ELEC 3040 – Electrical System Design Lab
ELEC 3050 – Embedded System Design Lab

Lab Session 1 – Project Creation and Debugging

The objective of this laboratory session is to become familiar with the process for creating, executing and debugging application programs with the *CodeWarrior* Development Studio for the Freescale MC9S12C32 microcontroller, which is mounted on a Wytec "DragonFly12 Plus" development module. Two "projects" will be created and tested, one in assembly language and the other in C. This will demonstrate the basic setup of an application project in each language, as well as the process for transferring the program to the microcontroller and debugging it.

The second objective of this laboratory session is to begin utilizing the engineering communication skills discussed in the lecture. Specifically, an ***engineering notebook*** is to be maintained by each student, documenting all designs, lab procedures utilized, observations of the results of each test or procedure, and any errors detected and their solutions. Also, a ***two-page memo*** documenting the activities of the first lab session is to be written and submitted by each member of the team (all writing assignments are listed on the syllabus and on the web site.)

The platform on which projects will be built and tested is the Digilent *Electronics Explorer Board* (EEBOARD), shown in Figure 1. The EEBOARD has two breadboards, plus connections to built-in power supplies and the hardware elements of various debugging and test instruments (digital I/O, LEDs, switches, oscilloscope, waveform generator, logic analyzer, etc.) The EEBOARD is controlled from a USB-connected PC via the *WaveForms* graphical user interface, shown in Figure 2.

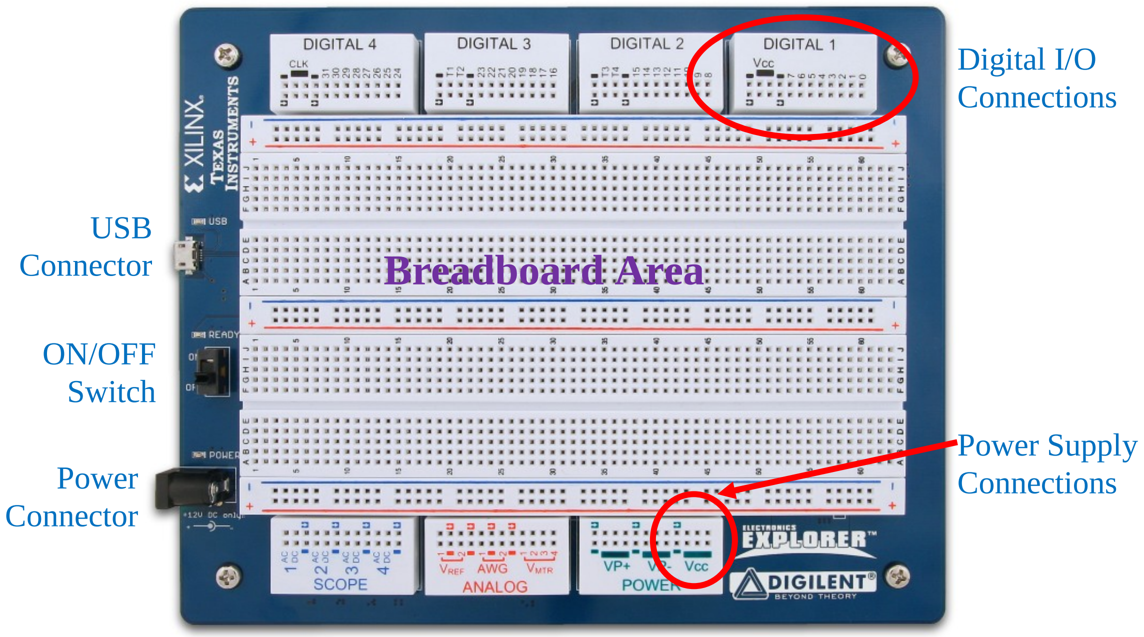
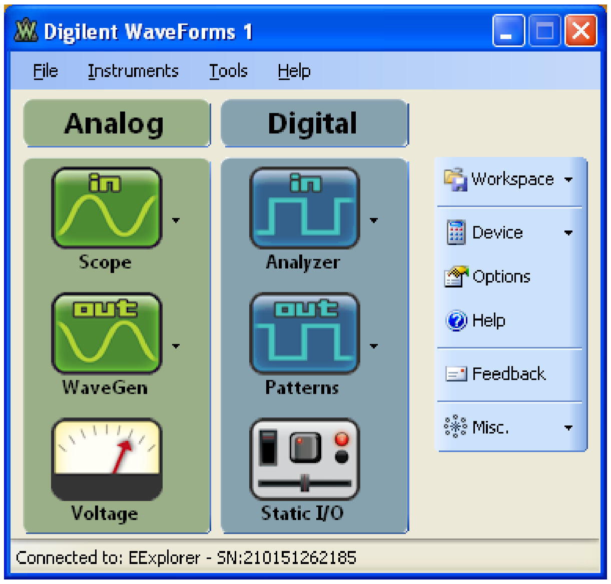


Figure 1. Digilent Electronics Explorer Board (EEBOARD)
http://www.digilent.cc/Products/Detail.cfm?NavPath=2,842,883&Prod=EEBOARD



Connected to: EExplorer - SN:210151262185

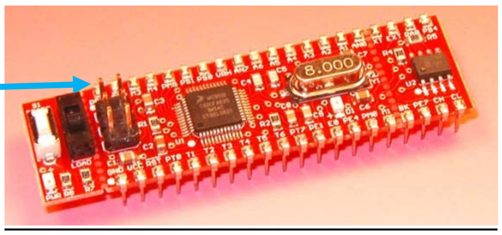
Figure 2. Digilent *WaveForms* - EEBOARD User Interface

For the first week's labs, the instructor will lend a DragonFly-Plus module and an EEBOARD to each team. Those will be collected at the end of the lab period, and each team will then be issued an EEBOARD kit to use for the duration of the semester, which includes a lockable case that will hold the project board, power supply, USB cable, and other parts. **Prior to the start of the second week of lab, each team must purchase their own DragonFly12-Plus module from the Scientific Supply Store in 211 Science Center Lab Building.** All components required for subsequent labs will also need to be obtained from the Scientific Supply Store.

