Laboratory Exercise 1: Controlling a Toaster Oven

This lab was inspired by a joke from a few years ago, the punch line of which included the execution of a Computer Scientist. Out of consideration for the Computer Science majors in the class, I will refrain from telling the joke. Nonetheless, the story had a serious point, which was that engineering problems have complexity, cost, and design-time constraints that limit the range of acceptable solutions. Here is an example of such a problem. (I should admit that the solution we have you design is not as clever and cheap as Black and Decker's, which did not use a processor at all.) We have taken a couple of Black and Decker™ toaster ovens and modified them so that you can safely add a microprocessor to control the oven. (To minimize fire hazards while giving you more apparatus to work with, we have also built two toaster simulators that will cook simulated toast with the same interface as the real toaster oven.) The processor must sense when one starts the toaster oven and whether it is TOAST or OVEN mode; the microcontroller then sets the heat time if it is toast or sets the amount of heat if it is in oven mode.

Requirements: You will be given 6 pieces of Microchip Technologies' PIC16C505-04 micro controllers. This is a small RISC processor with an 8-bit data bus and 12-bit instructions. **It is one-time-programmable – meaning you can only program it once and, if it doesn't work, it is useless. You only get the 6 parts so be careful!! If you use up the six and still don't have toast, then you're toast!** This will mean you have to be very careful about making sure the program is satisfactory before trying it out. You will get three micro controllers with a Lab 1 kit that includes the other parts you need as well. (See resources section.) When you have used the three, you can get three more and that's all. You return the kit to the lab TA when you demonstrate your program and submit it for recording. The software to program these devices is on the server for the Instructional Computing Facility. In addition to a cross-assembler for this device, there is a simulator. Test the program thoroughly with the simulator before you blow out a processor. Pay particular attention to the hints and instructions on using the software that follow this specification.

Figure 1 shows a block diagram of the whole toaster oven system. You build the block labeled "Your Lab 1 System." The oven has four controls, two knobs and two pushbuttons. One knob, originally the oven temperature knob, has a detent at its CCW end, and a pair of switch contacts is actuated as the knob moves in or out of that position. You will use those contacts to determine whether the machine is to be a toaster or an oven.

Figure 2 shows the detailed schematic diagram of the toaster controls as we have modified them. The signals to your breadboard are shown on the right side. Table 1 gives the signal functions and pin numbers of the lines on the cable that goes from the toaster oven to your breadboard. Pin numbering is marked on the connector and follows the standard used for integrated circuits. The oven knob switch and its contacts are wired to a pull-up resistor to provide you with a signal called TOAST. When this signal is HIGH, the

