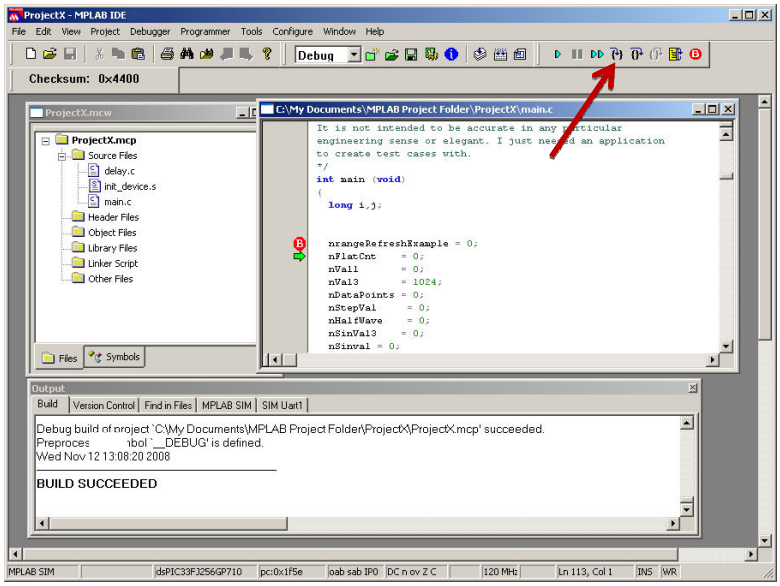
Pressing the “Run” icon, starts the simulation, stopping at the breakpoint.

The green arrow on top of the red “B” shows the current program counter location at the breakpoint set in the function “main.”

Topics Covered in This Web Seminar

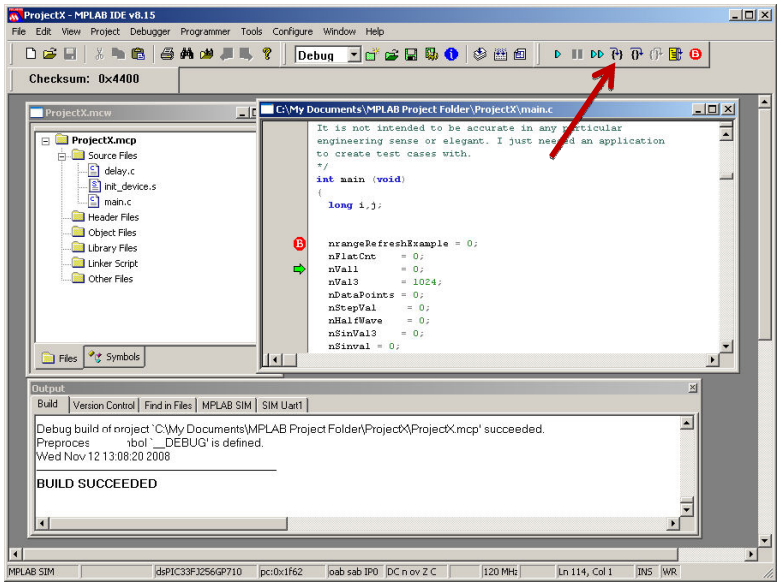
- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

Single Step

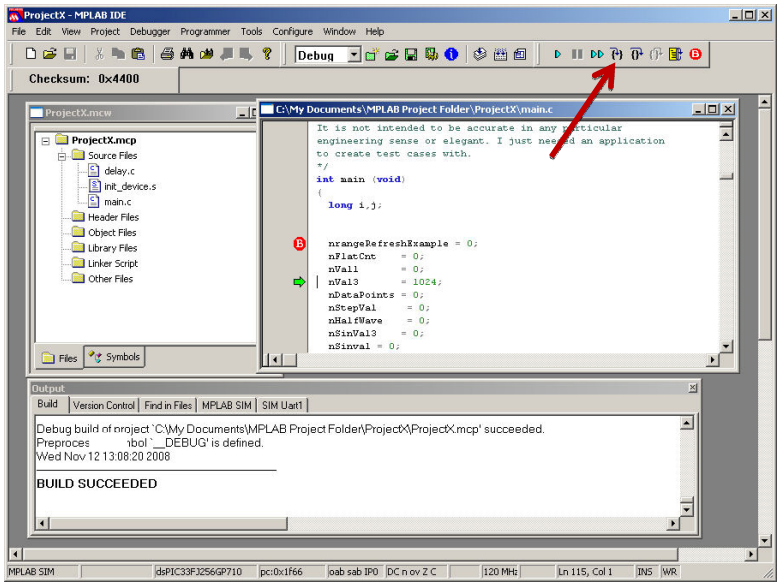


Pressing the “step” icon, single steps one line in the code.

Step Again


ProjectX - MPLAB IDE v8.15
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x4400
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
nSinval = 0;
Output
Build
Version Control
Find in Files
MPLAB SIM
SIM Uart1
Debug build of project `C:\My Documents\MPLAB Project Folder\ProjectX\ProjectX.mcp' succeeded.
Wed Nov 12 13:08:20 2008
BUILD SUCCEEDED
MPLAB SIM
dsPIC33FJ256GP710
pc:0x1f62
oab sab IP0
DC n ov Z C
120 MHz
Ln 114, Col 1
INS
WR

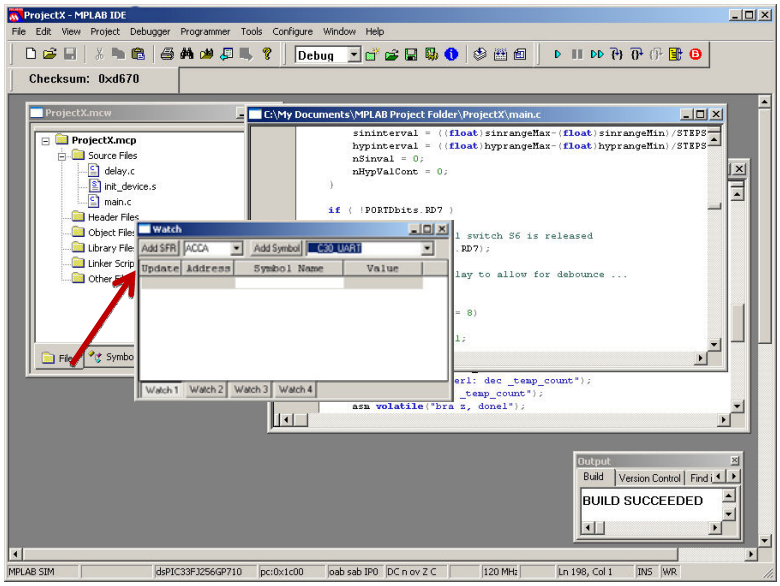
Step Again


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x4400
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
nSinval = 0;
Output
Build
Version Control
Find in Files
MPLAB SIM
SIM Uart1
Debug build of project `C:\My Documents\MPLAB Project Folder\ProjectX\ProjectX.mcp' succeeded.
Wed Nov 12 13:08:20 2008
BUILD SUCCEEDED
MPLAB SIM
dsPIC33FJ256GP710
pc:0x1f66
120 MHz
Ln 115, Col 1
INS
WR

Topics Covered in This Web Seminar

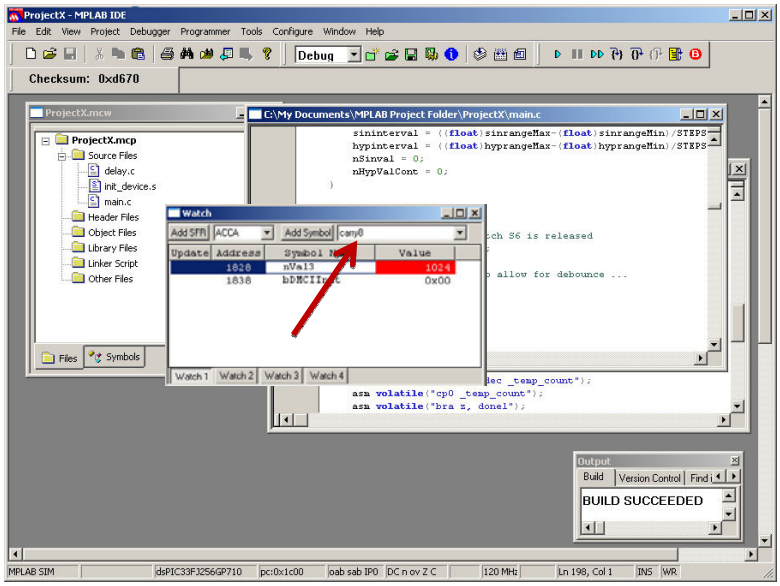
- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

Watchpoints
View>Watch



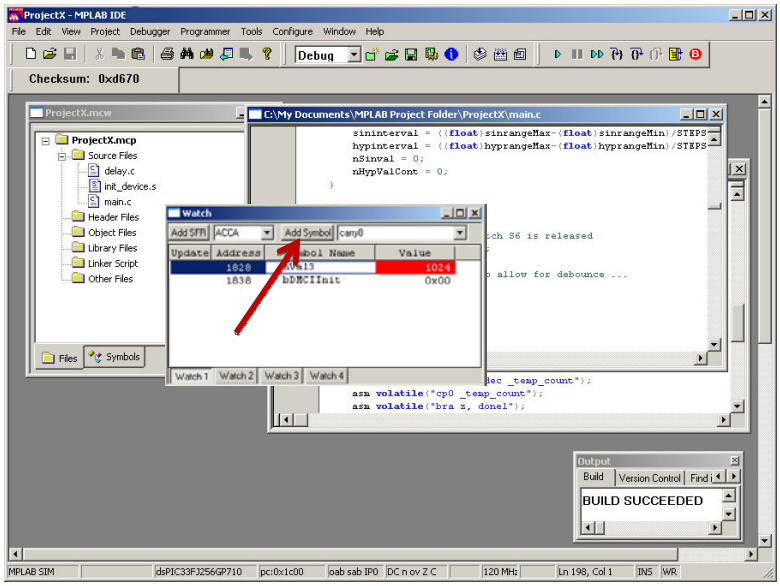
Watch windows inspect variables in your code. Select the Watch window from the View menu.