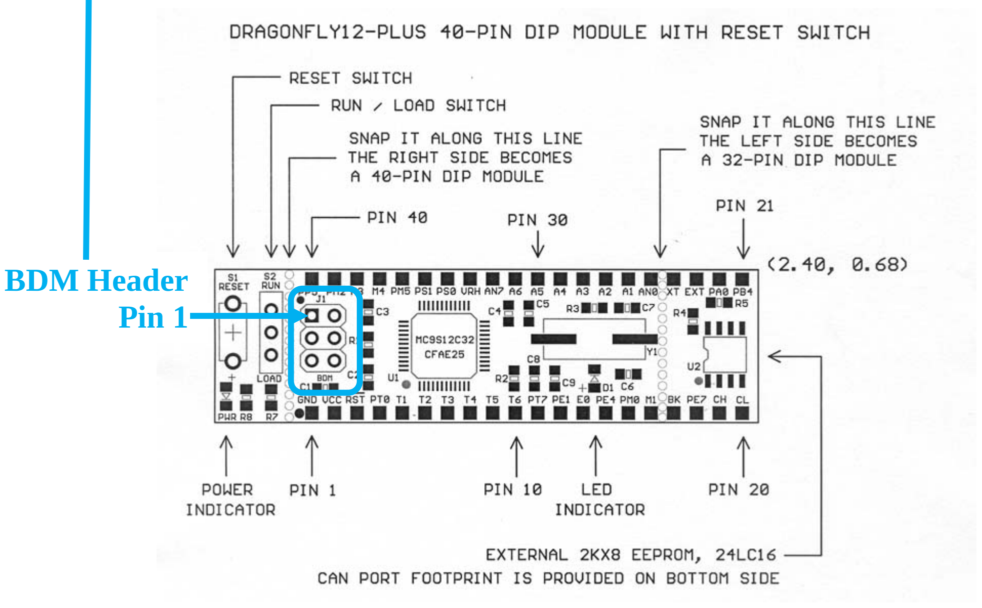
The DragonFly12 Development Module

Each lab position includes a personal computer, a power supply, and a P&E Microsystems *Background Debug Mode* (BDM) module (pictured in Figure 4) with a USB cable. The "DragonFly12-Plus" (http://www.evbplus.com/Dragonfly12P_9s12/DIP40_9s12.html), pictured in Figures 3(a)-(c), is a 40-pin Dual-In-Line (DIP) module containing a Freescale MC9S12C32 microcontroller, a reset button, a run/load switch, and a 6-pin BDM header, to which the BDM module is to be connected for downloading and debugging programs. The MC9S12C32 microcontroller, from the Freescale HCS12 device family discussed in the ELEC 2220 text book, contains 32K bytes of flash memory, 4K bytes of RAM, and a variety of input/output and other peripheral functions. The pin numbering on the DragonFly12-Plus is shown in Figure 3(b)-(c), and the DragonFly12-Plus schematic diagram is linked to the course web page. Note that a number of input/output port pins on the 48-pin MC9S12C32 chip are not accessible on the 40-pin DragonFly12-Plus module. This will be examined further in Lab 2.

When connecting the BDM module, please pay particular attention to the orientation of the ribbon cable on the 6-pin BDM header, shown in Figures 2 and 3. ***The red conductor on the BDM ribbon cable MUST be aligned with pin 1 of the BDM header when connecting the BDM module, or damage to the microcontroller and/or BDM module can occur.***



(a) DragonFly12-Plus photo



BDM Header
Pin 1
DRAGONFLY12-PLUS 40-PIN DIP MODULE WITH RESET SWITCH
RESET SWITCH
RUN / LOAD SWITCH
SNAP IT ALONG THIS LINE THE RIGHT SIDE BECOMES A 40-PIN DIP MODULE
SNAP IT ALONG THIS LINE THE LEFT SIDE BECOMES A 32-PIN DIP MODULE
PIN 40
PIN 30
PIN 21
(2.40, 0.68)
MC9S12C32 CFAE25
POWER INDICATOR
PIN 1
PIN 10
LED INDICATOR
PIN 20
EXTERNAL 2KX8 EEPROM, 24LC16
CAN PORT FOOTPRINT IS PROVIDED ON BOTTOM SIDE

(b) DragonFly12-Plus module details

1	GND	40	PP5/KWP5/PW5
2	VCC	39	PM2/RXCAN1/RXCAN0/MISO0
3	/RESET	38	PM3/TXCAN1/TXCAN0/SS0
4	PT0/IOC0	37	PM4/RXCAN2/RXCAN0/RXCAN4/MOSI
5	PT1/IOC1	36	PM5/TXCAN2/TXCAN0/TXCAN4/SCK0
6	PT2/IOC2	35	PS1/TXD
7	PT3/IOC3	34	PS0/RXD
8	PT4/IOC4	33	VRH
9	PT5/IOC5	32	AN7/PAD07
10	PT6/IOC6	31	AN6/PAD06
11	PT7/IOC7	30	AN5/PAD05
12	PE1/IRQ	29	AN4/PAD04
13	PE0/XIRQ	28	AN3/PAD03
14	PE4/ECLK	27	AN2/PAD02
15	PM0/TXCAN0/RXB	26	AN1/PAD01
16	PM1/TXCAN0/TXB	25	AN0/PAD00
17	BKGD	24	XT/ OSC. output
18	PE7	23	EXT/ OSC, input
19	CAN -	22	PA0
20	CAN +	21	PB4

(c) DragonFly12-Plus pin-outs

Figure 3. Wytec DragonFly12-Plus 40-pin DIP Development Module.
(http://www.evbplus.com/Dragonfly12P_9s12/DIP40_9s12.html)

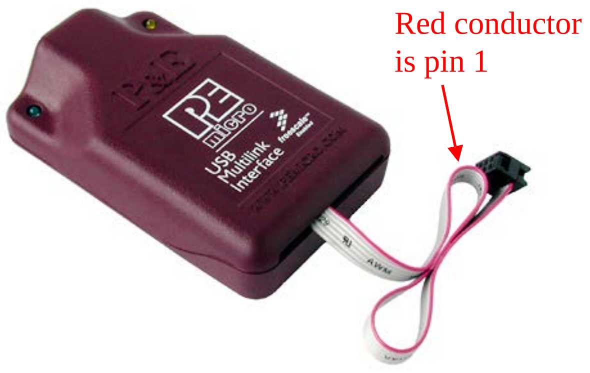


Figure 4. P&E USB BDM Multilink module and cable

Creating Projects: Program 1 – Assembly Language

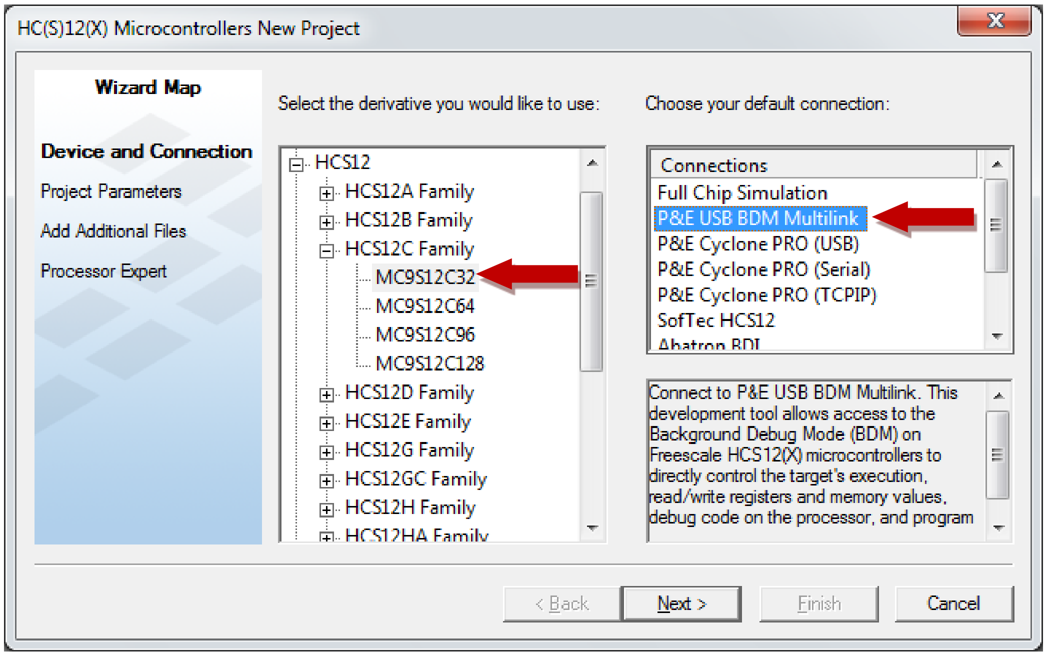
CodeWarrior (also used in ELEC 2220) is an integrated development environment (IDE) that is flexible, in that it can be used to develop projects in assembly language, C, C++, or any combination of these languages, with the ability to debug on the target hardware or on a simulated microcontroller.

