ECE263

Lab 1

Spring 2013

CodeWarrior & Dragon 12P

Revised: 20 Nov 12 DAL

(This page intentionally left blank)

Object: This lab will introduce the use of the CodeWarrior software package used to generate machine readable programs for the FreeScale MC9S12DG256 microcontroller. You will use CodeWarrior to set up a typical project, enter a simple source file, use the assembler and linker to generate the object file and then use the P&E Multilink to download the object file into the Dragon 12P Development board to test the program.

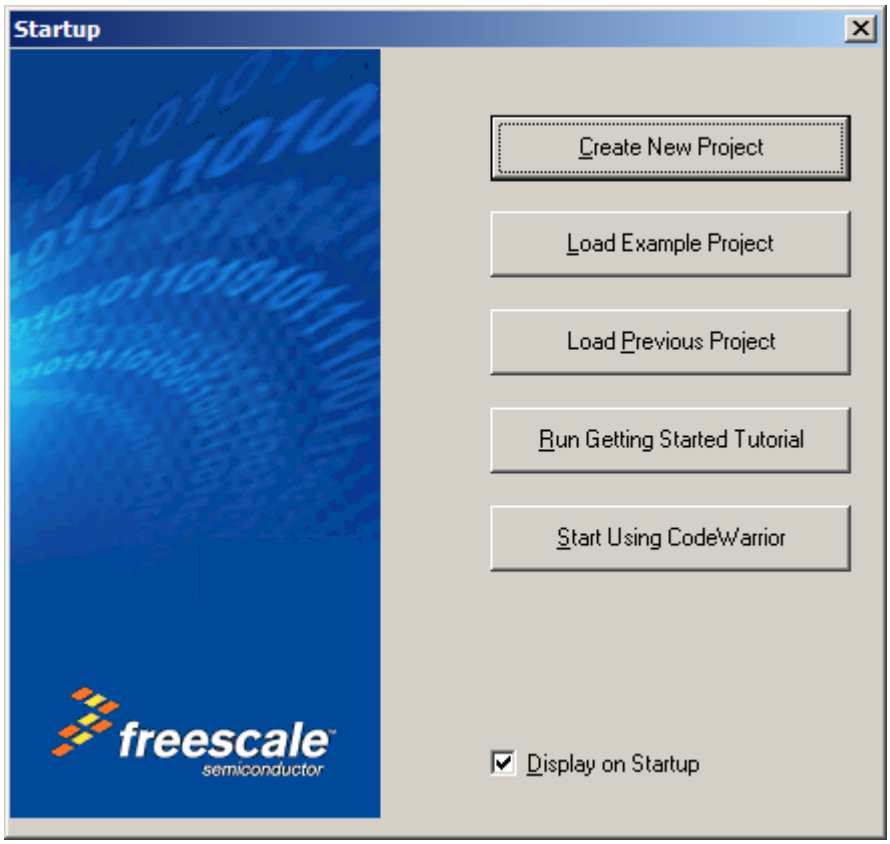
Material: CodeWarrior software package
Dragon 12 Plus Development Board
P&E USB Multilink
CPU2 Reference Guide (CPU12RG)

(Note: CodeWarrior is loaded into the computers in Rm 214 and Rm 218. You can also download a freeware version of the software from FreeScale - see the instructor for details)