Watchpoints
Select Variables to Watch



Use the right pull down list to see symbols, select the variables to watch…

Watchpoints
Add Variables to Watch Window



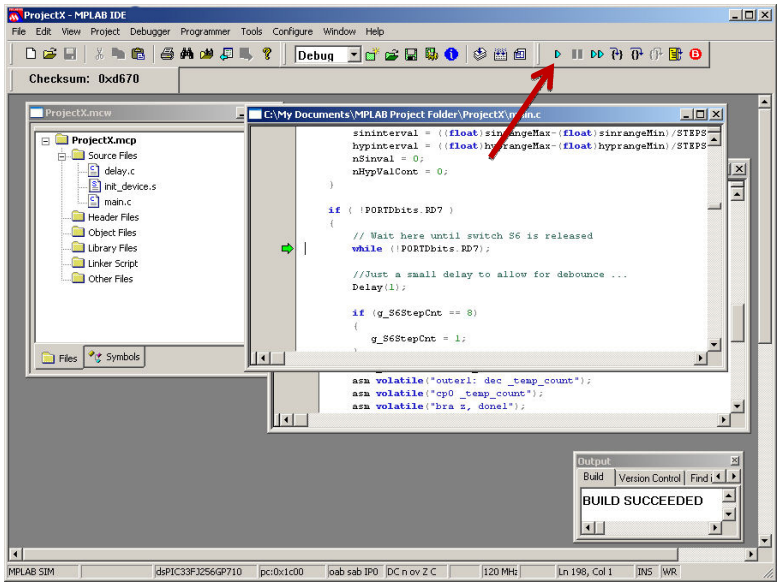
…then press the Add Symbol button to enter them in the watch window list.

Now when the program is halted or stepped, the variables will show changed values in red.

Topics Covered in This Web Seminar

- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

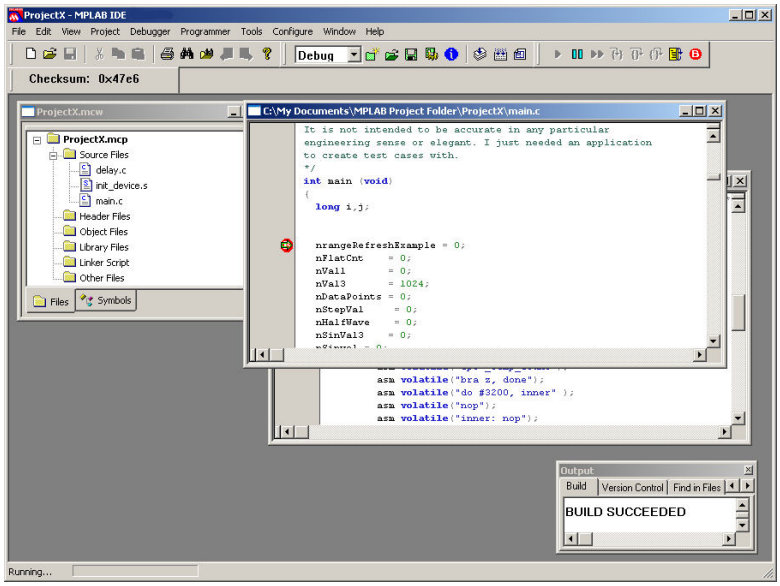
Run



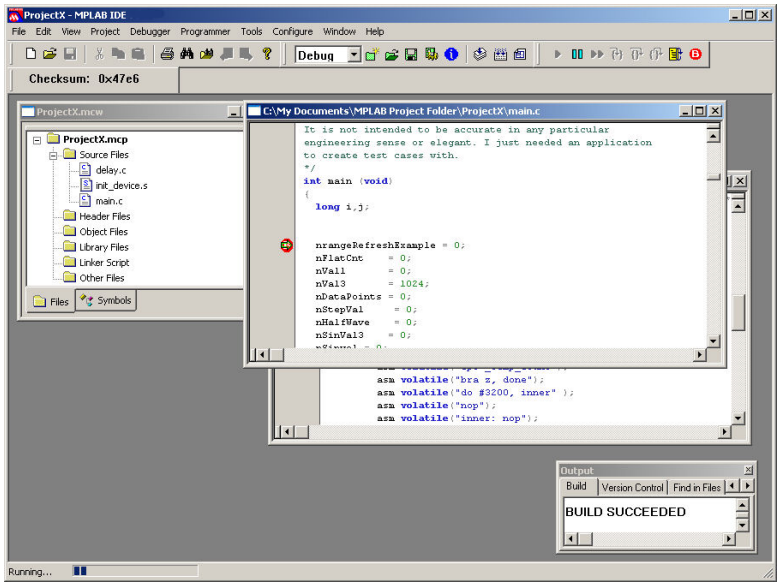
Pressing the “run” icon lets the processor continue running the code from the current breakpoint.

The status bar at the bottom disappears and is replaced by a “running” progress bar.