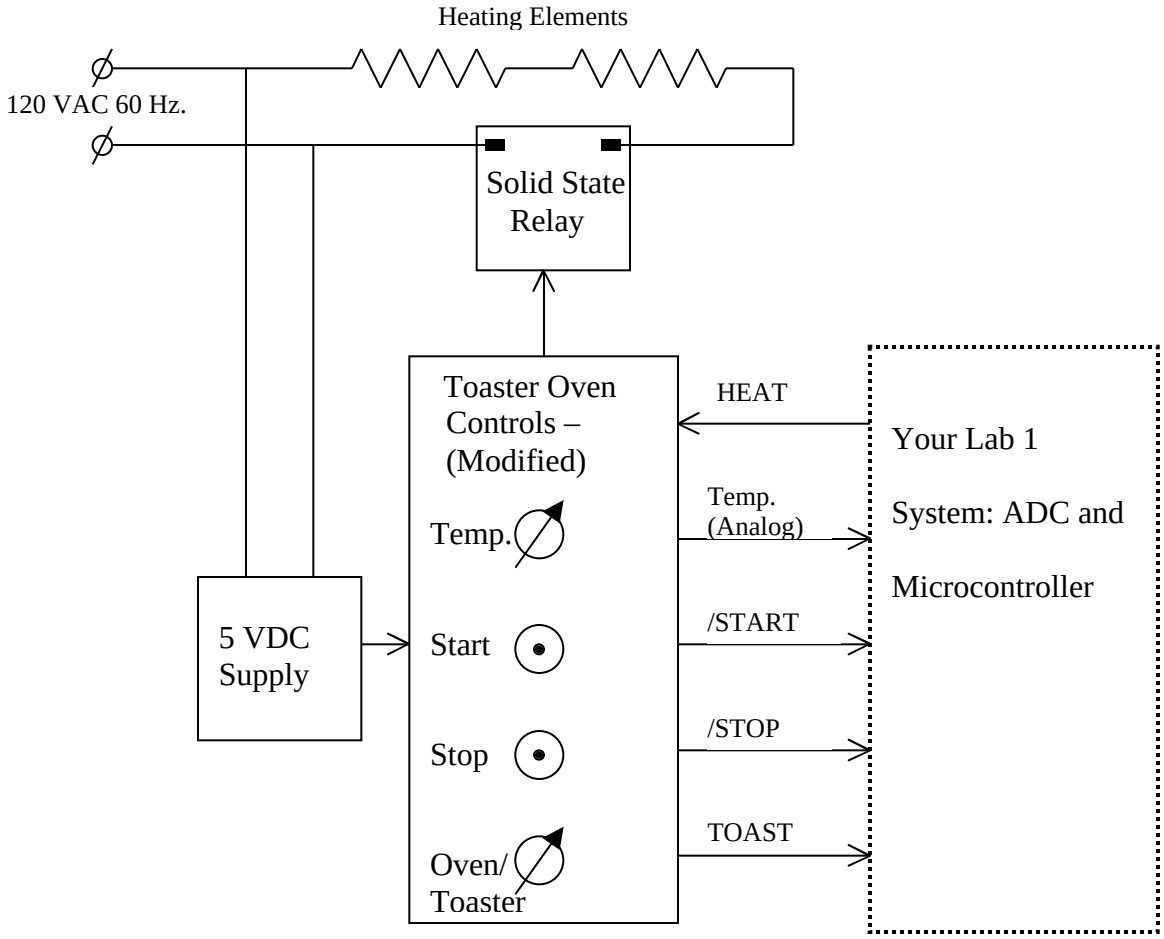

Figure 1: Block Diagram of Toaster Oven System

machine is to act as a toaster, that is, it runs at full heat for a set time determined by the toast control knob. When this signal is LOW, the machine acts as an oven, that is, it runs at partial heat indefinitely. The two pushbuttons are labeled Stop and Start. Like the TOAST switch, we have connected the button contacts to pull-up resistors and bypass capacitors to produce the negative-true, logic signals /STOP and /START. Finally, the toast color knob actuates a potentiometer that we have wired to the power supply. The wiper on this pot supplies a quasi-DC, analog signal called TEMP that the user can set to anywhere between 0.2 and 3.0 volts to represent light to dark toast or low to high temperature oven operation. (The precise voltage levels are not guaranteed. Measure them yourselves to be more exact.)

The TEMP signal poses the problem that the microcontroller does not accept analog signals. (One can buy controllers that do, but they cost more and are less pedagogically effective.) To determine the position of the color/temperature control, your processor will have to use external components to measure the voltage on the TEMP line. Figure 3 shows your system at a more detailed block diagram level. To do A/D conversion of the

TEMP signal, you will have a digital to analog converter (DAC) and a comparator. By programming the DAC at successive levels and reading the resulting state of the comparator output, you can find the TEMP voltage. (The most efficient strategy is probably binary sectioning – the more or less standard method.)