To create a new project, open *CodeWarrior.* If prompted, click on **"Create New Project"**, or else select **File -> New Project** (select "New Project" from the "File" menu), which generates the new project "wizard" form in Figure 5. This wizard will guide you through the project setup process.

In the list of microcontroller derivatives in this form, as shown in Figure 5, expand HCS12, then expand "HCS12C Family" and select MC9S12C32, corresponding to the microcontroller chip on your DragonFly12-Plus module. For reference, the HCS12 data sheet is linked to the course web page.

In the list of default connections on the right side of this form, select **P&E USB BDM Multilink**. This allows *CodeWarrior* to transfer your program to the flash memory of the microcontroller through a USB cable connected to the P&E USB Multilink BDM module, which should be plugged into the 6-pin BDM header on the DragonFly12 module. You can debug the program in the target hardware via the BDM using the same user interface as for the simulator that was used in ELEC 2220. If you wish, you can also use the simulator to help debug programs at home, although the simulator will not include any external devices connected to your microcontroller.

After selecting the derivative and connection, click **Next** to continue the setup.



Select the derivative you would like to use:

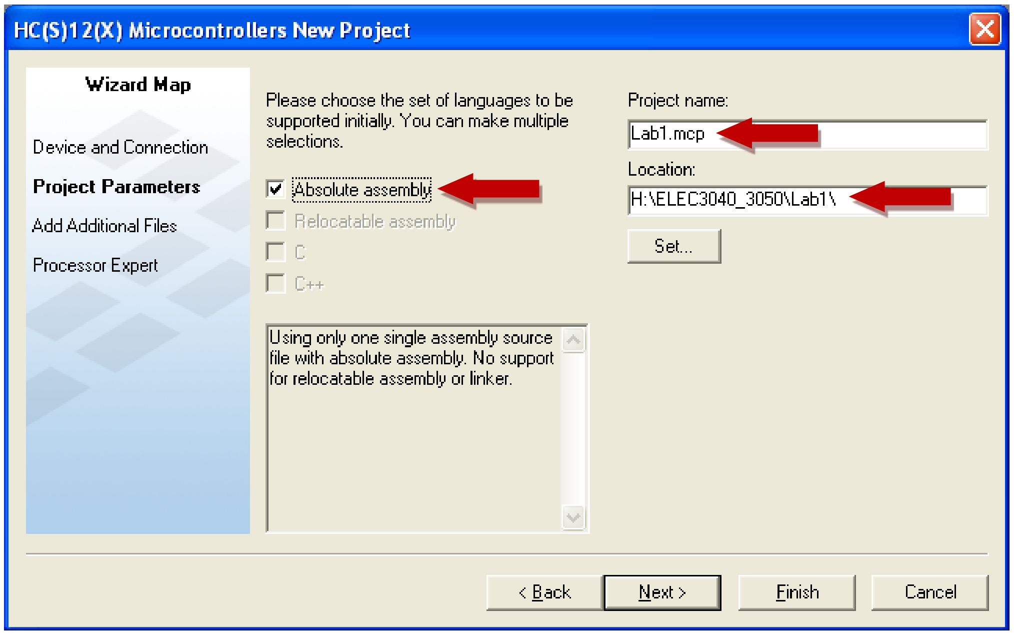
- HCS12
  - HCS12A Family
  - HCS12B Family
  - HCS12C Family
    - MC9S12C32
    - MC9S12C64
    - MC9S12C96
    - MC9S12C128
  - HCS12D Family
  - HCS12E Family
  - HCS12G Family
  - HCS12GC Family
  - HCS12H Family
  - HCS12HA Family

Choose your default connection:

Connections
- Full Chip Simulation
- P&E USB BDM Multilink
- P&E Cyclone PRO (USB)
- P&E Cyclone PRO (Serial)
- P&E Cyclone PRO (TCPIP)
- SofTec HCS12
- Abatron BDI

Connect to P&E USB BDM Multilink. This development tool allows access to the Background Debug Mode (BDM) on Freescale HCS12(X) microcontrollers to directly control the target's execution, read/write registers and memory values, debug code on the processor, and program

Figure 5. New project wizard. Select MC9S12C32 microcontroller and P&E USB BDM Multilink connection.



Please choose the set of languages to be supported initially. You can make multiple selections.

- [✓] Absolute assembly
- [ ] Relocatable assembly
- [ ] C
- [ ] C++

Using only one single assembly source file with absolute assembly. No support for relocatable assembly or linker.

Project name: Lab1.mcp

Location: H:\ELEC3040_3050\Lab1\

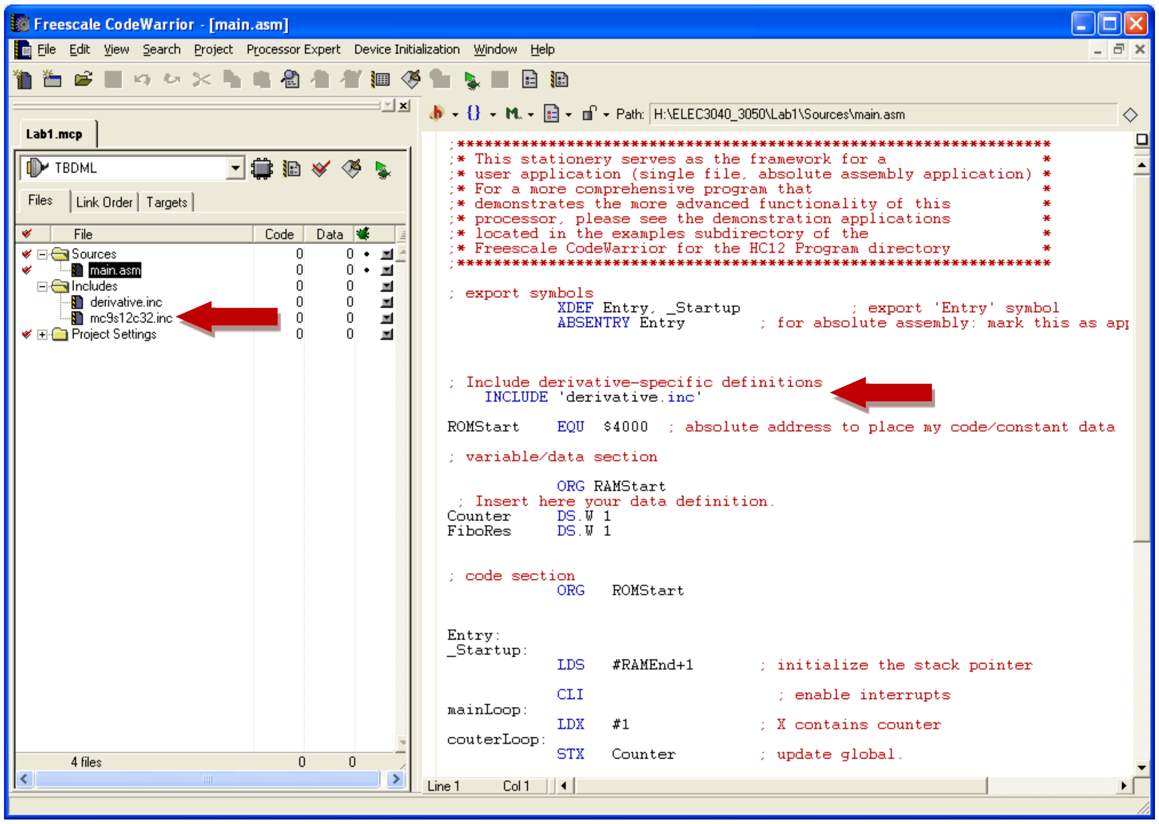
Figure 6. Select the programming language(s), project name and location.