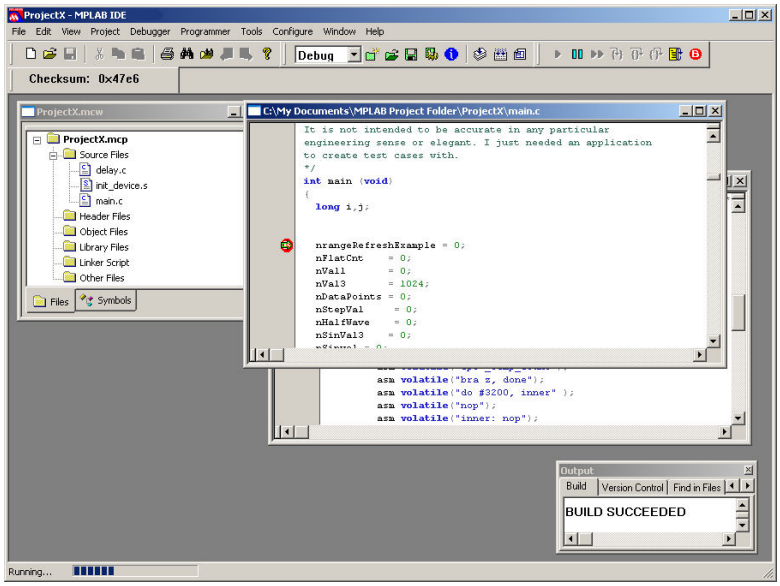
Run


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

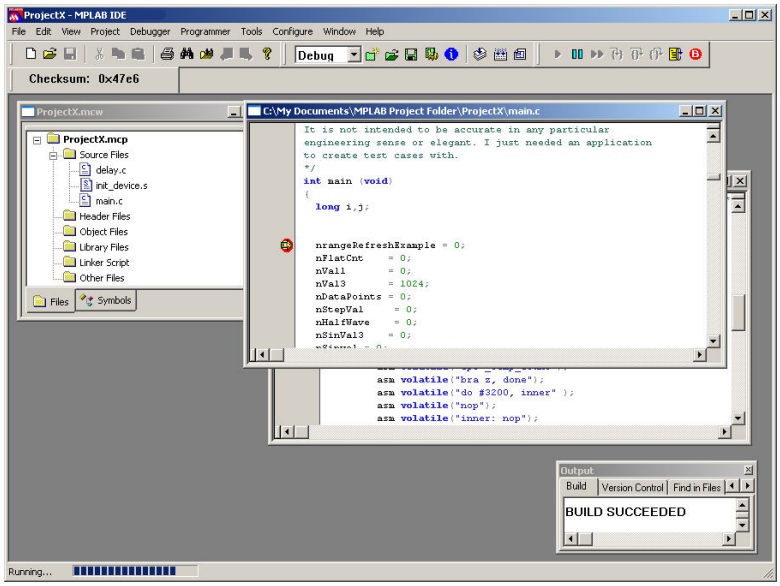
Run


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

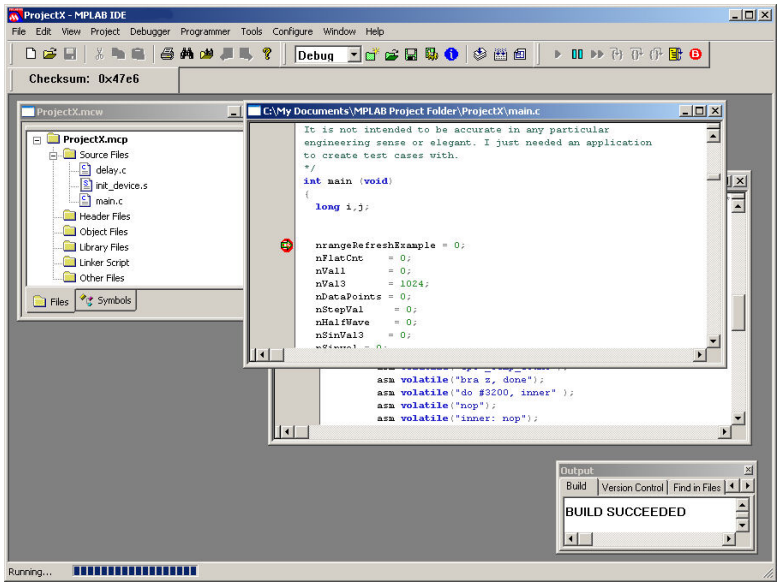
Run


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

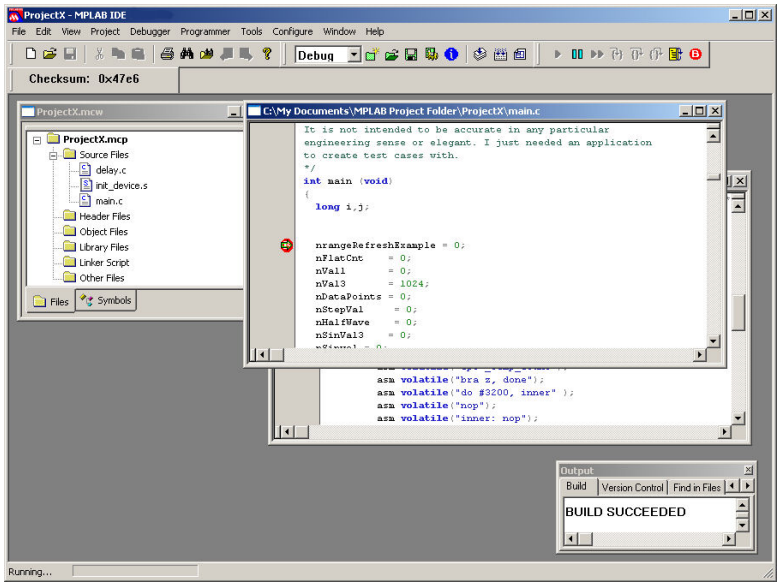
Run


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

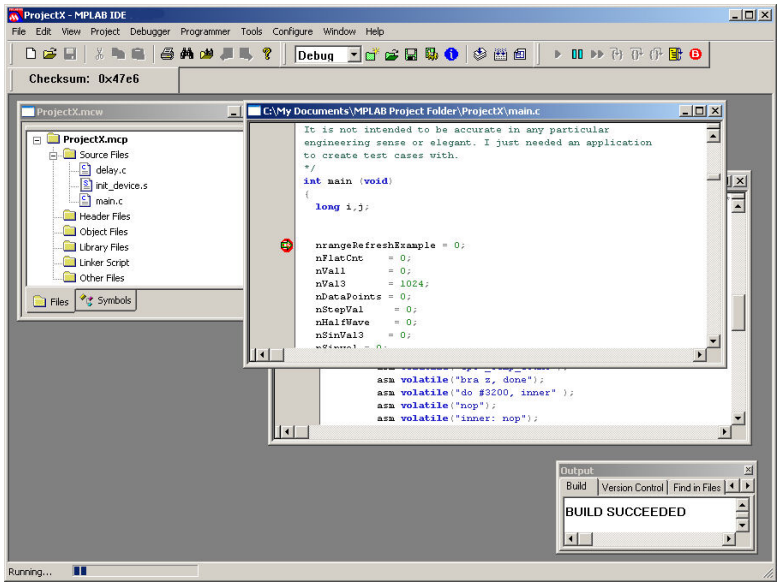
Run


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

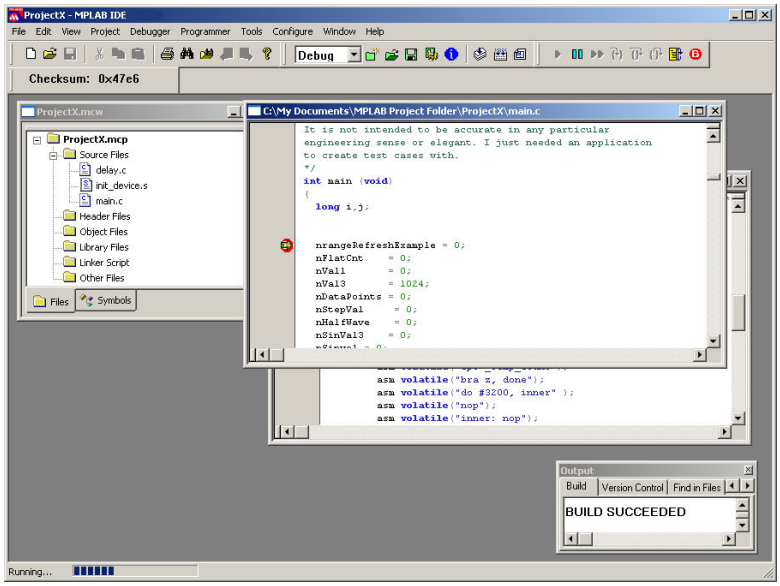
Run


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

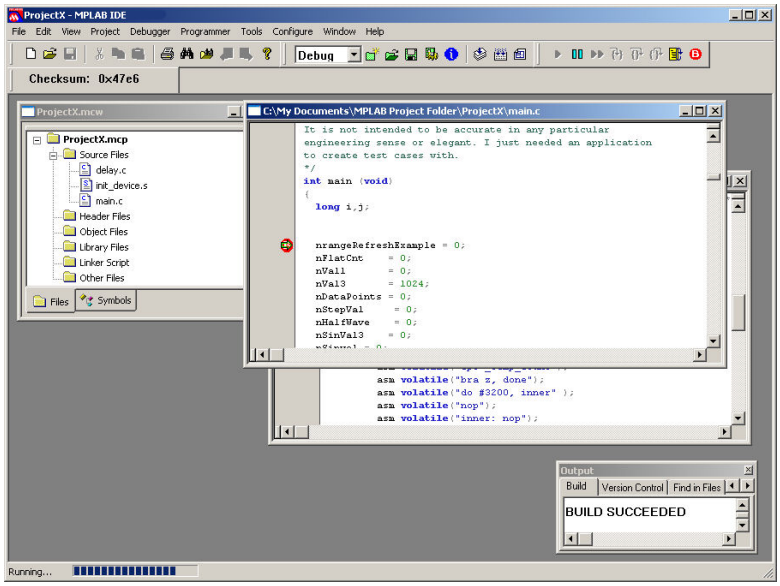
Run


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

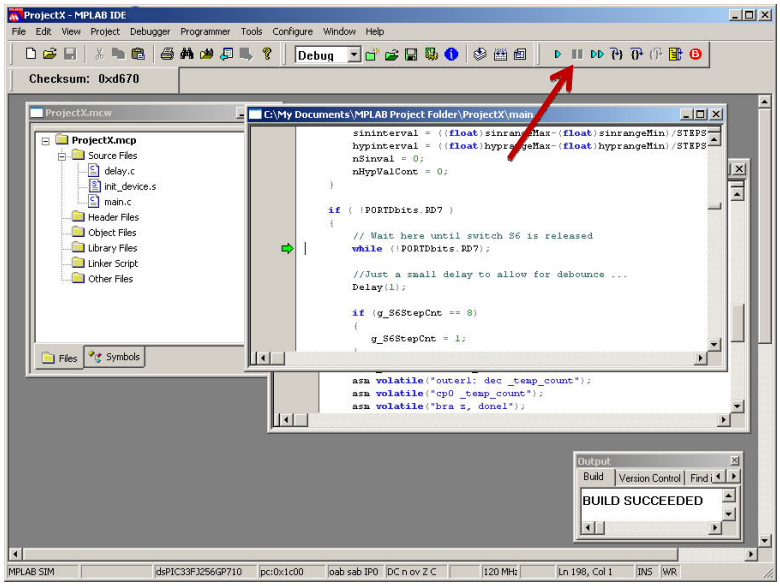
Run


ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

Run

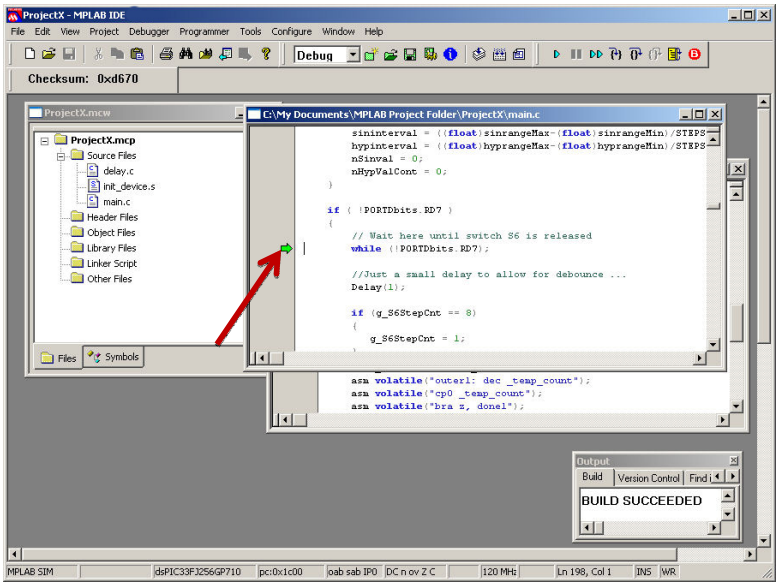

ProjectX - MPLAB IDE
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x47e6
ProjectX.mcw
ProjectX.mcp
Source Files
delay.c
init_device.s
main.c
Header Files
Object Files
Library Files
Linker Script
Other Files
Files
Symbols
C:\My Documents\MPLAB Project Folder\ProjectX\main.c
It is not intended to be accurate in any particular
engineering sense or elegant. I just needed an application
to create test cases with.
*/
int main (void)
{
long i,j;