Operation in one of two modes begins when the user presses the start button. The processor must poll the /START input to see when this happens. At that time, the processor looks to see whether the system is in TOAST or OVEN mode. If in toast mode, it turns on the heat by asserting the HEAT line, determines the setting of the TEMP voltage, and times the toast for 15 seconds for the lightest possible setting, 150 seconds for the darkest, and linearly interpolated times in between. If the controls are in oven mode when the start button is pushed, then the processor turns on the heat until told to stop. It sets the amount of heat, however, based on the same measurement of the position of the color/temperature knob as times the toast. The way it sets the heat is by turning the heat on for a fraction of every 10 seconds, a short time being low heat and longer times higher heat. The minimum control setting corresponds to 1.5 seconds out of every 10, while the highest setting is 9 seconds out of every 10. (This method of control is formally called time-proportioning control, and this is a very unsophisticated implementation of it.) The system must stop at anytime if the user presses the /STOP button. (Note that the stop button like the start button is momentary contact and so must be polled frequently enough to be a satisfactory user interface.)

If the user changes the color/temperature knob during operation, your code should lengthen or shorten the appropriate period without malfunction. You may decide for yourself what reaction the system makes when the user changes from oven to toast or vice versa during operation. **You must include comments in your code to indicate what design decision on this point you have made.**

Tolerances: As part of evaluating what you have done, the TA will measure the minimum and maximum and toast times using a stopwatch or digital scope. You may have to set up the scope for the TA. The chip clocks are guaranteed to be within 5 % of nominal, so we are requiring you to get the toasting times and heating duty cycles within ±15 %. There will be a small penalty for exceeding that and a bigger penalty for missing by ±25 % or more. To help in meeting that tolerance you might want to measure the maximum and minimum voltage from the toaster control before designing your software or burning a PIC chip.

