Developing the Initial Software for the Open Source Wig-Wag Project

This document provides an overview of the tools and process I used to develop the initial software for the Open Source Wig-Wag Controller Project. When selecting tools, I focused on development tools that would be easily available and low cost (free if possible). The use of the PIC12F683 microcontroller provides a lot of options for development tools but I found that, for me, going with the standard toolset from Microchip provided the capabilities I needed at a reasonable cost. In fact if you only want to write C or assembly code and test it with a simulator there is no cost at all. Only when you need to flash the code to the micro do you need to make a small investment. Beyond that point you can spend more money to get more functionality in the development tools. And I should mention that this is only one of many approaches that are available. I am sure that there are other options that would be just a good or better than the path I chose and hopefully others will provide info on these other options. I should also point out that I come to this point with a lot of prior experience in embedded software development, a novice might have a significant learning curve at several points in the process. Anyway, here is the process I used to get going…

1. Download and Install the Development Tools

I downloaded the following tools from the Microchip website:

MPLAB IDE Ver. 8.84
HI-TECH C for PIC10/12/16 MCU (Lite Mode) Ver. 9.83

Note that there is a newer IDE (Integrated Development Environment) that is available from Microchip (MPLAB X IDE) but I decided to go with MPLAB IDE since it provided all of the capabilities I needed for the PIC12F683 controller and it seemed to be a stable mature product. I expect that newer MPLAB X IDE would also be a reasonable choice and I will probably make a switch at some point. Note that MPLAB IDE is Windows only and MPLAB X IDE also supports development on Mac and Unix.

The Lite Mode version of the HI-TECH C compiler is free but it does not provide the code optimization capabilities of the other versions. For simple applications such as the wig-wag controller, the lack of code optimization should not be a significant issue. C code in these types of applications tends to have simple logic and data structures that translate cleanly into efficient assembly code.

There were no installation issues on the three machines that I set up with the IDE and compiler (one each of Windows XP, Vista, and 7). I have the project files loaded onto a directory on a network drive and there have been no problems accessing and updating the project files on the network directory from any of the three PC's.

2. Download and Study Reference Materials

This is the first time I have used a Microchip microcontroller so there was the initial learning curve for a new IDE and microcontroller but I would say that the documentation for the tools and micro are pretty good. Everything seems to work together right out of the box without any major issues. I would recommend the following reference materials from the Microchip website:

PIC12F683 Data Sheet	Not sure why they still call these things data sheets, it is a 176 page document. Covers everything you need to know about the functions provided by the chip. Seems to provide the information in an easy to reference fashion, I did not have any trouble finding what I needed. There is a lot of functionality on the chip that is not being used for the wig-wag, so the only challenge was making sure everything not being used is disabled.
HI-TECH C User Guide	Covers all of the info needed to write C code specific to the PIC micro. Make sure you also download the Readme file with the release notes for the current version. HI-TECH C seems to be a pretty standard cross-compiler for use with a microcontroller.
PIC12F683.h	This file is found in the "include" directory in the HI-TECH C installation in the program files under Windows. Contains all of the macro definitions for the on-chip registers and memory mapped I/O. The macro identifiers map pretty well to the identifiers used in the data sheet document.

	NOTE: This file contains the current standard macro definitions, if you are compiling older code, such as that provided in some of the Microchip Tutorial lessons, you need to use “Legacy Headers”, see the release notes for details. Easy to detect the problem when it occurs, the compiler will give error messages on undefined macro identifiers.
MPLAB IDE User Guide	Contains introduction to the IDE and tutorial examples of how to set up a project, add files, edit files, build, simulate, program, and do in-circuit debug. IDE is pretty user friendly but there are some areas where I found I needed to take a look at the User Guide when I got stuck.
Mid-Range MCU Family Reference Manual	Good overview of using the various chip functions.

3. Developing the Initial Wig-Wag Code

I had purchased the PICkit 1 Flash Starter Kit (see next section for more info) and used the tutorial lessons from this kit to get started. The first tutorial covered reading the pushbutton switch on the kit board, de-bouncing the input, and activating the LED’s on the board (the embedded software version of the “Hello World” program and all of the essential functions needed for the wig-wag controller). The kit comes with a PIC12F675 chip for use on the tutorials so I proceeded ahead knowing that I would transition to the PIC12F683 after I had verified everything was working with the PIC12F675. I downloaded the provided hex file to the chip and everything seemed to be working fine. I next created a new hex file from the assembly code example and it downloaded and worked as expected. I next did a build from the C code example provided and found the issue with undefined macro identifiers noted above. A quick Google search provided info on “Legacy Headers” (if I had read the compiler release notes I would have known about the issue). Added the #define to pull in the Legacy Headers and everything worked as expected when I downloaded the new build into the chip. At this point I had verified that I could create a new C project in the IDE, build the code for the PIC12F675, flash the chip with the code, and run it on the PICkit board.

After looking at the two de-bounce code examples in the tutorial I decided that neither was quite right for the wig-wag (one was too simple and the other was too complex). I would need to port the code to the PIC12F683 in any case and add a second switch input so I decided to start from scratch rather than modify something from the tutorial code. The tutorial code was useful in that I had a good idea of what would be needed to set up the micro resources for the wig-wag. I chose to use a simple de-bounce function that looks for a series of constant readings to filter the raw inputs into a de-bounced input. I spent some time studying the PIC12F683 in comparison to the PIC12F675 and looking at the HI-TECH C manual for guidance on PIC specific C features. I coded up some initial C code for the PIC12F683 to read the single switch on the board, de-bounce it, and light an LED. I was surprised that it worked as expected the first time (it has been awhile since I programmed anything in C…). I added the code for the second switch and LED, reflashed the chip, and moved it over to a breadboard setup with two switches and two LED’s. I had managed to not screw anything up and it worked as needed for the wig-wag. I spent some time reading through the PIC12F683 datasheet again to see if there were any items I had missed and added some additional chip initialization items that I had missed in the first pass at porting from the PIC12F675 to the PIC12F683. I also added additional comments into the source code to try to make it easy for anyone to proceed with modifications as needed for new applications.

