Getting Started with the Cortex-M3® and StellarisWare® Workshop

Student Guide and Lab Manual

Revision 2.01
January 2012

Technical Training Organization

Important Notice

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Revision History

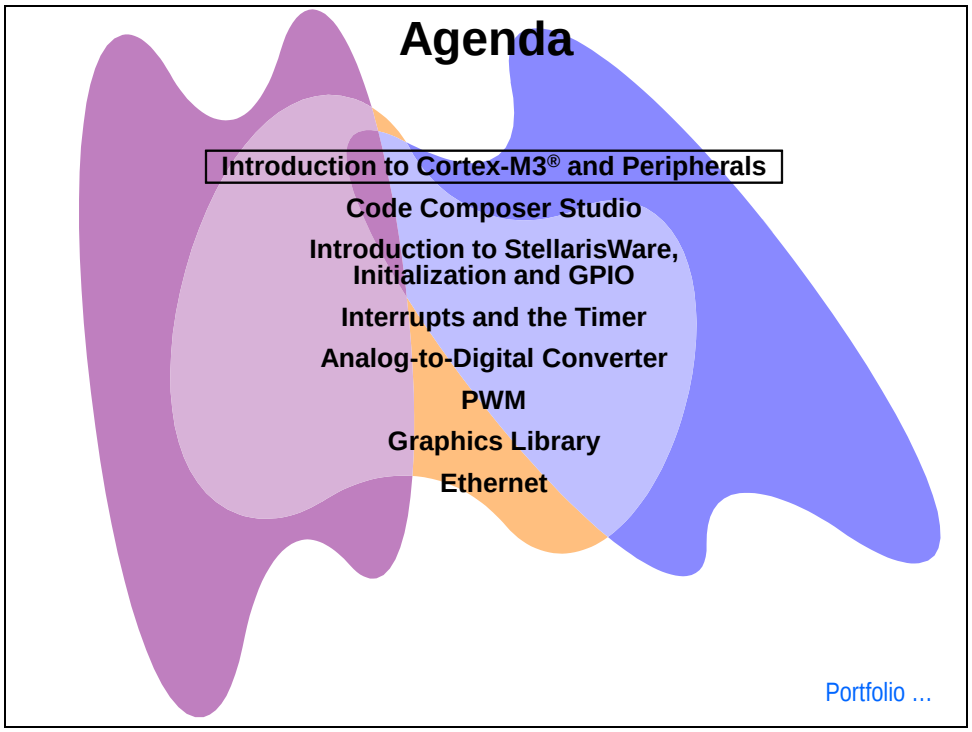
May 2011 – Revision 1.0
June 2011 – Revision 1.1 (errata, 8962 features slide addition)
July 2011 – Revision 1.2 (errata)
January 2012 – Revision 2.0 (update to CCS 5.1, removed low-power M3 material)
January 2012 – Revision 2.01 (minor errata)

Mailing Address

Texas Instruments
Training Technical Organization
6550 Chase Oaks Blvd
Building 2
Plano, TX 75023

Introduction

This module will introduce you to the basics of the Cortex-M3 core and the Stellaris peripherals. The lab will step you through setting up the hardware and software required for the rest of the labs.