On the next screen (Figure 6), select the language(s) to be used for the project. You can select more than one if you plan to mix assembly language, C and/or C++. For this program, unselect "C" and then select "Absolute assembly", since you will put your entire program in a single file, with the program and data locations fixed at specified memory addresses. If a project is to be partitioned into multiple files that need to be linked together, you would select "Relocatable assembly".

In the top right corner of this form, enter a project name (ex. Lab1.mcp) and the location at which the project is to be stored. **In this course, save all projects on your network (H:) drive, to ensure that they are not lost when maintenance is done on the lab computers.** This will also allow you to access your project files from any computer on the college network. It is suggested that you create a separate directory on your H: drive for the projects in this lab course (ELEC3040 or ELEC3050, for example), to keep your files organized and separate from your other work.

Finally, click **Finish** to bypass the remaining setup screens and begin entering your program.

At this point, *CodeWarrior* creates a project directory and files, including a "dummy" main program to show the basic program structure, as shown in Figure 7. You may delete the data/instructions from this dummy program and replace them with your own, or else delete file *main.asm* in the Sources folder in the project pane on the left side of the window and add your own file (right click on the "Sources" folder, and then navigate to your file.)



Freescale CodeWarrior - [main.asm]
File Edit View Search Project Processor Expert Device Initialization Window Help
Lab1.mcp
TBDML
Files | Link Order | Targets
File | Code | Data
Sources
main.asm
Includes
derivative.inc
mc9s12c32.inc
Project Settings
4 files
Path: H:\ELEC3040_3050\Lab1\Sources\main.asm
Line 1 Col 1

```
;**************************************************************
;* This stationery serves as the framework for a              *
;* user application (single file, absolute assembly application) *
;* For a more comprehensive program that                      *
;* demonstrates the more advanced functionality of this       *
;* processor, please see the demonstration applications       *
;* located in the examples subdirectory of the                *
;* Freescale CodeWarrior for the HC12 Program directory       *
;**************************************************************

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


; Include derivative-specific definitions
    INCLUDE 'derivative.inc'

ROMStart    EQU  $4000  ; absolute address to place my code/constant data

; variable/data section

            ORG RAMStart
 ; Insert here your data definition.
Counter     DS.W 1
FiboRes     DS.W 1


; code section
            ORG   ROMStart


Entry:
_Startup:
            LDS   #RAMEnd+1       ; initialize the stack pointer

            CLI                     ; enable interrupts
mainLoop:
            LDX   #1              ; X contains counter
couterLoop:
            STX   Counter         ; update global.
```

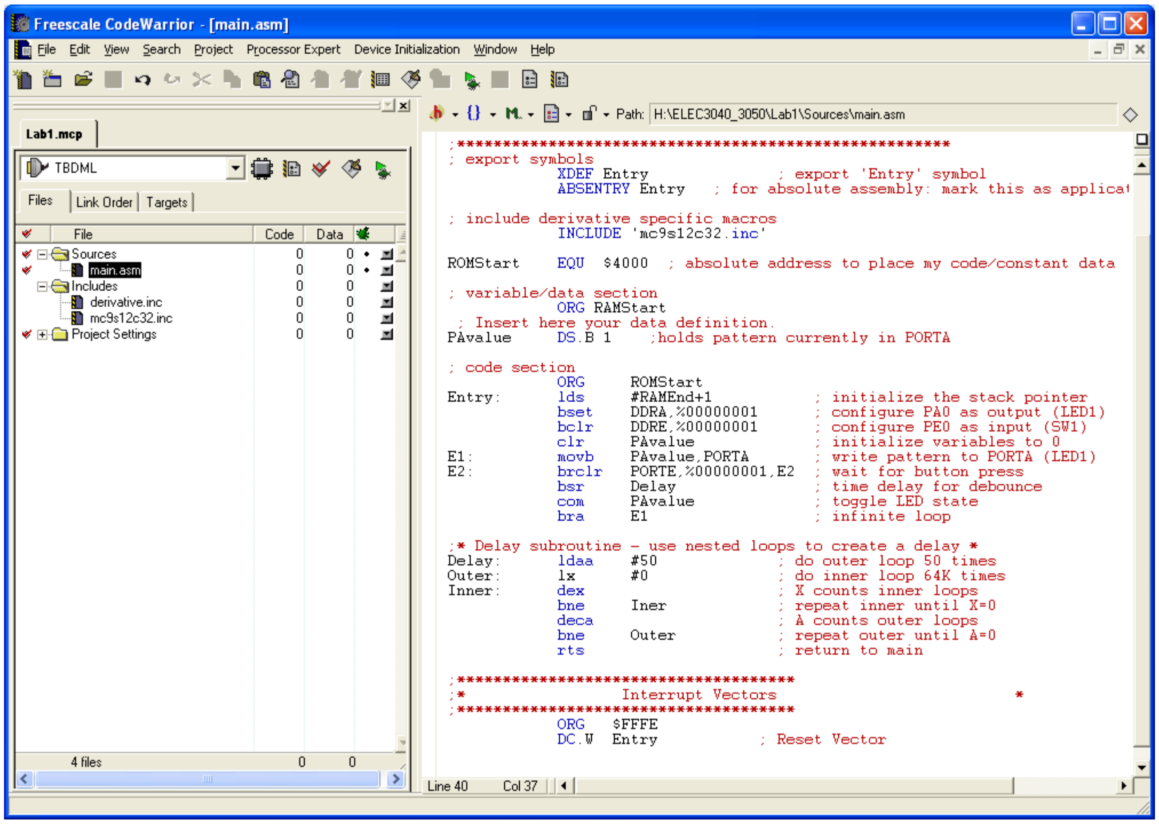
Figure 7. Main project window with initial "dummy" main program.