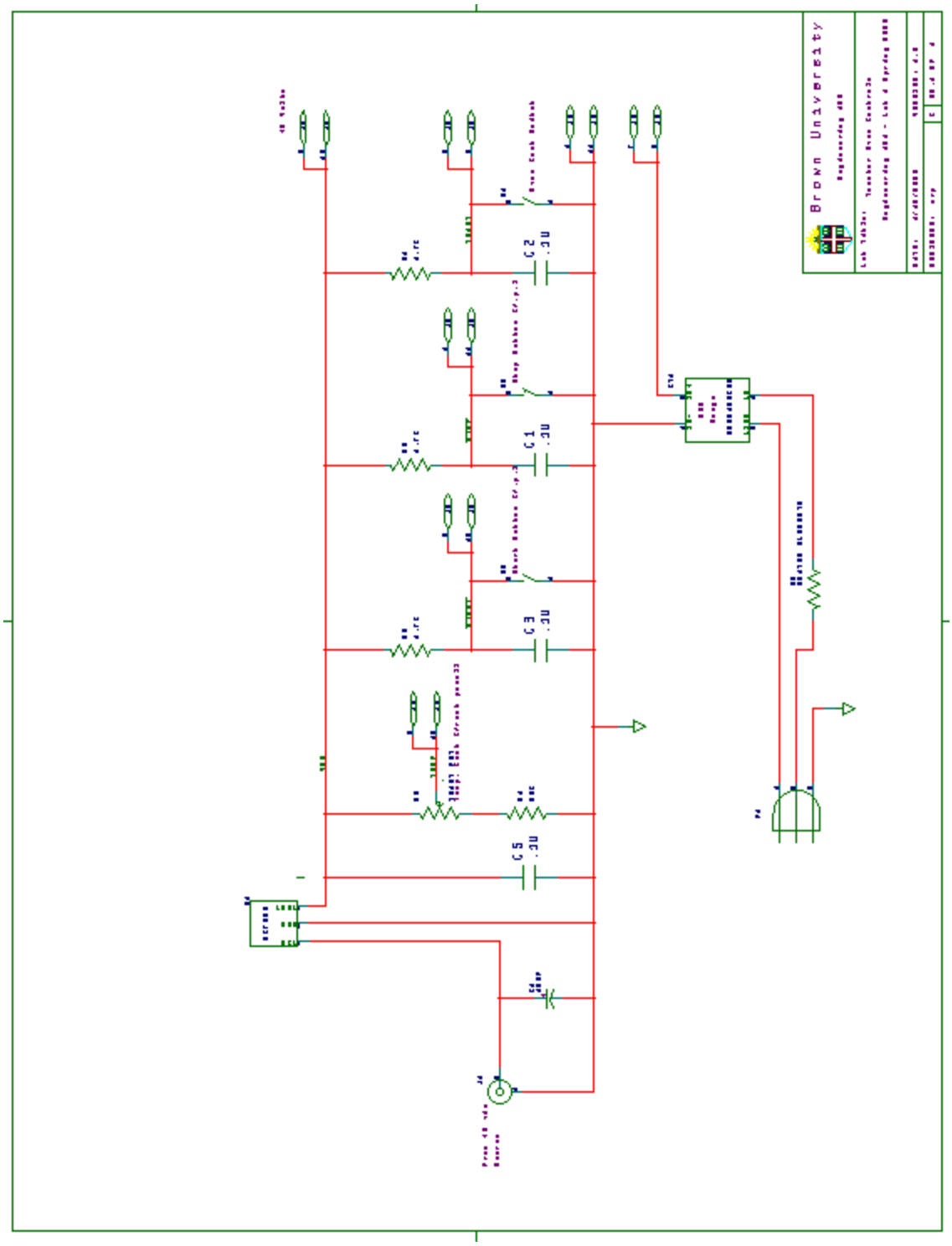

Figure 2: Schematic Diagram of Toaster Oven Controls

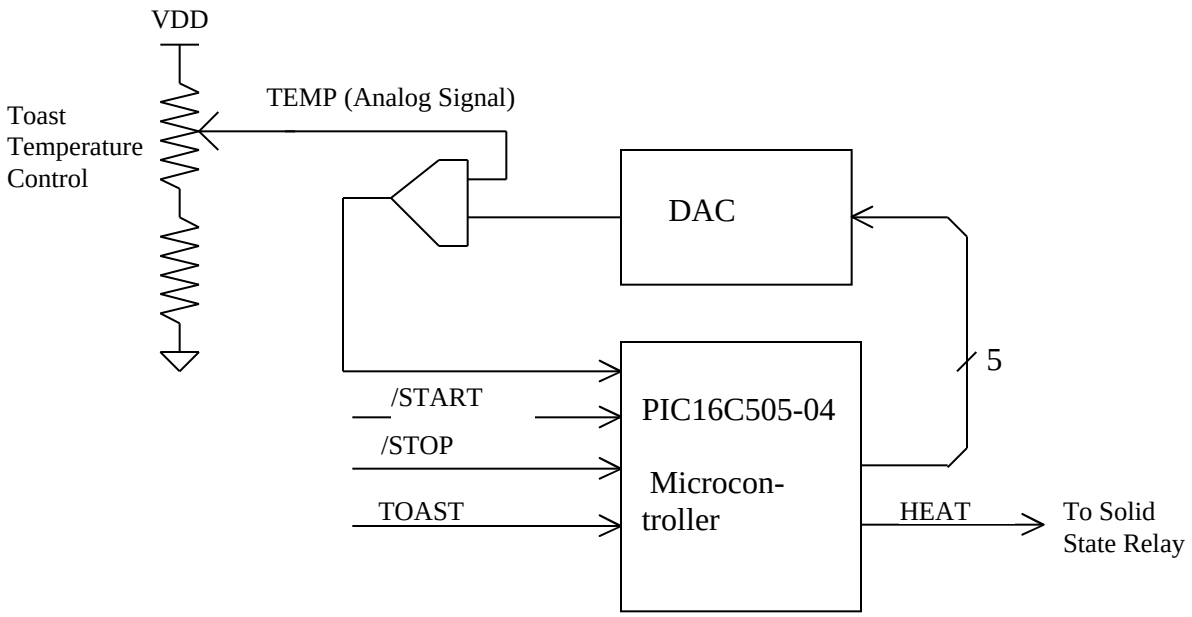

Figure 3: Detailed Block Diagram of Microcontroller System

Table 1: Pin connections on the Lab 1 toaster oven cable			
Signal	Type	Function	Pins
VDD	Power	+5 volt	2, 13
GND	Power	Ground reference	1, 14
/START	Dig. Input	Negative-true, momentary contact signal to start operation	5, 10
/STOP	Dig. Input	Negative-true, momentary contact signal to stop operation	4, 11
TOAST	Dig. Input	Selects toasting or roasting. HIGH implies toasting.	6, 9
HEAT	Dig. Output	HIGH on this line turns on heater elements through solid state relay	7, 8
TEMP	Analog Inp.	Voltage from .2 to 3.0 volts set by user's color/temperature knob	3, 12