nrangeRefreshExample = 0;
nFlatCnt = 0;
nVal1 = 0;
nVal3 = 1024;
nDataPoints = 0;
nStepVal = 0;
nHalfWave = 0;
nSinVal3 = 0;
asm volatile("bra z, done");
asm volatile("do #3200, inner");
asm volatile("nop");
asm volatile("inner: nop");
Output
Build
Version Control
Find in Files
BUILD SUCCEEDED
Running...

Halt



Selecting the “Halt” icon stops program execution at the current program counter location, just as if it encountered a breakpoint at that instruction in the code.

Halt



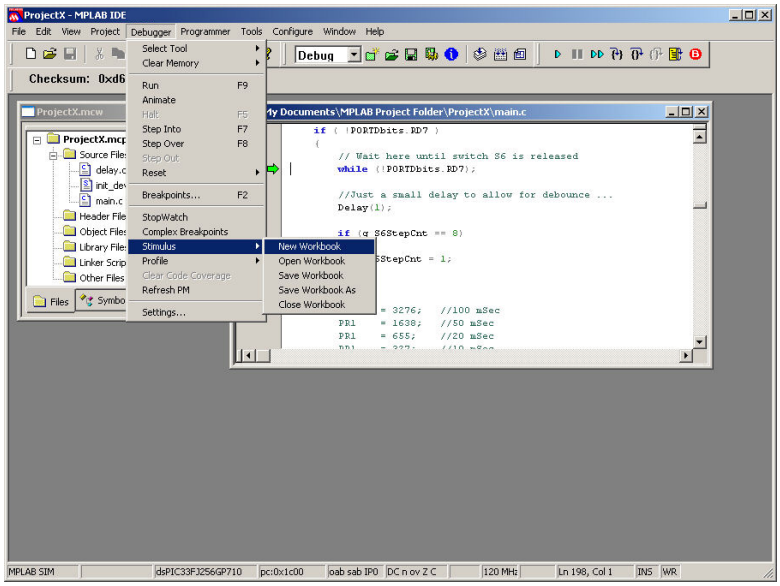
In this code, after pressing single step, the green arrow does not advance.

The code is in a “while” loop, waiting for a switch on port pin RD7 to go high.

Topics Covered in This Web Seminar

- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

Stimulus
Debugger>Stimulus>New Workbook

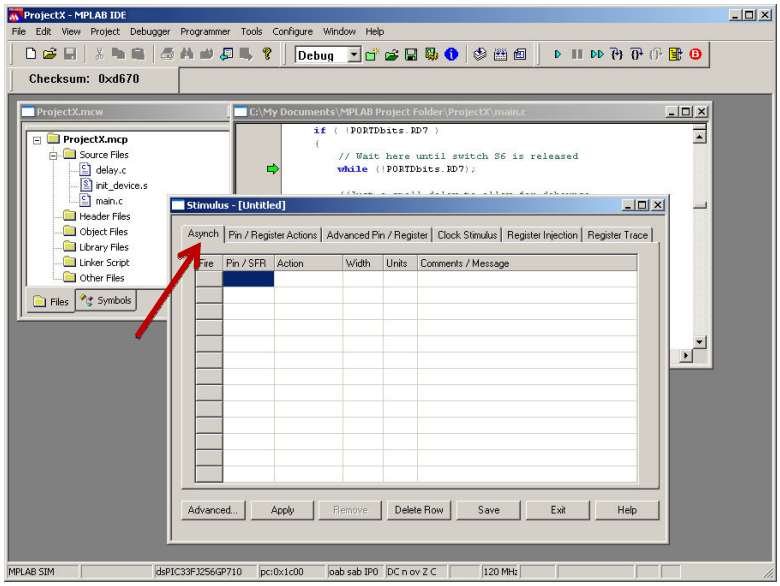


The action of the switch on pin RD7 can be simulated with a stimulus function.

MPLAB SIM can respond to the action of external hardware, such as switches, with the stimulus functions.

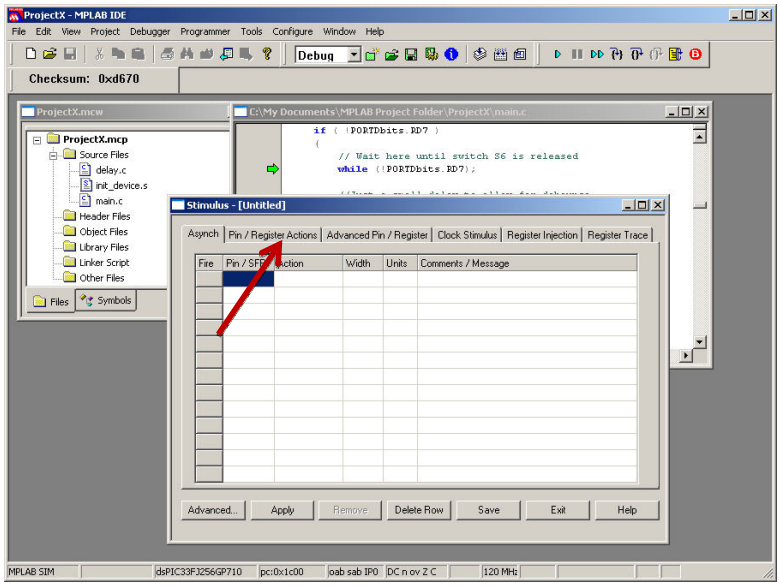
The stimulus dialog is accessed from the Debugger pull down, scroll to “Stimulus” and create a “New workbook.”

Stimulus
Stimulus Dialog



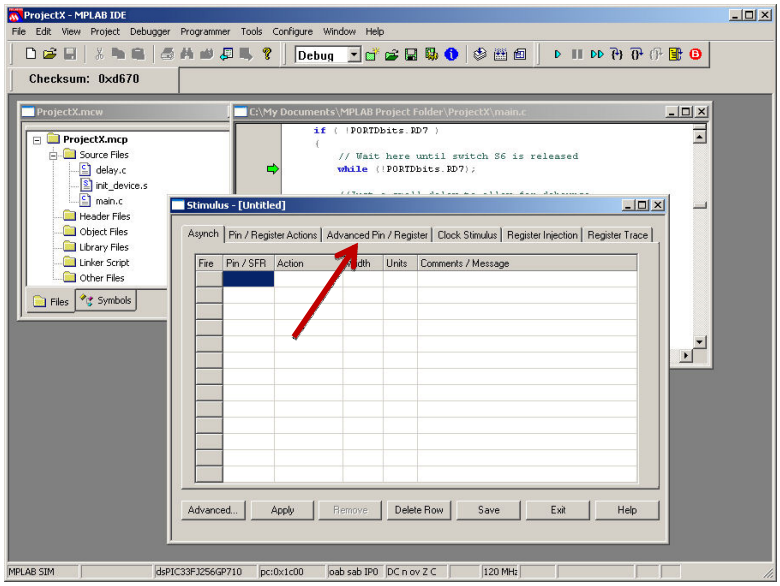
The stimulus workbook has tabs for the various types of stimulus.

