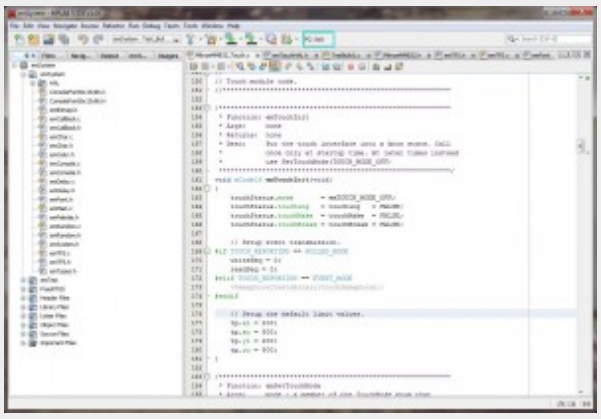
MPLAB-X Impressions

Posted on **February 16, 2012**

This post is a change of pace from the previous ones. We're going to step back and take a look at the development environment that I use in my current work. I speak of the Microchip MPLAB-X (V1.00) IDE. I will cover the compiler I currently use, C32 (V2.01) at another time. MPLAB-X is Microchip's new flagship IDE for PIC development tasks of all sorts. I first learned of MPLAB-X about a year ago while it was still in early beta. I confess that Betaphobia (not to be confused with Cetaphobia) ruled and I stayed away from it. However, all was not lost; I got to use MPLAB-X at last years Master's Conference where I have to admit I was most impressed. After the conference, I switched to it, in spite of the fact that it was still in beta.

So some basics first: It's free! Just download V1.00 and install. Yes, it's a dreaded V1.00, but more on that later. I have long since taken advantage of the extensive customization capability, so while my MPLAB-X looks like this, your's won't unless you share my need to see as many lines of source code at once as possible:



— The MPLAB-X IDE as configured by me.

MPLAB-X is a full featured IDE designed to integrate with a wide variety of compilers and development tools for the full line of Microchip products. There is so much much ground to cover that this is not so much a review as my impressions. One really good bit of news is that MPLAB-X

is based on the open Net Beans IDE. I look forward to being able to develop and make available my own special pluggins. As with most IDEs it comes with project management, source code editing, project building, and debugging capabilities. Nice extras are even though it supports popular source code control systems, it also supports revision tracking on the local machine. Right out of the “box” when started, MPLAB-X displays a most helpful start-up page:

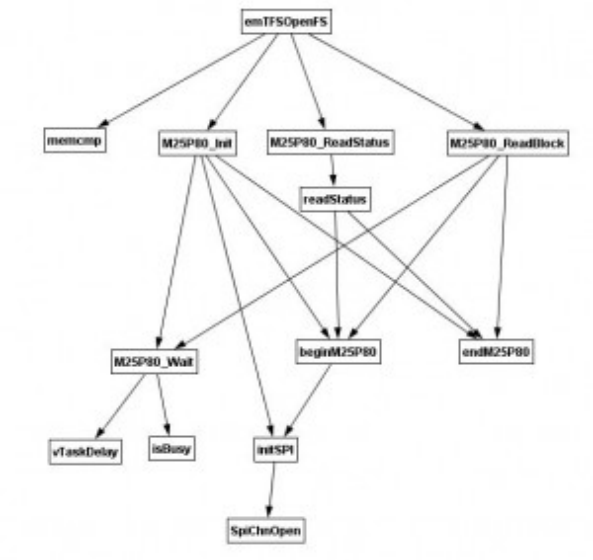
— The Start Up page. Not just a nuisance!

In the Learn and Discover Tab you find Getting Started material, Tutorials, a number of helpful Wizards, and links to community and support. While there is a tendency to skip over this page, You shouldn’t as it contains valuable information. I like to at at least glance at it before moving on to my code.

Once into the development environment it is pleasant and easy to use. The editor includes many of the conveniences of modern editors such as color syntax, symbol and error highlighting, auto look up and completion of symbol names, code re-factoring, code navigation, and text formatting. The editing environment is a joy to use and I won’t go back to the old MPLAB, however I did run into one glitch here though. I noticed that sometimes when I entered a symbol, the editor would add a redundant include directive to the code. Apparently, the editor had lost track of the fact that the file was already included elsewhere.

To fix this, I unchecked the “Auto Insert #include Directives for Completed Identifiers: option under Tools -> Options -> Editor Tab -> Language Selection -> C++. See the illustration for the view at the end of this rather laborious menu navigation challenge. Once unchecked, the phantom include directives ended, but I was again responsible for adding them myself, which suits me just fine.

.



— A sample call tree from my Trivial File System module.

Also present in MPLAB-X are some superb features to enhance code understanding.The call tree graph is especially useful in studying code that you did not write. What does this function call and where do the calls go is usually a very tiresome exercise. With the call tree, it's simple to get great looking graphs like the sample to the left. These graphs also make great documentation for internal system specifications or for impressing peers, managers and pets. Well maybe not pets, they're much more difficult to impress.

A major area where MPLAB-X shines is that of build management. The old MPLAB had only one build configuration per project.

Under MPLAB-X a single project can command multiple build configurations with different source files selected, preprocessor symbols defined and different compile and link options. All of this being very customizable in this easy to use dialog shown to the left:

In addition, if you select a file in the project tab, and right click for the menu and select properties, you are given a dialog that allows you exclude that file from the build or use build options different from the other files.

This is shown to the left. There is one fly in the ointment here. If you have set a great many paths and other options up, you will have to redo all of that effort for the exception file. It would be nice and save a lot of time, if the exception file started off with the same options it would have had as a regular file.

The final major area I want to touch upon is the debugger. MPLAB-X comes with an integrated debugger with all the usual features we've come to expect: The call stack, watch windows, breakpoints, memory and register windows, plus single stepping of all kinds and flavors. Now my experiences are based on my use of MPLAB-X with my ICD-3, and I must say that this is the weakest link of the MPLAB-X package.

For starters, the ICD-3 drivers seems to get lost every once in a while. Once lost, the only answer is to use the little MPLAB-X driver switcher utility to bring the ICD-3 back as an MPLAB-X device. This also happens if you are debugging and walk away in debug mode for a long while. On return the debugger will not exit. Again to recover, exit MPLAB-X and run the driver switcher to correct this. I suspect that Windows 7 maybe interfering and reverting to old drivers because it feels like it, but I can't say one way other the other. Anyway you look at it, this does not auger well for long debug testing.

Earlier betas of MPLAB-X had severe problems with watch windows being unable to locate local variables, but those issues seem to have been sorted out in V1.00. A constant problem with advanced processors such as the PIC32 is the need to remember to turn off the optimizer for

debugging. The optimizer completely scrambles the relationship between source code lines and machine language instructions to the point that single stepping seems to jump around the code randomly. This can be a problem if code space is tight, because unoptimized code is much bulkier. The answer is to compile most of the files optimized with only the file under test unoptimized.

I realize that my look at MPLAB-X has been far too brief to give any idea of the real flavor of the tool in use. I can say in summary that I am very glad I decided to make the leap of faith and give it a try. Even the little problems I encountered were opportunities to learn and improve my skills and for a V1.00 it's pretty nice today. So my advice is to plunk down the $0.00 (after getting budget approval of course) and give MPLAB-X a try. I'm pretty sure you'll like it!

Peter Camilleri (aka Squidly Jones)

This entry was posted in **Dev Tools**, **Embedded Development**, **MPLAB-X** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=329]** .