Resources: You may pick up a kit for Lab 1 from Mr. Arpie Kaloustian in room 325; we will announce when the kits and lab setups are available. The kit contains three PIC16C505-04's, an LM393 comparator, a single-supply DAC, a protoboard, wire strippers, and bypass capacitors. Any wire or resistors you may need will be available in the lab. Please return resistors to the resistor boxes when you are done and do not scatter them on the floor. Resistors are shared with Engineering 162 and are enough of a problem for that class without your aggravating the situation. There is no charge for the kit, but you must return the materials to the lab TA when you get credit for the lab. (The protoboard, wire strippers, and DAC represent $ 550 for our budget given the size of your class.) If you use all three of the processors before getting the lab to work, you may get

an additional three pieces by again going to Mr. Kaloustian. Arpie works from about 8:00 AM to 4:30 PM weekdays and goes to lunch at 12:00 or 12:30 for an hour, so plan accordingly. The break in supply of parts imposed by this two-part distribution system is deliberate. I want to slow down any headlong plunge down a wasteful path.

You will receive a large part of the PIC16C505 datasheet along with this writeup. The rest of it is available as a "pdf" file on the instructional server at the class website -- www.engin.brown.edu/courses/en164. Please do not print out the whole thing! Much of it covers all the different packages the part comes in and what tests are done on it. Please don't waste paper! Similarly the manufacturer's documentation for the assembler, linker, and simulator software is on-line as help files in the same place. Again be judicious in any printing you do.

Discussion: Before making specific suggestions about your hardware design, I should like to make two more general comments. The first is to salute the designers of the original Black and Decker™ product. This oven was designed and built in Connecticut and sold locally for under $ 40. In a time when almost all consumer appliances are made overseas to save on labor costs, this was a considerable and laudable accomplishment.

My second comment has to do with food and fire hazards. I recognize the temptation to bring bread and food into the lab to test out your design realistically. Please don't! NO FOOD IN THE LAB. Similarly please be careful and sensible about the fire hazard and the chances of burning yourself. Keep flammable materials away from the oven and watch your fingers. These are not joking matters as a recent fatal dormitory fire at Seton Hall College showed. It costs the University several hundred dollars in direct billing any time the fire department has to come, so don't set off the smoke alarms! If there is an alarm, we may well be forbidden to use the apparatus and that would make me unhappy. I like the sense of an actual product – there are many microprocessor-driven toasters and toaster ovens out there now, although most are less expensive than ours. Also, be careful even with the toaster simulator; the simulated heater gets hot enough to hurt if you touch it!

Here are some specific suggestions and considerations around your hardware design:

1. If you wish to use the internal clock of the PIC16C505, you must set the oscillator calibration bits to have it work properly and to have a fairly exact 4 MHz frequency. This is done by the first instruction in the template file. Do not change this instruction.

2. You must "configure" the PIC16C505 to select the timer functionality and default pull-ups for input pins. This is done with a special instruction (OPTION) that writes the contents of the accumulator register to a special register described in section 4.4 of the Microchip data sheet. Setting this should be the first thing in your own code.

3. You will probably want to use the timer facility to simplify setting time intervals for the heat. To do so, requires setting up the timer module to be a user rather than a watchdog timer. This is done at the time you set the oscillator and reset

sources through a separate "configuration" command to the MPASM program, not through code instructions. See remarks in the software hints below.