Stimulus
Stimulus Dialog



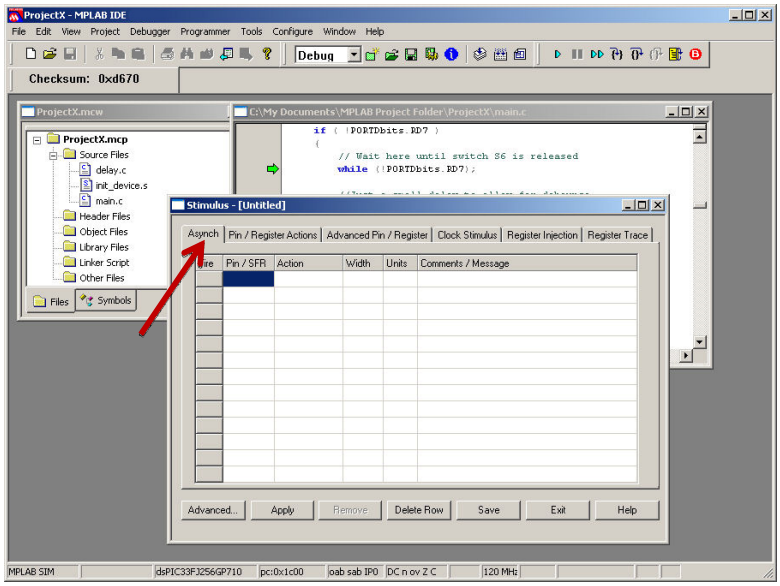
There are six tabs, offering a options to apply stimulus to the simulator,

Stimulus
Stimulus Dialog



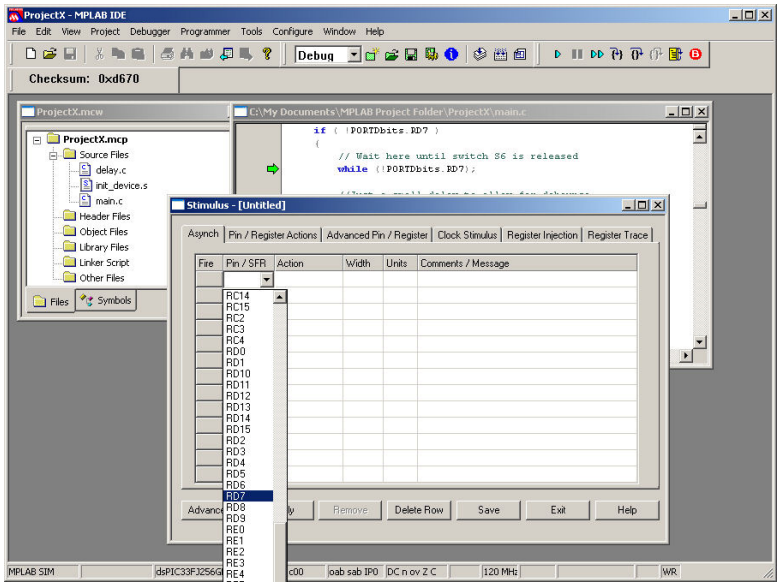
Regular, repeating waveforms, lists of voltage levels, events from files, and sequences of data values can be injected into registers and applied to external pins.

Stimulus
Asynchronous Stimulus



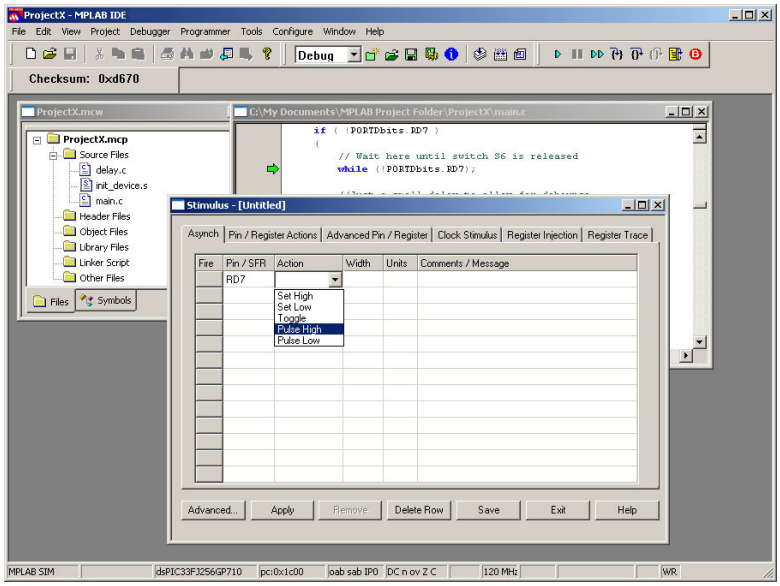
The “asynchronous” stimulus tab can be used to simulate the action of pressing a switch.

Stimulus
Set Pin



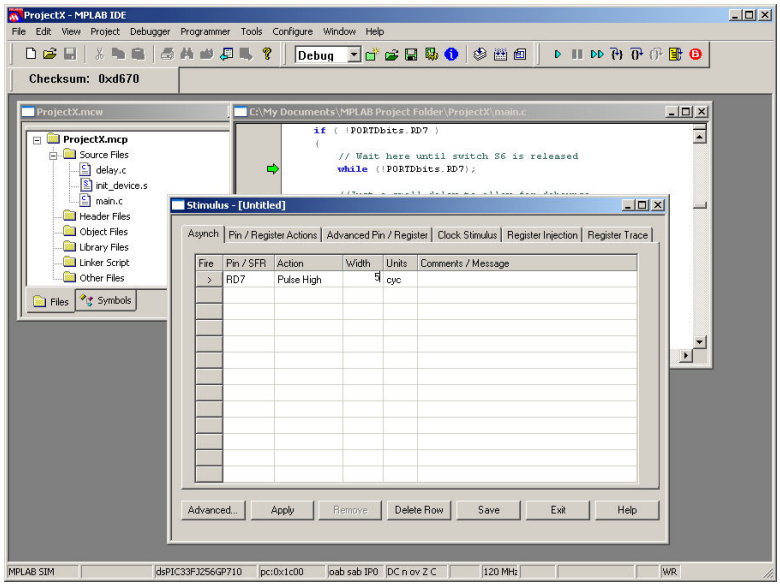
Click on the PIN/SFR column to select the pin RD7.

Stimulus
Set Action



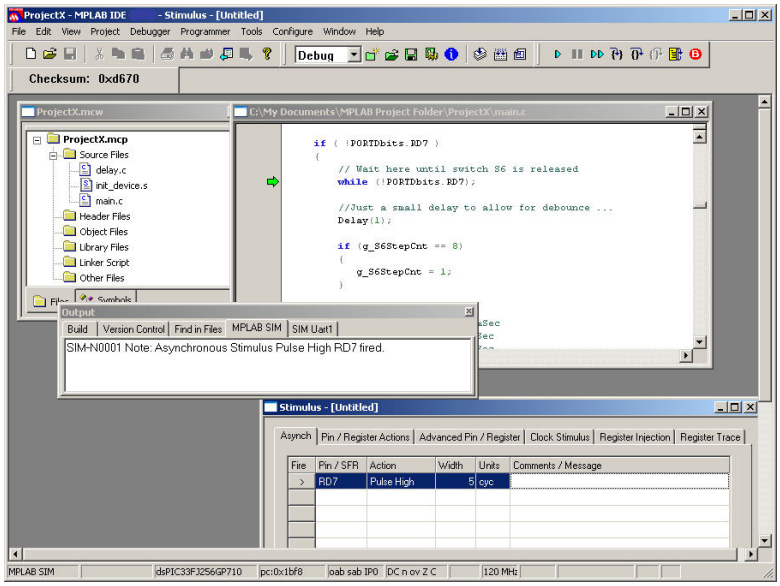
In the “Action” column, set the action for the pin to pulse high when pressed.

Stimulus
Set Time



In the width column, a somewhat arbitrary value of “5” makes the pulse last 5 cycles, just ensuring the pulse lasts beyond a single instruction.