In the project pane, the two directories of interest are "Sources", which contains your assembly language and/or C files, and "Includes", where you will find, in the default setup, two files. File "derivative.inc" reads in (includes) the specific include file for the selected derivative (mc9s12c32.inc). The latter defines on-chip memory address ranges and symbolic names for all microcontroller register addresses and other resources. Note the INCLUDE directive in the dummy program that includes “derivative.inc” in the main program, thereby making all the symbolic names available for use. If you wish, you can change “derivative.inc” to “mc9s12c32.inc” to save a step.

Your own comment header, data definitions and code should replace those of the dummy program. For this exercise, replace them with the test program provided on the next page. Every program in this course should have a “comment header” at the top, listing the name of the program, the authors, and a very brief description of what the program does. You should also use comments throughout the program for documentation, helping readers to follow what is being done by the program. Such comments should be only a few words each, so they do not carry over to multiple lines (making the program instructions difficult to follow.)

Now, assemble the program by clicking the “Make” button in the project pane (Figure 8), or select **Project > Make** from the menu bar. Correct any syntax errors reported above the program pane, and reassemble the program until it is free of syntax errors.

NOTE: The given program intentionally has two syntax errors that will be detected by the assembler, and will need to be corrected.


Freescale CodeWarrior - [main.asm]
File Edit View Search Project Processor Expert Device Initialization Window Help
Lab1.mcp
TBDML
Files | Link Order | Targets
File | Code | Data
Sources
main.asm
Includes
derivative.inc
mc9s12c32.inc
Project Settings
4 files
Path: H:\ELEC3040_3050\Lab1\Sources\main.asm
Line 40 Col 37

```
;**********************************************************
; export symbols
            XDEF Entry            ; export 'Entry' symbol
            ABSENTRY Entry   ; for absolute assembly: mark this as applicat

; include derivative specific macros
            INCLUDE 'mc9s12c32.inc'

ROMStart    EQU  $4000  ; absolute address to place my code/constant data

; variable/data section
            ORG RAMStart
 ; Insert here your data definition.
PAvalue     DS.B 1    ;holds pattern currently in PORTA

; code section
            ORG     ROMStart
Entry:      lds     #RAMEnd+1           ; initialize the stack pointer
            bset    DDRA,%00000001      ; configure PA0 as output (LED1)
            bclr    DDRE,%00000001      ; configure PE0 as input (SW1)
            clr     PAvalue             ; initialize variables to 0
E1:         movb    PAvalue,PORTA       ; write pattern to PORTA (LED1)
E2:         brclr   PORTE,%00000001,E2  ; wait for button press
            bsr     Delay               ; time delay for debounce
            com     PAvalue             ; toggle LED state
            bra     E1                  ; infinite loop

;* Delay subroutine - use nested loops to create a delay *
Delay:      ldaa    #50             ; do outer loop 50 times
Outer:      lx      #0              ; do inner loop 64K times
Inner:      dex                     ; X counts inner loops
            bne     Iner            ; repeat inner until X=0
            deca                    ; A counts outer loops
            bne     Outer           ; repeat outer until A=0
            rts                     ; return to main

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

Figure 8. Click “Make” to assemble the file.

```
;***************************************************
;* ELEC 3050 - Lab 1, Program 1                    *
;*   Exercise the HCS12 with assembly language     *
;*   Project board LED1 is connected to PORTA, bit 0  *
;*             SW1  is connected to PORTE, bit 0      *
;***************************************************
; export symbols
        XDEF Entry           ; export 'Entry' symbol
        ABSENTRY Entry   ; for absolute assembly: mark this as application entry point

; include derivative specific macros
        INCLUDE 'MC9S12C32.inc'

ROMStart    EQU  $4000  ; absolute address to place my code/constant data

; variable/data section
        ORG RAMStart
 ; Insert here your data definition.
PAvalue     DS.B 1              ;holds pattern currently in PORTA

; code section
        ORG     ROMStart
Entry:  LDS     #RAMEnd+1              ; initialize the stack pointer
        bset    DDRA,%00000001      ; configure PA0 as output (LED1)
        bclr    DDRE,%00000001      ; configure PE0 as input (SW1)
        clr     PAvalue                 ; initialize variables to 0
E1:     movb  PAvalue,PORTA          ; write pattern to PORTA (LED1)
E2:     brclr   PORTE,%00000001,E2  ; wait for button press
        bsr     Delay                     ; time delay for debounce
        com     PAvalue                   ; toggle LED state
        bra     E1                        ; infinite loop

;* Delay subroutine - use nested loops to create a delay *
Delay:    ldaa    #50        ; do outer loop 50 times
Outer:     lx     #0         ; do inner loop 64K times
Inner:    dex                ; X counts inner loops
          bne     Iner       ; repeat inner until X=0
          deca                ; A counts outer loops
          bne    Outer       ; repeat outer until A=0
          rts                 ; return to main

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

When the program is ready to test, the following test and debug procedure should be performed on the target hardware.

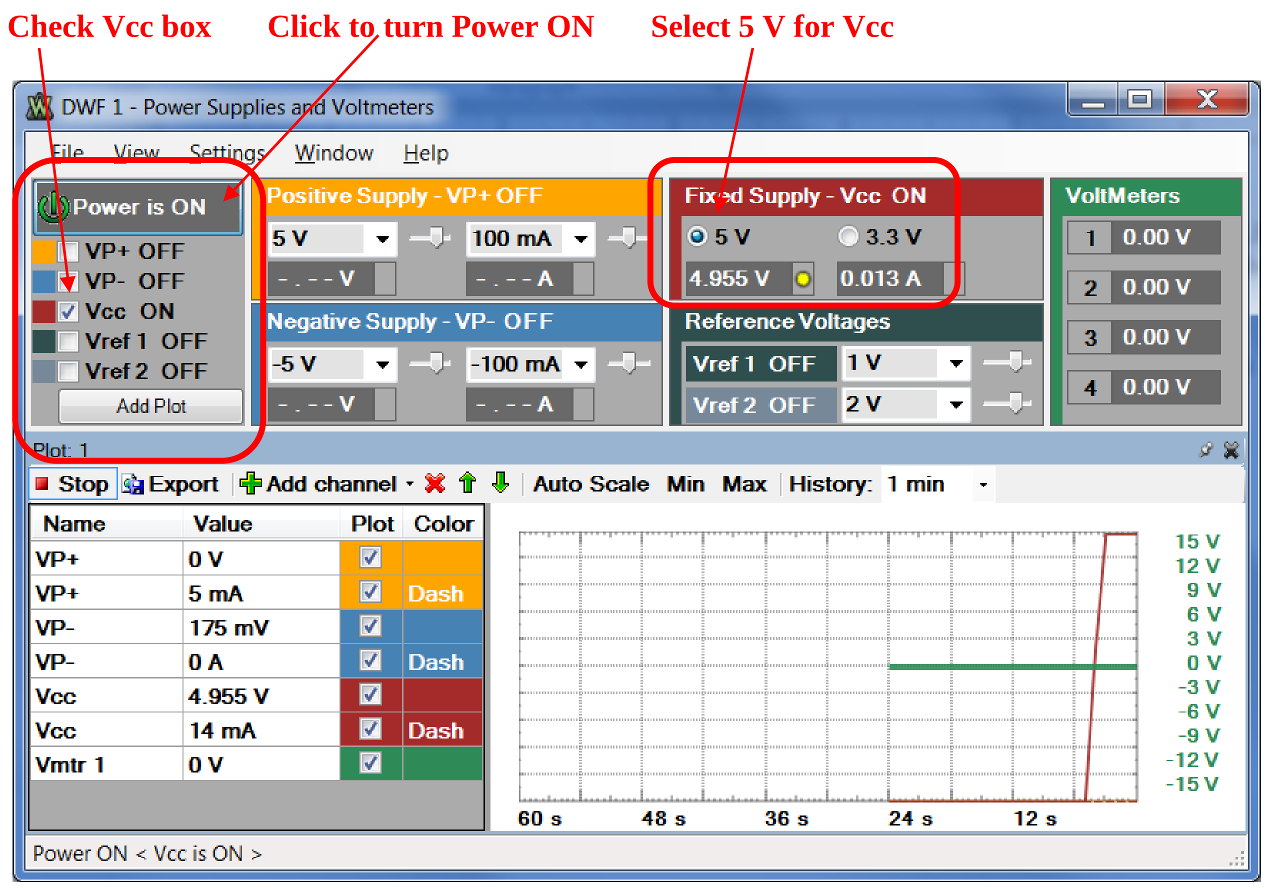
Before starting a debug session on the target hardware, the DragonFly module must be inserted into one of the EEBOARD breadboards.