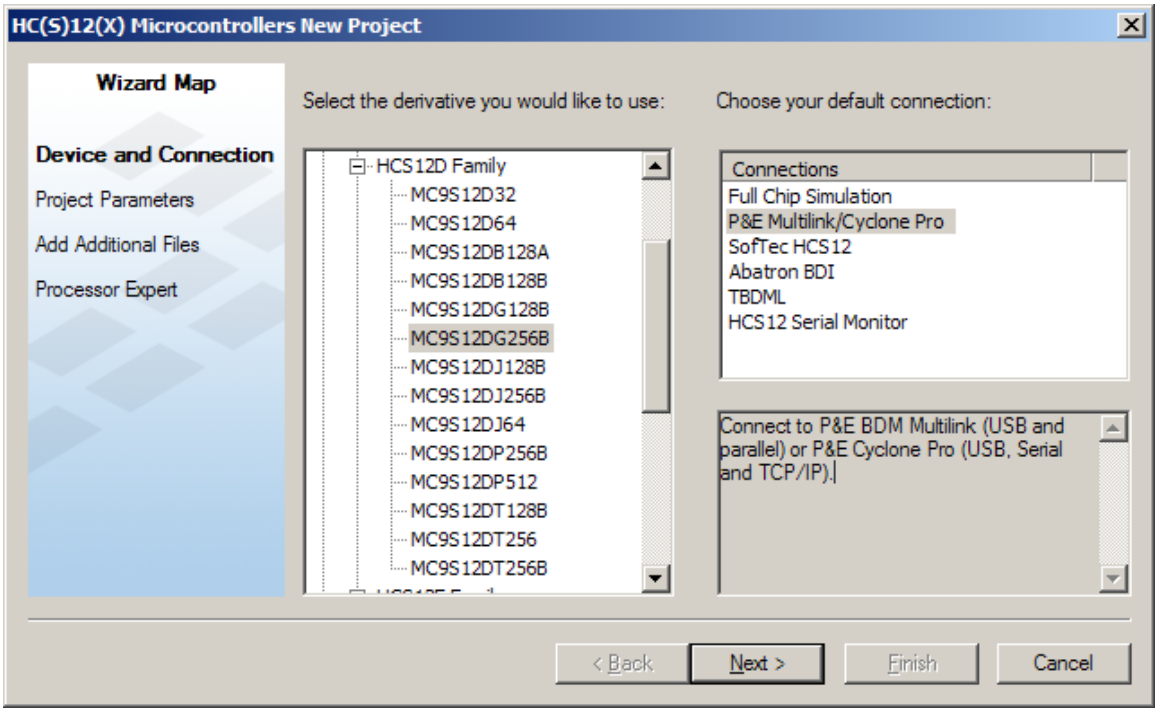
Preparation: Before coming to lab, read through the lab, esp. the sample program that you will be testing. Write up a brief explanation of what the program is doing and what you expect to see.

Procedure: Working with your lab partner, complete the following steps

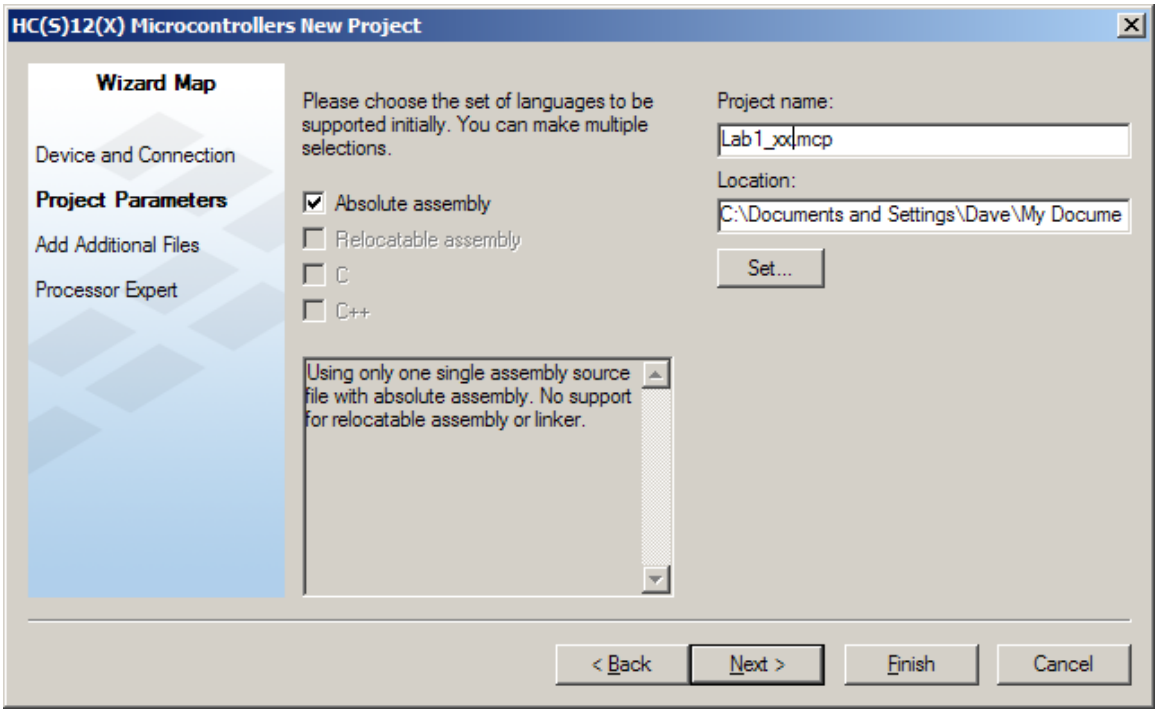
1. Start CodeWarrior either from the PROGRAMS menu or by double clicking the icon on the desktop.
2. The program should open a new window and display the following splash screen.