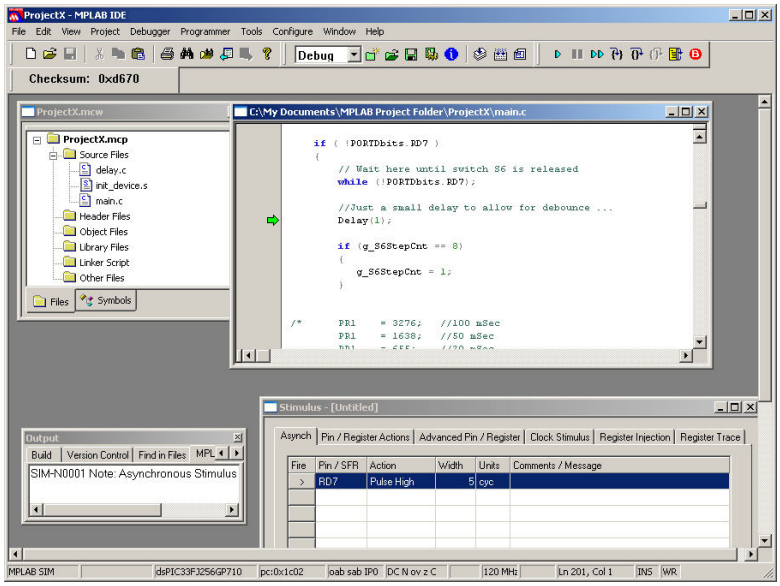
Stimulus
Fire Stimulus



Press the “Fire” button to apply the pulse to the RD7 pin.

The output window logs the action.

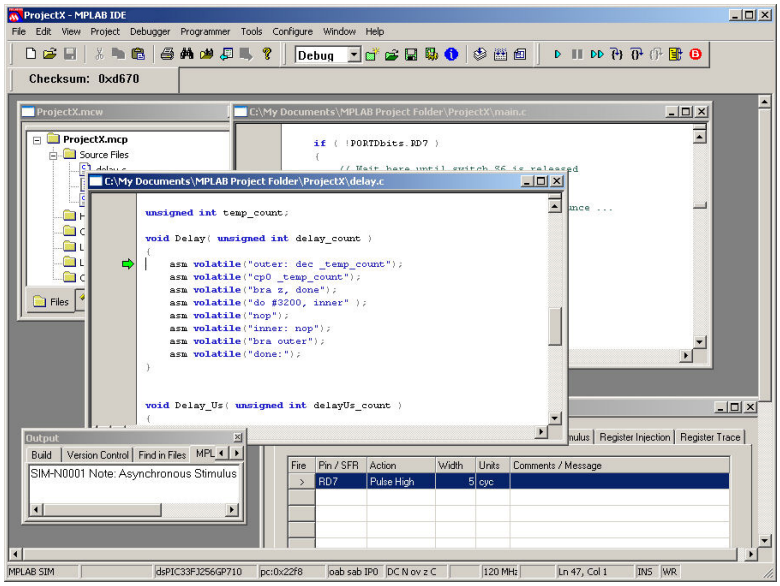
Stimulus
Step



Press the single step key to go forward,
into the Delay routine.

MICROCHIP
WebSeminars

Stimulus
Delay



The next step enters the delay routine.

Delay routines such as this are often used to slow things down so that a display can be seen by the human eye, for instance.

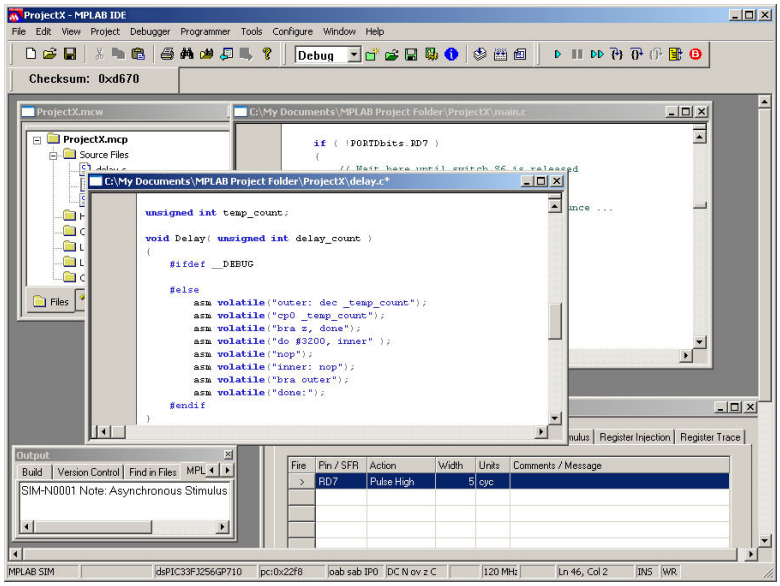
When simulating, which runs at a slower speed than the actual processor, these delays are often not needed.

While using the simulator, we just need to know that the delay routine is called, but we don't want to step through thousands of iterations of the delay loop.

Topics Covered in This Web Seminar

- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

__DEBUG



Use the variable underscore-underscore DEBUG to change the way the delay loop operates while debugging using the simulator.

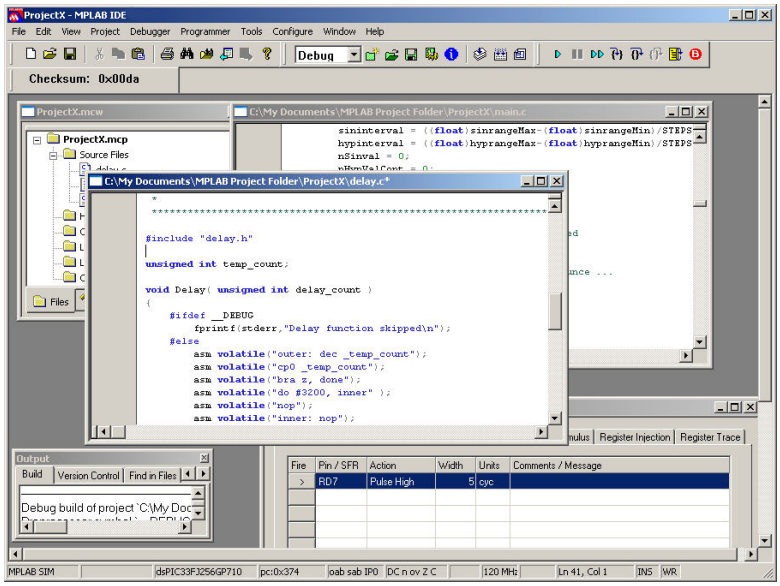
To modify the delay routine to operate differently while debugging, but retain its function in the application add an #ifdef function to check the state of the variable underscore-underscore DEBUG.

If the variable underscore-underscore DEBUG exists, then we'll skip the thousands of loops in the delay routine, and just exit.

Topics Covered in This Web Seminar

- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

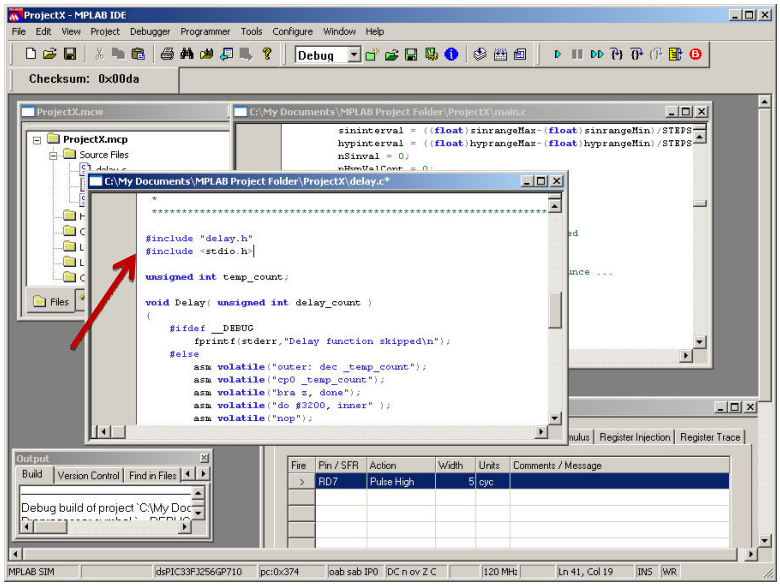
fprintf
Include stdio.h



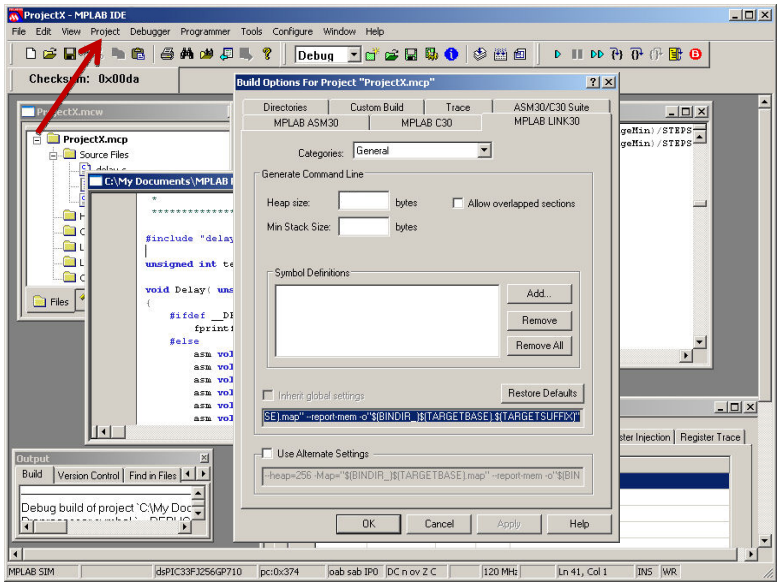
Additionally an fprintf function can print out a message to remind us that the delay routine operates differently while debugging with the simulator.