I spent some time testing the functionality on the breadboard and did not see any anomalies with as many crazy switch presses as I could come up with. Not what you would call a structured approach to testing but testing nonetheless. At this point I decided to switch to using an in-circuit debugger (see next section) and spent some time getting that set up correctly. Required some changes in the code for chip configuration in debug mode. Note that one of the build options in the IDE is to build for debug or build for release. A macro identifier is available to allow selection of source code for debug and release. I did some initial testing with the in-circuit debugger and found no issues. This is the code I sent out for review. Total effort so far is probably about 15-20 hours over about 10 days.

Next step is to create a breadboard setup to test the de-bounce logic which seems to be working OK but is not really tested to see if it works at the detailed level. I plan to use another micro (probably a PICAXE) to create the raw switch inputs into the wig-wag micro so that I can create bouncing inputs of various periods to verify that the de-bounce logic is working as it should. I then will do testing using one of the initial samples of the wig-wag controller board to verify that things work on the actual hardware. At that point I will probably declare success and get back to building the motorglider that will be using the wig-wag controller.

4. Flash Programming and In-Circuit Debug

The lowest cost option from Microchip is to purchase the PICkit 1 Flash Starter Kit for $36. This USB based board provides the capability to flash an array of 8 and 14 pin chips (see the Microchip website for details). The board also provides some I/O devices (LED's, switch, pot) to support a set of Tutorial Lessons and a small prototyping area. You need to download the Windows application (PICkit 1 Classic Flash Ver. 1.74) to interface with the board. The Windows application allows you to do the basic programming functions (download, verify, read). Note that there is no capability to do in-circuit debug with this board. Tutorial lessons are performed by building the example code in the IDE, downloading to the micro with the PICkit 1 Flash Starter Kit, and observing that the micro performs the functions of the lesson. The micro simulation function in the IDE is used to study the line-by-line execution of the code. For $36, this seems to be a reasonable option for getting started with the PIC micro and it provides everything needed to do the code for the wig-wag. I purchased this and had it available as I started developing the code for the wig-wag controller.

Having spent a lot of time doing embedded software development as a profession I find it frustrating to not have an in-circuit debugger available. I am willing to spend a little more money to get in-circuit debugging. Here are two scenarios where in-circuit debugging can save lots of time and effort. You've verified the operation of your code using the simulator in the IDE and it seems to do what you want. You program the micro and install it in your circuit and the code doesn't function as expected. What do you do next? With no direct visibility into the micro you have a challenge on your hands. Or in the case of the wig-wag controller, I had finished the initial code and the micro functioned as expected but I wanted to be able to verify that the switch de-bounce logic works as it should with real raw inputs into the chip. For me, an in-circuit debugger is worth the money, so I purchased the PICkit 3 In-Circuit Debugger.

The PICkit 3 In-Circuit Debugger is the low-end Microchip debugger at a base price of $45. For the PIC12F683 you also need to purchase two additional items, an AC164110 adapter for $10 and the AC162058 Debug Header for $25, bringing the total price to $80. Debug headers are placed in the circuit where the chip normally resides and are only needed for low pin count chips that do not have built in debug capability. The debug header contains a special version of the chip with additional pins to support the debug function. Make sure you are careful with the header installed since burning out the chip requires buying a whole new header. Once the code is debugged you flash a regular chip and replace the header with the "production" chip. There is no capability to do debug with the actual PIC12F683 chip. Note that in-circuit debugging on the low-end PIC chips using the Debug Header is limited (e.g., setting a single breakpoint in the code, single stepping the code, and reading memory locations when halted). The PICkit 3 In-Circuit Debugger also has programming capability and could be used to flash a PIC12F683 chip but you would need to wire up a connection between the programming pins on the debugger and the chip to be programmed.

Final Thoughts

Hopefully this summary of my experience will provide some useful information to anyone thinking of getting involved in software development for the Open Source Wig-Wag project or follow-on projects. Again, I want to emphasize that this is simply the path I happened to take to get started. It seems to have worked fine for me but I am sure that there are other options for development tools that would probably work just as well if not better…

For anyone who sees this process as too big of a jump into the pool but they still want to get their feet wet there are several options that ease the learning curve. One example is the PICAXE line of processors. Through the use of an easy to use IDE, the use of BASIC (or Flowcharting) as the language for developing the functional logic, the use of a hardware abstraction layer that hides all of the gory details of the chip from the programmer, and the use of a simple USB/Serial interface between the development PC and the microcontroller for both programming and communication, anyone can get started right away developing embedded applications. The major limitations are speed of execution and program size but by the time you find those to be a problem you should be ready to move on to the next step. Another popular starting point would be the Arduino line of open source hardware boards and the associated development toolset, much more processing power than the PICAXE (or the PIC12F683), with lots of community support and not too hard to learn. Arduino was designed for non-technical folks to develop interactive objects and environments so it has a pretty short learning curve.

Feel free to contact me via the AeroElectric List or directly at gregmchugh@aol.com if you have any questions or if you need some help getting started with embedded software development.