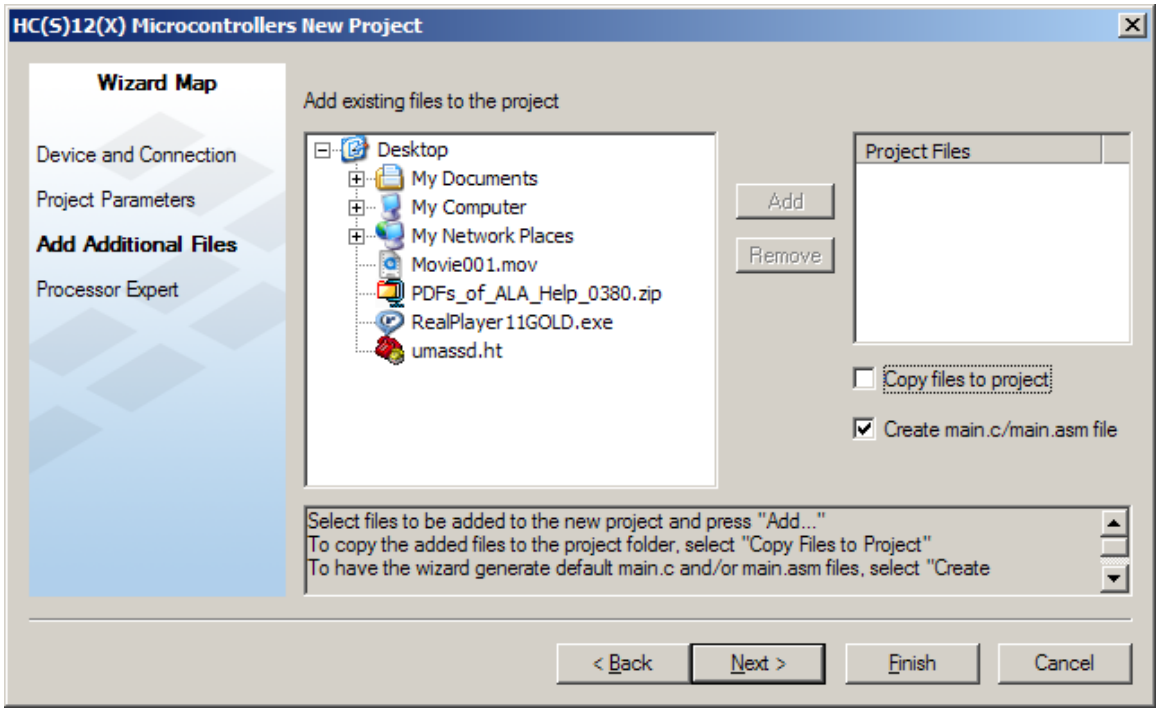
3. Click on *Create New Project*. You should then see the following. Expand the entries and select the MC9S12DG256B in the derivative window and select P&E Multink/Cyclone Pro in the default connection window then click *Next*.