In order to use the printf routine to echo that the delay routine executed (but in a different way when debugging) the stdio.h file must be included.

fprintf
Include stdio.h



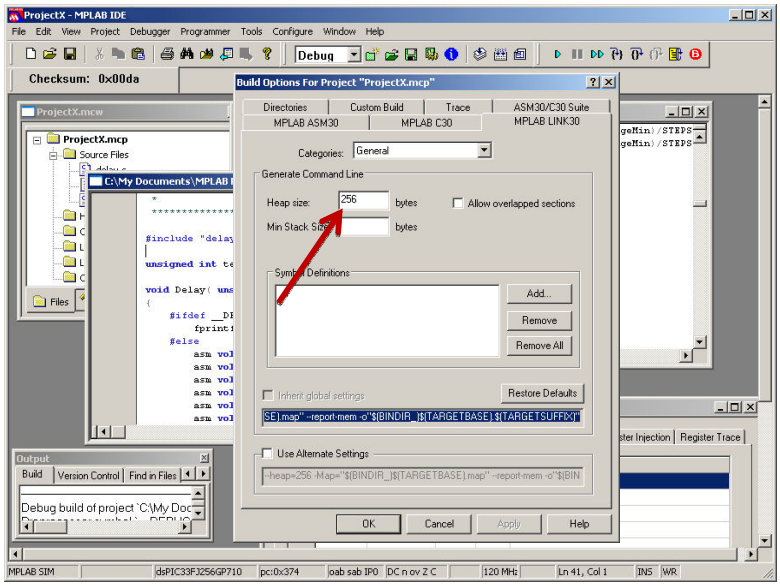
fprintf
Build Options – MPLAB LINK30



A couple of other things need to be set up to use fprintf.

Under the Project menu are the build options.

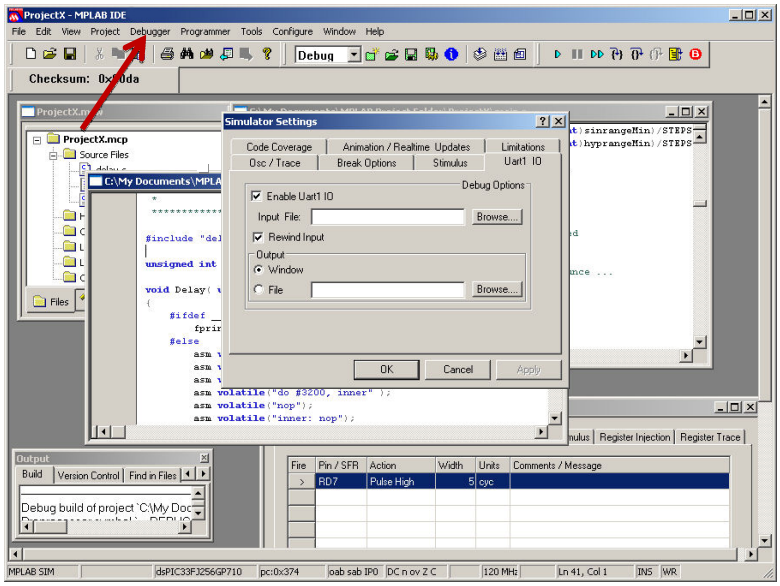
fprintf
Set Heap



The MPLAB LINK30 tab needs to generate a heap to handle the character storage for fprintf.

256 bytes is ample for any message we need to print out.

fprintf
Debugger>Simulator Settings

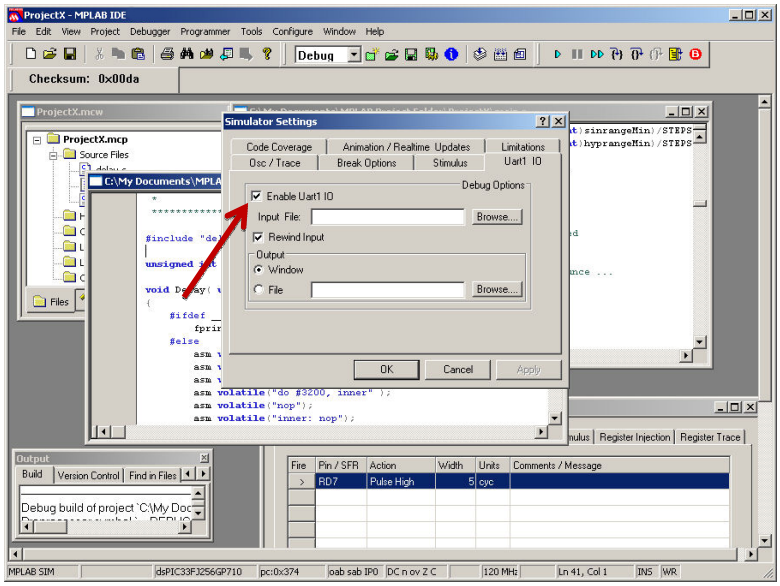


Messages from an embedded controller must come from a peripheral device on that controller.

The UART needs to be configured to send I/O to the Output window.

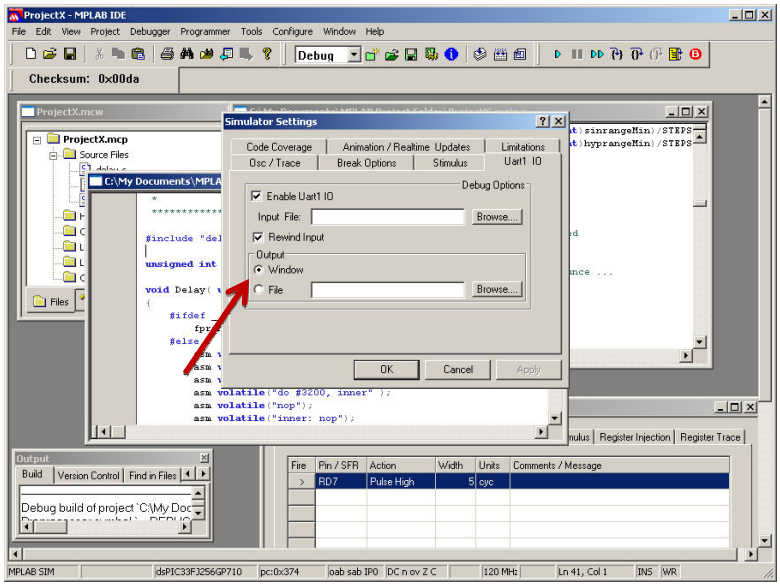
The simulator settings dialog is on the Debugger menu.

fprintf
Enable Uart



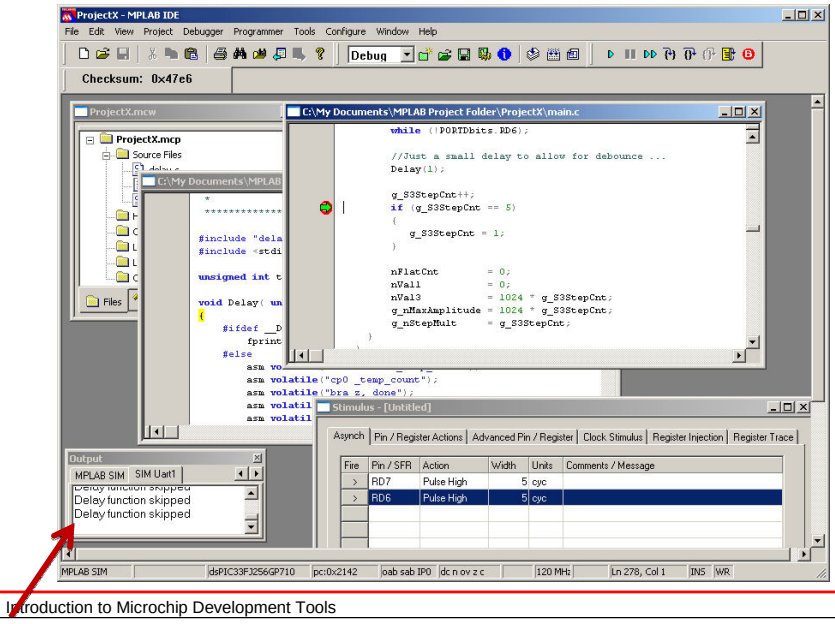
Check the box to enable the UART I/O…

fprintf
Output to Window



…and check this button to see its messages in the Output window.

fprintf
Output Message

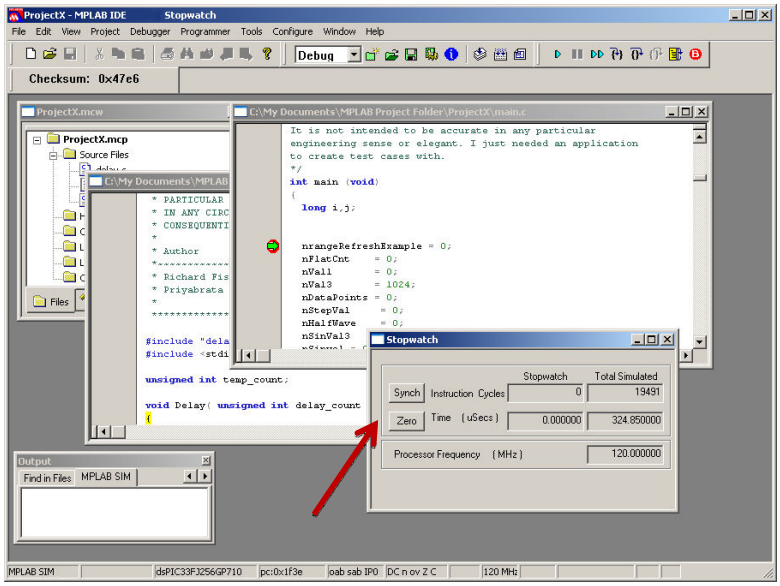


Now as you go through your code, you'll see the fprintf string in the output window each time the delay function is called.