- Connect the EEBOARD Vcc power supply output (in the POWER section in the lower right corner of the EEBOARD) to pin 2 (Vcc) of the DragonFly module,
- Connect ground (holes marked by the arrow above and dash below them, next to the Vcc holes in the POWER section of the EEBOARD) to pin 1 (GND) of the DragonFly.
- **Double-check your power and ground connections, since improper connections will likely damage the module.**
- Connect DragonFly pin 22 (HCS12 I/O port pin PA0) to pin 0 of the "Digital 1" section at the top of the EEBOARD (this will be configured as an LED in *WaveForms*),
- Connect DragonFly pin 13 (HCS12 I/O port pin PE0) to pin 1 of the "Digital 1" section (this will be configured as a switch in *WaveForms*).
- Connect the BDM cable to the DragonFly module via the 6-pin header. When connecting the BDM module, please pay particular attention to the orientation of the ribbon cable (shown in Figure 4) on the 6-pin BDM header (shown in Figure 3(a)-(b)). ***The red conductor on the BDM ribbon cable MUST be aligned with pin 1 of the BDM header when connecting the BDM module, or damage to the microcontroller and/or BDM module can occur.***

NOTE: NEVER INSERT OR REMOVE WIRES ON THE DRAGONFLY MODULE WHILE POWER IS ON. This can cause voltage transients capable of damaging the module and/or the breadboard station electronics.

After checking module connections one more time, follow the following procedure to power on the EEBOARD, set up I/O devices/instruments, and run/debug the program.

1. Connect the EEBOARD to the PC with the provided USB micro AB cable.
2. Connect the EEBOARD power supply.
3. Turn the EEBOARD On-Off switch to ON.
4. Launch the *WaveForms* software on the PC.
5. Launch the "Power Supply and Voltmeter" instrument in *WaveForms*.
6. Check the Vcc power supply box and uncheck all the other power supplies and reference voltages (see Figure 9).
7. Set Vcc to 5V (see Figure 9).
8. Turn the power ON (see Figure 9).
9. Launch the Static I/O instrument in *WaveForms*.
10. By right-clicking on each (see Figure 10), configure Digital I/O pin DIO0 as an LED and DIO1 as a push button. If you wish, you may remove DIO31-24, DIO23-16, and DIO15-8 via the "Config" buttons at the left.



DWF 1 - Power Supplies and Voltmeters

File View Settings Window Help

Power is ON

🟧 ☐	VP+ OFF
🟦 ☐	VP- OFF
🟥 ☑	Vcc ON
⬛ ☐	Vref 1 OFF
⬜ ☐	Vref 2 OFF

Add Plot

Positive Supply - VP+ OFF: 5 V, 100 mA; - . - - V, - . - - A

Negative Supply - VP- OFF: -5 V, -100 mA; - . - - V, - . - - A

Fixed Supply - Vcc ON: [✓ 5 V] 3.3 V; 4.955 V, 0.013 A

Reference Voltages: Vref 1 OFF 1 V; Vref 2 OFF 2 V

VoltMeters	
1	0.00 V
2	0.00 V
3	0.00 V
4	0.00 V

Plot: 1

Stop Export Add channel Auto Scale Min Max History: 1 min

Name	Value	Plot	Color
VP+	0 V	☑	🟧
VP+	5 mA	☑	Dash
VP-	175 mV	☑	🟦
VP-	0 A	☑	Dash
Vcc	4.955 V	☑	🟥
Vcc	14 mA	☑	Dash
Vmtr 1	0 V	☑	🟩

Power ON < Vcc is ON >

Figure 9. WaveForms Power Supply and Voltmeter Instrument

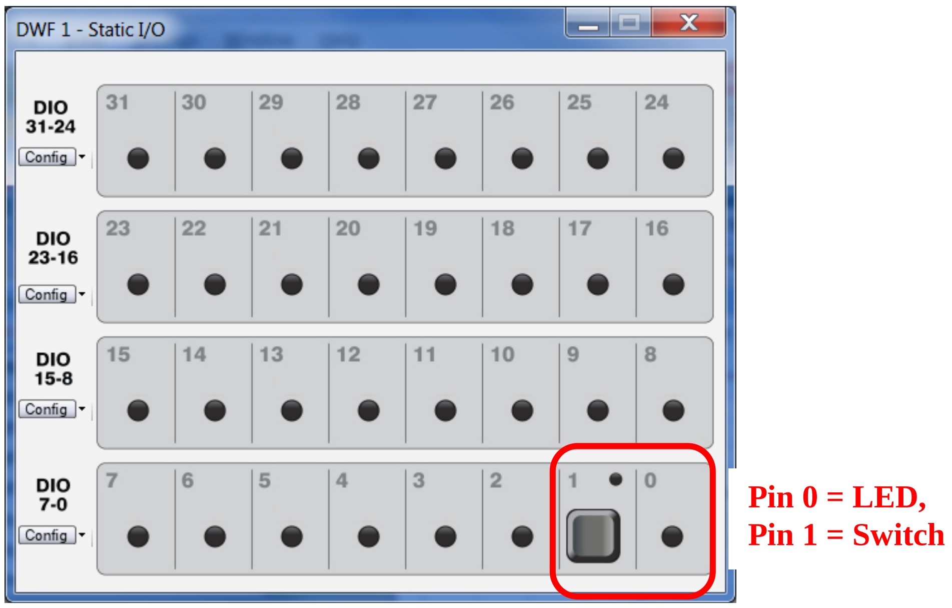
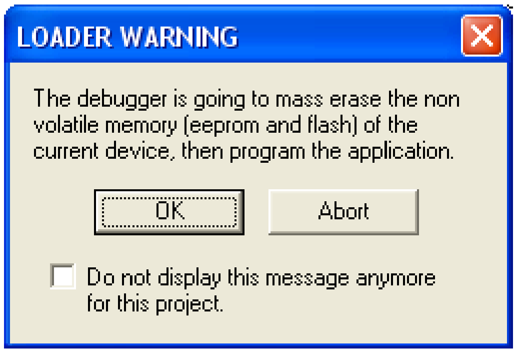