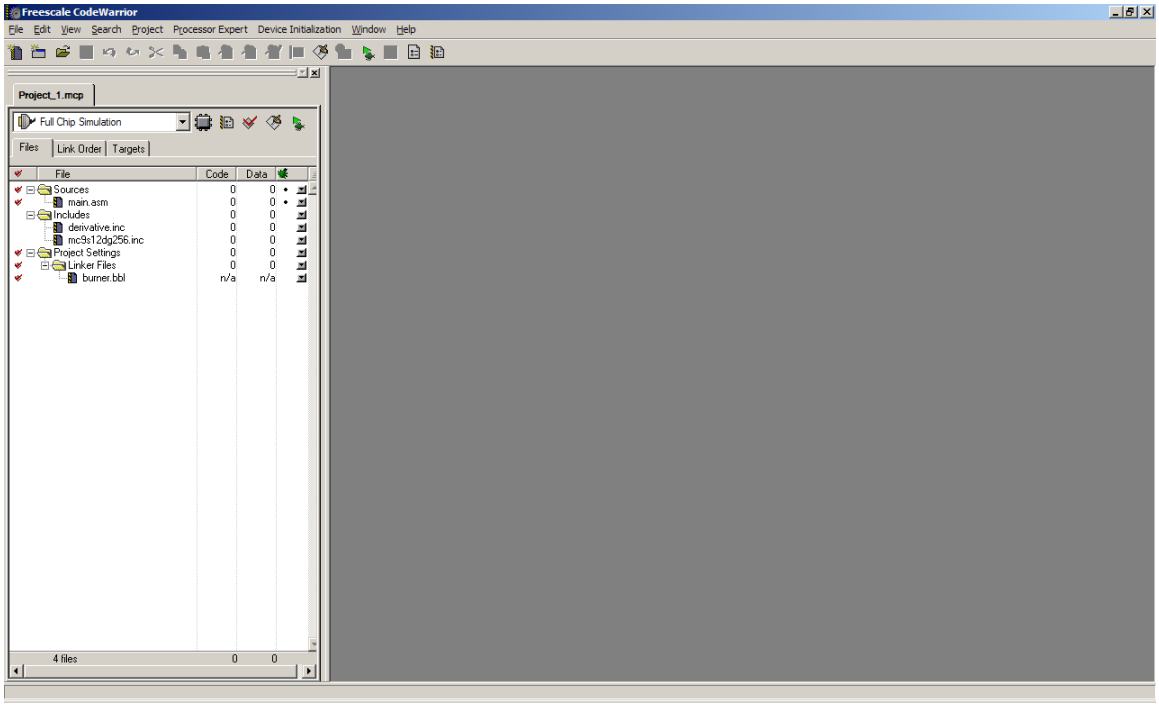
4. Now you need to set the type of project, the project name and location using the following screen. Uncheck the box next to *C* then check the box next to *Absolute Assembly*. Enter a project name such as *Lab1_xx*, replacing the *xx* in the project name with your initials. Enter a location for the project files to be stored. You need to have read/write privileges to the file area so the location you enter should be on a USB drive or in your home directory on the "Z-drive". A number of sub-folders and files will be generated for the project so make sure you have sufficient space available. NOTE: you should keep each lab project in a <u>different folder</u> to avoid confusion since so many similar sub-folders and files will be generated for each project. Click *Next* when you are done.



5. The program will now open the following screen. Uncheck the box labeled *Copy files to Project* and check the box labeled *Create main.c/main.asm file*. Click *Finish* when you are done.