Topics Covered in This Web Seminar

- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

Stopwatch
Debugger>Stopwatch

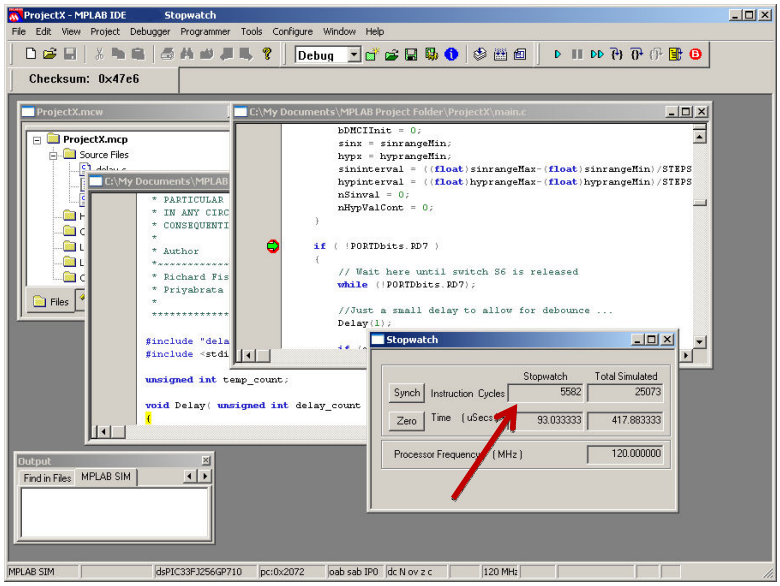


MPLAB SIM has tools to analyze how the code is running.

The stopwatch can time the execution of code.

While stopped at a breakpoint, select the stopwatch from the Debugger menu, then press the “Zero” button to clear its contents to get ready for a measurement.

Stopwatch

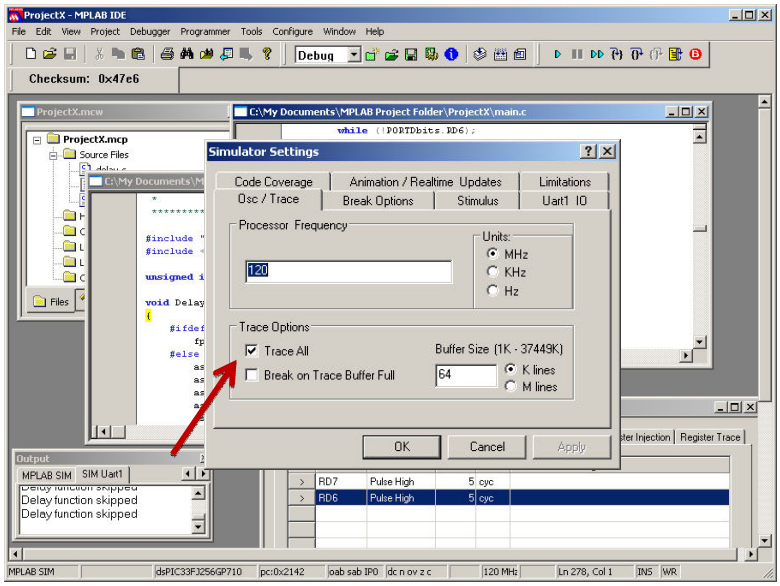


Run to the next breakpoint, and the stopwatch accurately measures the instruction time in cycles and microseconds.

Topics Covered in This Web Seminar

- Starting MPLAB SIM
- Running to a breakpoint
- Single stepping
- Setting watchpoints
- Running and halting
- Using a stimulus
- Using the __DEBUG variable
- Using fprintf statements to debug
- Measuring routine execution time with the stopwatch
- Tracing code as it executes

Trace
Debugger>Settings

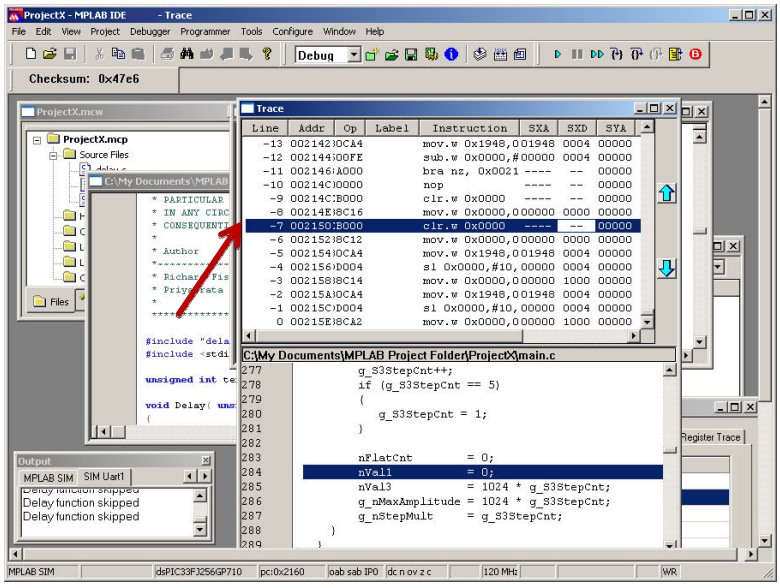


Another way to optimize code is to use the trace analyzer.

While setting breakpoints and single-stepping through code is one way to see how your code is functioning, an alternative is to use the trace facilities of MPLAB SIM to record instructions as they execute while the simulator is running.

Use the Debugger menu to select the Settings dialog, and enable the “Trace All” checkbox.

Trace

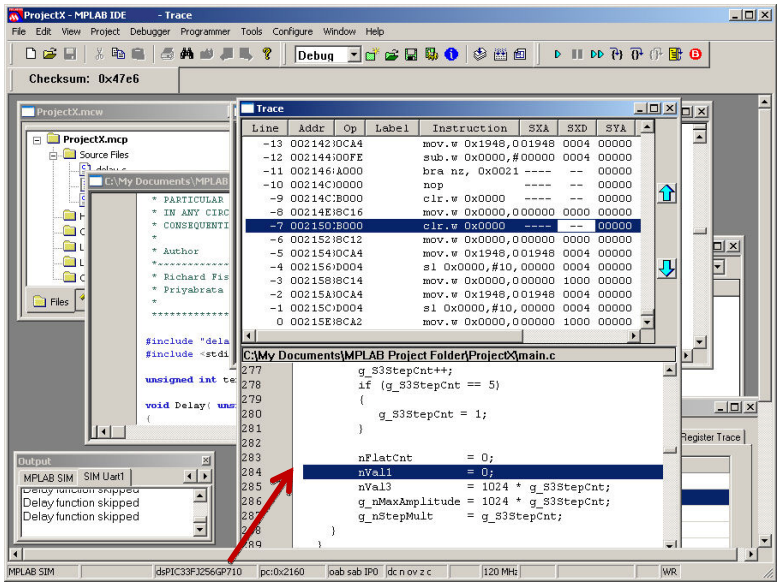


Press run, then halt, or stop at a breakpoint, and view the trace window from the View menu.

The upper half of the trace window shows the instruction flow.

When you click on an instruction there…

Trace



…the corresponding section from the source code is shown in the lower half.

Trace is useful to see how you got to a certain point in your code.

Download MPLAB and Try It!

WWW.MICROCHIP.COM/MPLAB

That's a quick tour of the simulator. We hope this will give you a few pointers and you'll explore these and other simulator features.

If you haven't already done it, now is the time to get started with MPLAB IDE.

Go to our web site at www.microchip.com/mplab and download your free copy of MPLAB software.

This is the end of our presentation.

Thank you for your time.