4. Consider setting the PIC16C505 to have an external reset and using this for the /STOP input. This is not mandatory, but it can simplify the program design.

5. Similarly, you must set up the general purpose input-output port register for the bits that will be DAC and heat control signals. Again, this is an early operation in the program, probably the third thing you do.

6. Have the processor either output some simple bit pattern for a moment at program startup or have it do an ADC operation before reaching any real branch points in the program. This will make it easy to use a logic analyzer on your first chip to see if it is programmed and configured correctly even if it is not fully functional.

7. Consider defining the timer reload value with a #define statement so that it can easily and visibly be changed to speed the time up during simulation and be safely returned to its proper value before programming. (See comments on simulation speed below.)

8. Remember that your LM393 comparator has open collector outputs and that the PIC16C505 is a CMOS part. You must have a pull-up resistor. (4.7 K will do the trick.)

9. Use at least two bypass capacitors (0.1 μf will do nicely) across the VDD to GND connections. Without them, there may be erratic operation.

10. The output of the TEMP signal is a little bigger than the range of the DAC. You will make a more satisfactory system if you load that signal with a resistor from TEMP to GND that brings the maximum voltage down to about 2.7 volts. You may need to experiment a bit, but start with a couple hundred kilohms.

Additional Hints:

Based on last year's experience, here are a couple more hints:

1. Begin by drawing for yourself the circuit diagram of your part of the system. You need this to do the wiring. Your decisions on pin assignments affect the very first couple of lines of code you write, and that effect ripples down throughout the code, affecting test bits and output words. An early diagram will also assure that you are clear about what is going on. I would even suggest that this diagram ought to be neat – even, for those with Engineering 163 background, generated by *Workview Office* perhaps. (I will try to get symbols for the PIC and AD557 into the en163 library on the balcony soon.)

2. You do not have enough pins on the PIC to drive all the DAC inputs. Use only the 5 or 6 most significant bits and ***ground the unused inputs*** to the DAC. See figure 2 of the AD557 data sheet for required gain connections to the DAC.

3. Almost certainly you will want to use the built-in timer for controlling intervals. The use of this timer depends on several things: you have to configure it for program timing rather than watchdog timing, a process done in the same step as you configure the oscillator and reset choices. See the notes on the software for this procedure. You must also set the time base, and probably that should be as long as possible. You can get the timer register to increment roughly every quarter millisecond for a 65 msec count cycle. You must do any counting beyond that in your software.

4. Stopping the toaster oven depends on the stop button, which means this button must either cause a chip reset or be polled often enough never to be missed.

Hints on Using the MPLab Software

We have adapted a template file from Microchip Technology to this particular application and that file can help you get started writing the code. It has the first couple of instructions to calibrate the clock oscillator and set the address range correctly. Then it leaves a blank area for your program. The first thing you do is set up your own working copy of the Microchip Technology assembler and simulator. Then make a copy of the template file, renaming it appropriately for your project. The steps are:

1.) Log onto one of the machines in the Instructional Computing Facility. From the NT desktop, choose **Start -> Microchip MPLab -> Run me First**. This will create a directory called Mplab off your root directory and will copy a complete set of programs and support data sufficient for this class into it. (To conserve disk space with 50 students, we do not copy all the Microchip Technology files. If you ever want to use another processor, its support files can be copied off the server easily.)

2.) From the directory "<your root>\Mplab\User Template", copy the file "toast_templ.asm". In doing so, give it a new name which will identify it as your file for the toaster lab, lab 1. Unfortunately, the Microchip software limits the file name to eight characters. Try to use a name that is unlikely to be shared by other members of the class.