6. The following screen should now open.



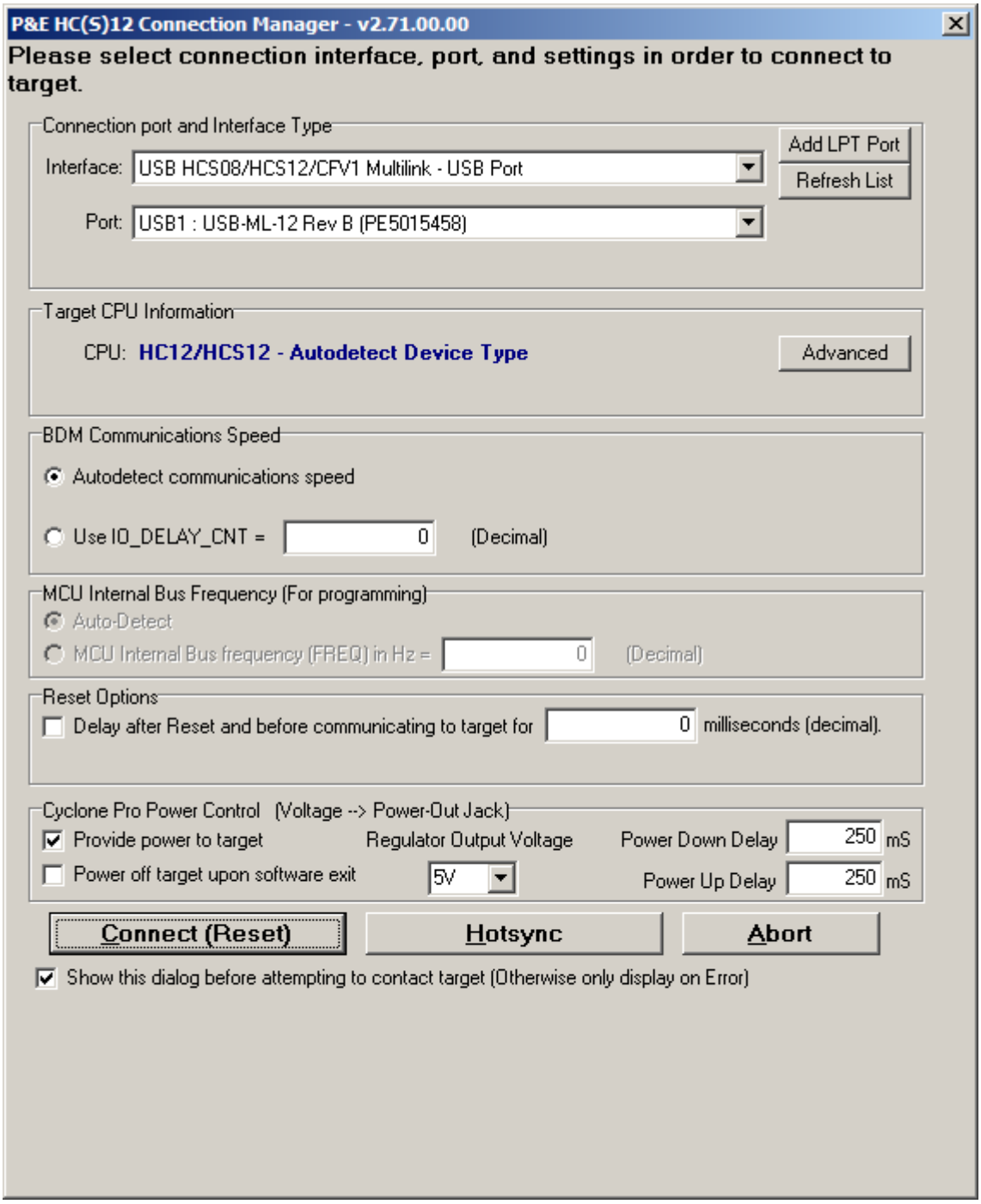
7. In the Project pane under Sources, you will see a file called *main.asm*. This is a default example file that we will replace with a different program. Using the built in editor modify the existing code so that matches the following:

```
; export symbols
        XDEF Entry, _Startup      ; export 'Entry' symbol
        ABSENTRY Entry            ; for absolute assembly: mark this as entry point

ROMStart    EQU  $C000            ; absolute address to place code/constant data

; code section
        ORG   ROMStart

Entry:
_Startup:

;  Program 1
            LDAA     #$55
OUTER:      LDX      #$1000
INNER:      STAA     1,X+
            CPX      #$1040
            BNE      INNER
            COMA
            BRA      OUTER
            NOP
```