Figure 10. WaveForms Static I/O Instrument

After the EEBOARD has been powered and configured, return to the main *CodeWarrior* window and click on the "Debug" button (green arrow button to the left of the "Make" button) to invoke the debugger. Note that **P&E USB BDM Multilink** should be selected in the project pane, and not "Full Chip Simulation". You will receive a warning (Figure 11) that the debugger is going to erase the on-chip flash memory and reprogram it with your application program via the BDM cable. Click OK in this window, or abort if you wish to retain the program currently in the microcontroller. Once programming is finished, you may test the program with exactly the same interface and procedure as used in the full chip simulation in ELEC 2220, except that you will see the actual hardware operate, i.e. LED1 and switch SW1 in the WaveForms Static I/O window, controlled by microcontroller pins PA0 and PE0, respectively.


LOADER WARNING

The debugger is going to mass erase the non volatile memory (eeprom and flash) of the current device, then program the application.

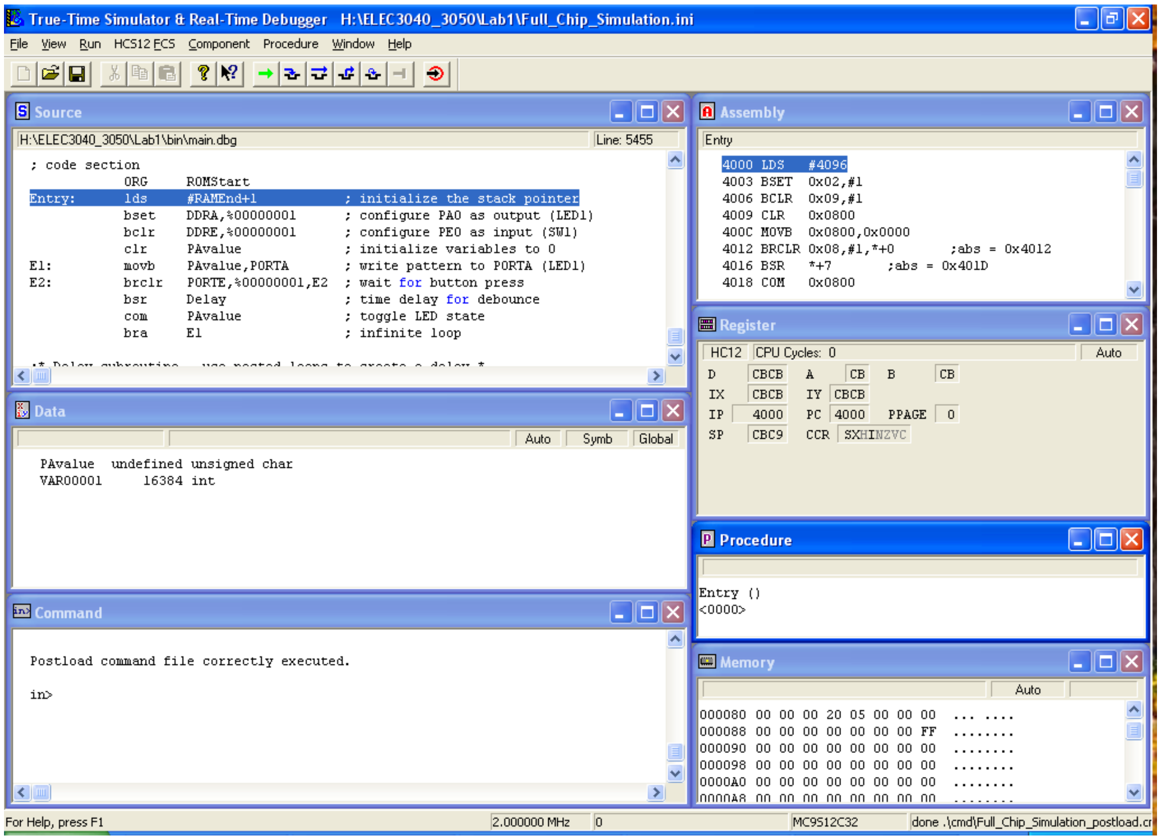
Figure 11. Program code written to on-chip flash memory via the BDM.

The debug window (Figure 12), which is the same whether debugging in the target hardware or the simulator, shows the source program and the corresponding disassembled code in the two top panes, with the next instruction to be executed highlighted. Current variable and register values are displayed in the two middle windows, and memory contents in the bottom right window. In the memory window, you may display your current data values in RAM by right clicking in the window, selecting "Address", and entering the RAM starting address (0800).

The program may be executed one instruction at a time, executed up to a breakpoint, or simply executed without halting via the buttons below the menu bar. From left to right, the button functions are as follows.

- Start/Continue – execute until a stop condition (breakpoint) is reached
- Single Step – execute a single instruction (C or assembly)
- Step Into – execute an entire subroutine/function as a single instruction
- Step Out – executed until the current subroutine is exited
- Assembly Step – for C programs, execute one assembly language instruction
- Stop – stop executing
- Reset – reset the microcontroller

These commands, plus the ability to set breakpoints in the source window, are also available in a popup menu accessed with a right mouse click.


True-Time Simulator & Real-Time Debugger H:\ELEC3040_3050\Lab1\Full_Chip_Simulation.ini
File View Run HCS12 FCS Component Procedure Window Help
Source
H:\ELEC3040_3050\Lab1\bin\main.dbg
Line: 5455
Assembly
Entry
Register
Data
Auto
Symb
Global
Command
Procedure
Memory
For Help, press F1
2.000000 MHz
MC9S12C32
done .\cmd\Full_Chip_Simulation_postload.cr

```
; code section
            ORG     ROMStart
Entry:      lds     #RAMEnd+1            ; initialize the stack pointer
            bset    DDRA,%00000001       ; configure PA0 as output (LED1)
            bclr    DDRE,%00000001       ; configure PE0 as input (SW1)
            clr     PAvalue              ; initialize variables to 0
E1:         movb    PAvalue,PORTA        ; write pattern to PORTA (LED1)
E2:         brclr   PORTE,%00000001,E2   ; wait for button press
            bsr     Delay                ; time delay for debounce
            com     PAvalue              ; toggle LED state
            bra     E1                   ; infinite loop
```

```
4000 LDS   #4096
4003 BSET  0x02,#1
4006 BCLR  0x09,#1
4009 CLR   0x0800
400C MOVB  0x0800,0x0000
4012 BRCLR 0x08,#1,*+0       ;abs = 0x4012
4016 BSR   *+7       ;abs = 0x401D
4018 COM   0x0800
```

```
HC12   CPU Cycles: 0                    Auto
D    CBCB    A    CB     B      CB
IX   CBCB    IY   CBCB
IP   4000    PC   4000   PPAGE  0
SP   CBC9    CCR  SXHINZVC
```

```
PAvalue  undefined unsigned char
VAR00001     16384 int
```

```
Postload command file correctly executed.

in>
```

```
Entry ()
<0000>
```

```
000080  00 00 00 20 05 00 00 00  ... ....
000088  00 00 00 00 00 00 00 FF  ........
000090  00 00 00 00 00 00 00 00  ........
000098  00 00 00 00 00 00 00 00  ........
0000A0  00 00 00 00 00 00 00 00  ........
0000A8  00 00 00 00 00 00 00 00  ........
```

Figure 12. Debug window, showing source and disassembled code, register values, memory, data values, and a command window, plus two simulated IO_Led components.