3.) Now start the Microchip software integrated design environment. You do this by the menu choice **Start -> Microchip MPLab -> MPLAB.** Set up your working environment by selecting **Project -> New Project** from the menu bar and enter a name for your project. This must be the same name as the file you just created to hold your assembly code. Next, select the *Development Mode* by hitting the **Change…** soft button in the middle of the Edit Project window. This should be set to the MPLAB-SIM Simulator with a processor of PIC16C505 and click **Reset**. You will be prompted with an import error but this is normal when creating a new project. Now highlight the only line in the *Project Files* section (this will be the name of your project with a [.hex] extension next to

it). Click node properties and verify that the *Language Tool* in the upper left-hand corner is MPASM. Select *OK* and then in the previous window hit the *Add Node* soft button. Type the name of your project in the selection box with the '.asm' extension. If you type a different name in this window, it will change all of the names associated with this project and prompt you accordingly. (A change at this point is not recommended since it will confuse the simulator later) Clicking *OK* will return you to the empty MPLab Window.

4.) Next, open the file you copied in step 2 by selecting **File -> Open** or use the icon on the toolbar.

5.) Write the code and configure the microcontroller. There are several places where configuration takes place. First, one must include an include file with synonyms for configuration bits; the first line of the code should be '#include "P16C505.INC"' as is already done in the template file. Next set the *Option Register* by loading an appropriate byte into the accumulator register and then calling a specific instruction: 'OPTION'. Thirdly, configure the I/O ports by loading the correct byte in the W register and call 'TRIS' with appropriate port name. The description of these values can be found in the specifications of the microcontroller. Finally, in the **Options -> Processor Setup -> Hardware** menu, set the clock type and the master clear functionality. (***Warning Note: You must do this step twice, once now while compiling and simulating and again when you program the chip. You have to set these values a second time even if you have not left the program.*** Failing to set the processor options at programming time is a very simple way to waste a processor.)

To build the project and check for compilation errors, go to the **Project -> Build All** or hit Ctrl-F10. This will bring up a Build Results window with all the errors and warnings in your code. **Double clicking on a line in this window will move you to the corresponding line in the assembly code window.**

6.) Once you have written and compiled the assembly code, you can test it by simulation before you program the controller. In fact, you **MUST** simulate the code first because the micro controllers are **ONE-TIME** programmable. You have only six of them and failure to make the program work within that constraint will make it impossible to get full credit for the lab. There are several additional windows that are helpful when simulating your program. Go to the **Window** menu on the menu bar and open the *File Register* and the *Special Function Register* windows. With these you will be able to monitor the progress of your program as it is simulated. In addition, to simulate the external environment of the microcontroller, click **Debug -> Simulator Stimulus -> Asynchronous Stimulus**. Each button can be assigned to stimulate a pin. By right clicking you can set the mode (probably the most helpful is *Toggle*) and also *Assign Pin*.

All of the functionality needed to simulate your program can be found in the **Debug -> Run** menu. The most helpful of these functions are *Run*(F9), *Step*(F7), and *Halt*(F5). Additionally, after a successful build, one can set breakpoints as well. Right click on a line, and a menu will appear by which you can set a breakpoint. On that same

menu, the *Run to Here* is also a very helpful command. However, breakpoints disappear when you use the *Run to Here* option.

7.) Finally, when you are ready to program the microcontroller, go to the PICSTART™ programmer in room 196. You can retrieve your programming file by FTP from the Instructional Computing Lab. Run the Mplab software and select the menu item: **Picstart Plus -> Enable Programmer**. (Be careful, if you do this on a computer without a programmer, the MPLab software may crash). **Set the hardware configuration as in step 5,** and then write the assembled code to the processor.

Extra Hints on Simulation:

1) The timing loops of your program have many, many iterations. When debugging, the simulation of these can take a very long time. Once several iterations of the loop have been run, skip the rest by altering the register which holds the loop counter. To do this, go to the *File Register* window and right click on the register you want to change and click on *Fill Registers*. Set the value in the Data/Opcode section and write the value into the register. Another way to shorten the simulation is to use a smaller range for the timer module by temporarily adjusting its reload value as discussed above. From the *File Register* window, you can also modify your configuration registers without recompiling and rerunning your program. This will shorten the simulation time considerably.

2) Save your work frequently. The MPLab software is not bug free. I have come across several of them. All are easily handled, but they are annoying nonetheless.