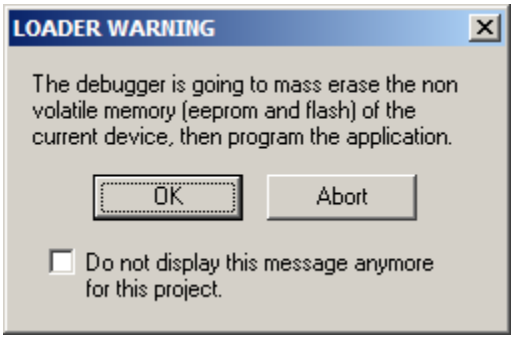
```
 ; Program 2
            LDX     #$1000
LOOP:       STX     2,X+
            INCB
            CPX     #$1100
            BNE     LOOP
HERE:       BRA     HERE

;*****************************************************************
;*              Interrupt Vectors                *
;*****************************************************************
            ORG     $FFFE
            DC.W    Entry       ; Reset Vector
```

Note: 1. the above lines all start in column 1
2. anything following the ; is a comment
3. keep the fields lined up as above using tabs and/or spaces

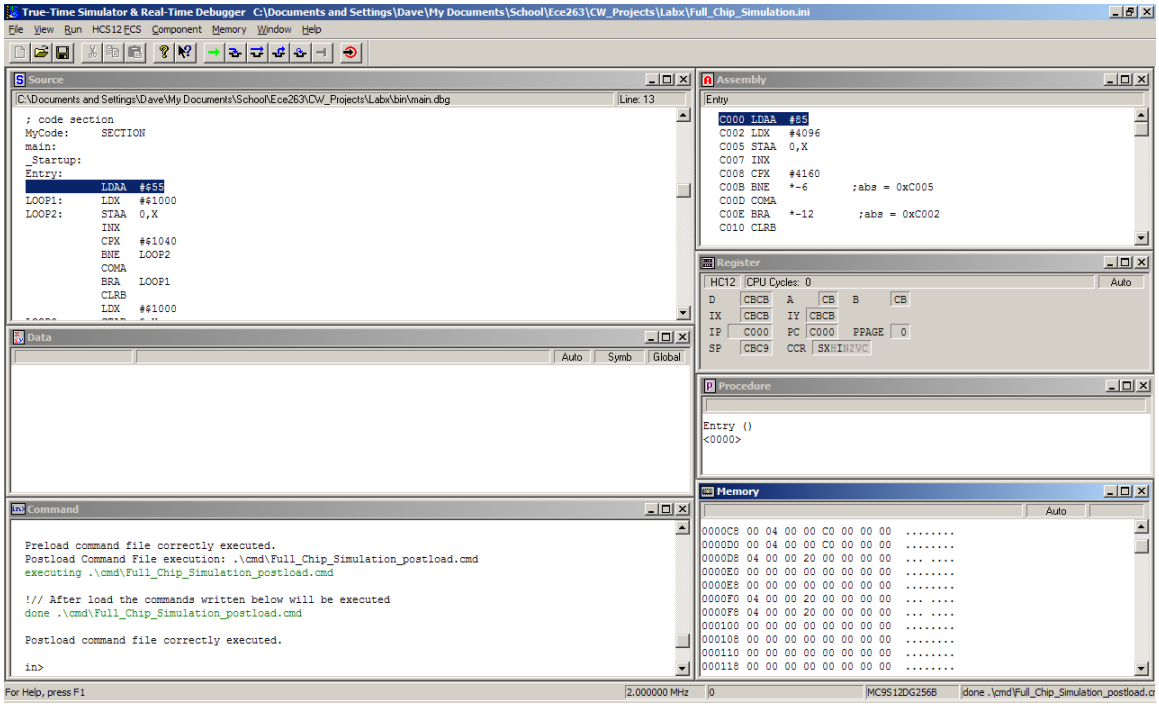
8. Once you have the source code entered you can generate the machine readable object file using the built-in *Make* function. *Make* calls the assembler and then the linker and will report any syntax errors. To run *Make* do one of the following:
   a. click on the *Project* tab at the top of the window, and select *Make* in the drop down window,
   b. hit the F7 key,
   c. or click on the Make icon at the top of the Project pane (fourth icon to the left of Full Chip Simulation).
9. If you have not entered the program correctly, *Make* will report the errors that must be fixed before you can continue.
10. You can see the output of the assembler by running the *Disassemble* function. To call *Disassemble* do one of the following:
    a. click on the *Project* tab at the top of the window, and select *Disassemble* in the drop down window,
    b. hit <ctrl>, <Shift>, and F7 keys simultaneously,
    c. or right click mouse in the *main.asm* Window and *Disassemble* select in the pop-up Window.
11. When you get an error free assembly, print the contents of the Disassembly window to include in your Lab Report.