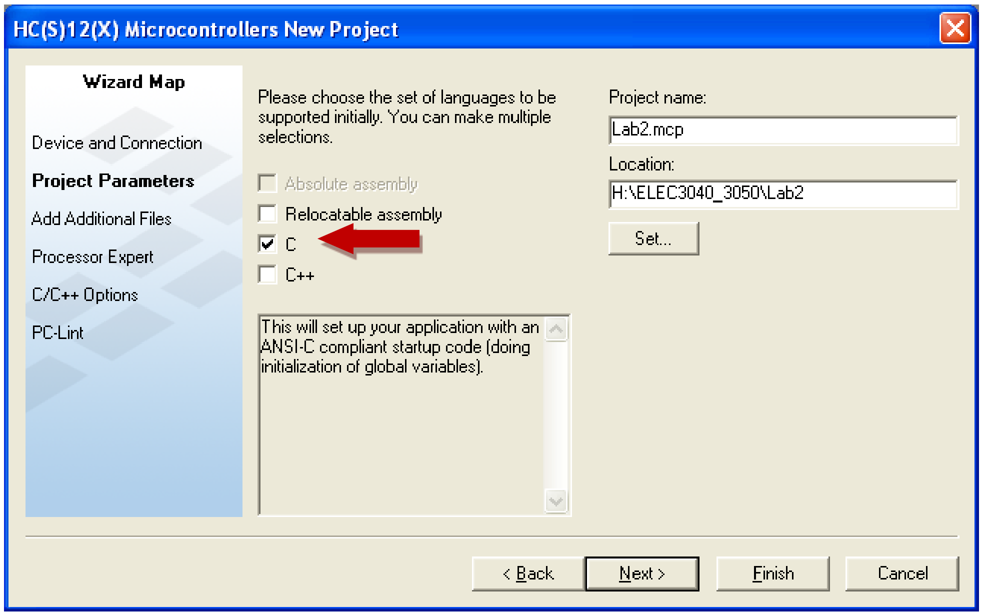
You might wish to use the simulator on your own PC or in one of the other labs, as was done in ELEC 2220. Since the program controls an LED and tests a switch, simulated components can be used to test the program. **However, the simulated components (LEDs, switches, LCD, etc.) were discontinued in *CodeWarrior* after Version 4.6.** You can still use these in Version 5 and later versions, but the component models must be copied to the *CodeWarrior* installation directory as instructed on the course web site. (A zip file containing these component models is provided on the web site.)

To get a simulated LED, from the debug window menu bar select **Components > Open** to get a list of simulated devices, and select an Io_Led component. Right click on the component and enter the port address (0 for port A) and data direction register address (2 for Port A). An Io_Led component can also simulate the switch. repeat the above for port E (address 8 for data register , 9 for data direction register.) When used as a system input, the LED reflects the current value, and clicking on it changes the value. Once you have things set up, from the menu bar select File > Save Configuration. This will restore this configuration the next time you enter the debugger.

Program 2 – Example C program

The C program on the next page performs the same function as Program 1. Create a new project for this program, and verify that it produces the same behavior on the DragonFly12-Plus module. The project creation procedure is identical to the above, with the following differences.

- On the Project Parameters form (Figure 13), select C instead of assembly.
- Click "Finish" to create the project with the following default settings:
    - No additional project files added
    - Processor Expert not used
    - ANSI startup code added
    - Small memory model used (we will use the minimum amount of memory)
    - No floating point arithmetic
    - PC-lint not used
- If you click "Next" instead of "Finish", the setup wizard will guide you through selection of the above settings.


HC(S)12(X) Microcontrollers New Project

Wizard Map: Device and Connection; **Project Parameters**; Add Additional Files; Processor Expert; C/C++ Options; PC-Lint

Please choose the set of languages to be supported initially. You can make multiple selections.

- [ ] Absolute assembly
- [ ] Relocatable assembly
- [x] C
- [ ] C++

This will set up your application with an ANSI-C compliant startup code (doing initialization of global variables).

Project name: Lab2.mcp

Location: H:\ELEC3040_3050\Lab2

Figure 13. Select C instead of Absolute assembly.

As in the case of an assembly language project, a "dummy" C program is created as file main.c, listed under Sources in the project pane. Again, you may either replace main.c with your own file, or edit the dummy program to enter your own C code.

The compilation and debug procedure is identical to that used for assembly language programs, as described above.

```
/*------------------------------------*/
/* Victor P. Nelson                   */
/* ELEC 3040/3050 - Lab 1, Program 2  */
/* Toggle LED1 while SW1 pressed,     */
/*   with short delay inserted        */
/*------------------------------------*/

#include <hidef.h>          /* common defines and macros */
#include <MC9S12C32.h>      /* derivative information */
/* Defines PORTA,DDRA: LED1 connected to PA0 */
/* Defines PORTE,DDRE: SW1  connected to PE0 */

/* Delay function - do nothing for about 1 second */
void delay () {
  int i,j;
  for (i=0; i<100; i++) {       //outer loop
    for (j=0; j<20000; j++) {   //inner loop
    }                           //do nothing
  }
}

void main(void) {
  char sw1;                 //state of SW1
  char led1;                //state of LED1

  /* Set port directions */
  DDRA = 1;                 //PORTA PA0 = output to LED1
  DDRE = 0;                 //PORTE PE0 = input from SW1

  /* Initial LED state */
  led1 = 0;                 //state of LED1

  /* Endless loop */
  while (1) {                //Can also use:  for(;;) {
    PORTA = led1;            //Display LED1 state via PA0
    sw1 = PORTE & 0x01;      //Read PORTE and isolate bit 0

    /* Wait until SW1 pressed */
    while (sw1 == 0) {       //Wait for SW1 = 1 on PE0
      sw1 = PORTE & 0x01;    //Read PORTE and isolate bit 0
    }
    delay();                //Time delay for button release
    led1 = ~led1;           //Complement LED1 state
  } /* repeat forever */

}
```

Laboratory Status Report and Lab Notebook

For Lab 1, each student is to submit a separate memo-style laboratory report, as described in the syllabus. The memo should (1) begin with a brief statement of the objectives of the lab, (2) provide a short discussion of what was done and the results that were observed, and (3) conclude with a brief statement of what was learned, and any suggestions for the lab instructors. In future labs, when programs are written and/or circuits designed, these should be attached to the memo, or otherwise included.

In addition, each student should enter all pre-lab and in-lab activity in the lab notebook. Lab notebooks will be collected from all students every two weeks, beginning with week 2.