12. We are now ready to download the object file into the MC9S12DG256 microcontroller on the Dragon 12P. In the Project pane of the CodeWarrior window click on the Debug icon. The following window should pop up asking you to connect to the hardware. Make sure the box labeled *Interface* has the *USB HCS08/HCS12/CFV1 Multilink – USB Port* selected. In the box labeled Port make sure that the *USB-ML-12-Rev x* is selected. In the BDM Communication Speed box make sure the button for *Autodetect communication speed* is selected. Finally click on the *Connect* button on the bottom of the window.


P&E HC(S)12 Connection Manager - v2.71.00.00
Please select connection interface, port, and settings in order to connect to target.
Connection port and Interface Type
Interface: USB HCS08/HCS12/CFV1 Multilink - USB Port
Port: USB1 : USB-ML-12 Rev B (PE5015458)
Add LPT Port
Refresh List
Target CPU Information
CPU: HC12/HCS12 - Autodetect Device Type
Advanced
BDM Communications Speed
Autodetect communications speed
Use IO_DELAY_CNT = 0 (Decimal)
MCU Internal Bus Frequency (For programming)
Auto-Detect
MCU Internal Bus frequency (FREQ) in Hz = 0 (Decimal)
Reset Options
Delay after Reset and before communicating to target for 0 milliseconds (decimal).
Cyclone Pro Power Control (Voltage --> Power-Out Jack)
Provide power to target
Power off target upon software exit
Regulator Output Voltage
5V
Power Down Delay 250 mS
Power Up Delay 250 mS
Connect (Reset)
Hotsync
Abort
Show this dialog before attempting to contact target (Otherwise only display on Error)

13. If everything is connected and working properly you should see the following pop-up window.



14. Click OK. This will cause the non-volatile flash memory to be erased and replaced with the object file that you are going to debug.
15. You should now see the following screen. There is a main window with a number of smaller windows within it. These sub-windows can be resized or rearranged as you prefer. You can also add multiple copies of the windows if you desire. This is especially useful for the memory window as described later.



16. In the window labeled *Source* you will see a view of your source program in the language in which it was written (i.e. assembly or C). This allows you to "source level debugging". Right clicking the mouse in this window gives access to a number of commands for running your program. These commands will be described later.
17. In the window labeled *Assembly* you will see the program in assembly code regardless of the source language. The left column is the memory address of the location of the program and the assembly code is on the right. Place the cursor in the Assembly window and right click the mouse. You again get access to commands for running your

program along with commands to format the display of the data. In the pop-up window, click on *Display* and then on *Code* – the window will now display the machine code of the program in hexadecimal format. Right click again, this time click on Format then on Hex – this changes the display of numbers in the assembly code to be in hexadecimal rather than decimal. (Note: in this window hex notation uses the 0xA234 format rather than $A234)

18. The window labeled *Registers* shows the current contents of the various registers of the Programmers Model of the MC9S12 processor. Right clicking in this window will allow you to change the format of the numbers displayed.
19. The *Memory* window shows the contents of a block of memory. The left column shows the address of the data, the next eight columns show the value of the data stored at that location, and the last columns show the ASCII equivalent of the data. If you expand the width of the window then you get more columns of memory data per line. You can change a specific location by double clicking on a number and then entering a new value. Right clicking the mouse in this window gives access to a number of options such as formatting the way the data is displayed, changing the address of the block displayed, and altering the contents of the memory. Right click in this window then on Address and enter 1000. In the Memory window you should see the value of all of the locations which at this point will contain random values. Try modifying some memory locations.
20. The remaining windows will be discussed in a later lab.
21. The data being displayed in the *Source* and *Assembly* windows is dependent upon the value in the program counter as displayed in the PC block of the Registers window. Double click on the PC block and change the value to $D000. Select the window labeled *Assembly* and then execute a *Single Step* by pressing the F11 key and observe the change to the display in the Assembly window. The source window does not change however because there is no corresponding source program data at that memory location.
22. Now reset the processor by clicking on the reset icon (far right) on the toolbar. The program counter magically returns to $C000 which is the start of your test program. (Note: this process will be explained later). In the Source window the first executable line should be highlighted and in the *Assembly* window the line labeled C000 will also be highlighted.
23. To start testing the program, select the *Source* window. We will first single step through the program either by pressing the F11 key or by clicking on the single step icon on the toolbar (to the right of the green arrow). Single step through the program a few dozen times and as you do this, observe the display in the other windows. What is being displayed? Is the program doing what you expected?
24. Single stepping allows us to execute one instruction at a time, however once we see that the flow is correct we would like to be able to run the program at "full speed" until we get to a new section that has not been tested. This is especially true when we encounter repetitive loops such as in the test program. Now we will insert a breakpoint in the program. A breakpoint is a type of "trap" that will halt the processor when it is encountered and will return control to the debugger.
25. To enter a breakpoint point to the line BNE INNER and click the right mouse button. In the pop-up window select Set Breakpoint. A red indicator should show up in the left margin on the selected line.
26. Click on the *Run* icon on the toolbar (green arrow). Each time you hit Run, the program will execute at full speed until it hits the breakpoint. How many lines are being executed each time? What is changing in the other windows?
27. Disable the breakpoint and set another on the line BRA OUTER using the procedure described above. Hit RUN – what changed in the other windows?

28. Re-enable the first breakpoint and hit run several times. Is the program doing what you expected?
29. Disable all of the breakpoints and hit *RUN*. The program is now running at full speed – what can you see happening in any of the windows?
30. Halt the processor by clicking on the *Halt* icon (to the left of the Reset icon). What do the windows show now? What other way(s) can you activate the Run and Halt commands?
31. The whole test program has not been executed – why not?
32. Point to the line CLRB and click the right mouse button. In the pop-up window select Set Program Counter. This forces the PC to point to that location in the program. You will see that the corresponding line in the assembly window will be highlighted as well. What are some other ways to force the debugger to start execution at a specific line?
33. Using the single step, breakpoints, etc. go through the operation of the second part of the test program and then describe what it is doing.
34. Spend the remaining time trying different aspects of the debugger. You will be using it for every lab this semester.

Lab Report: see Lab Guidelines