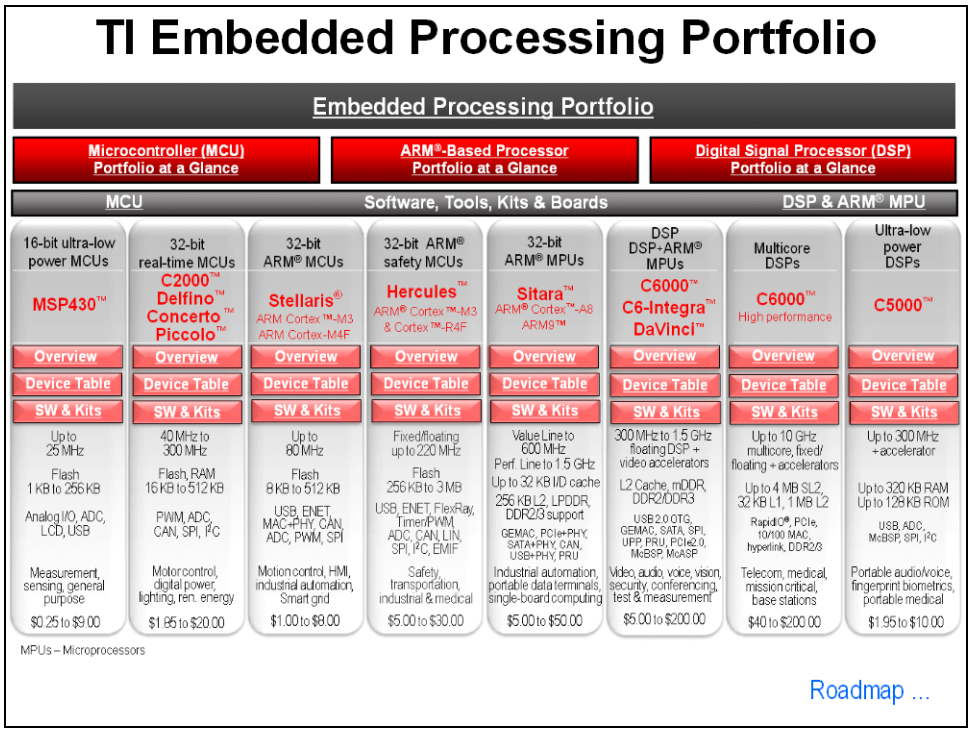


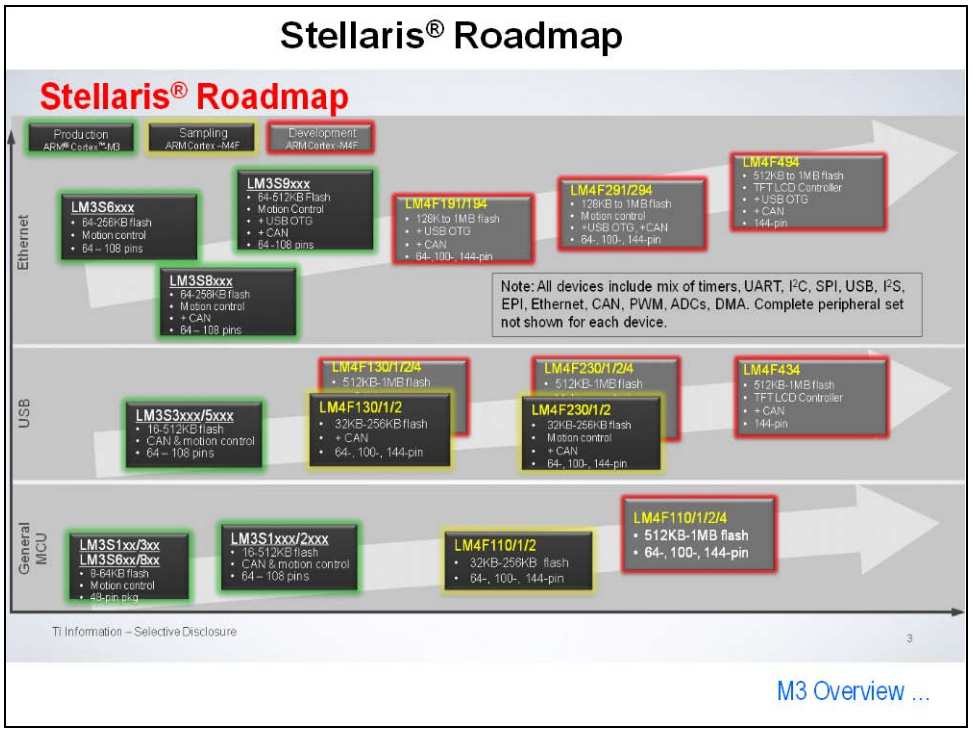
The Wiki page for this workshop is located here:

http://processors.wiki.ti.com/index.php/Getting_Started_with_StellarisWare_Workshop

Module Topics

TI Processor Portfolio and Stellaris Roadmap

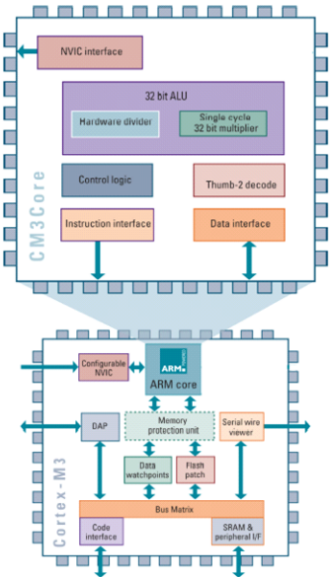




Overview and Benefits

Cortex-M3® Processor Overview

- Harvard architecture
- 3-stage pipeline with branch speculation
- Thumb®-2 instruction set
- ALU with H/W divide and single cycle multiply
- Configurable Nested-Vectored Interrupt Controller (NVIC)
- Memory Protection Unit (MPU)
- Advanced debug components

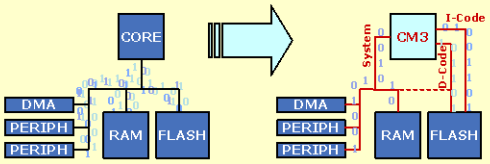


Benefits...

Cortex-M3® Benefits

Harvard architecture

- Separate buses for instructions and data speeds application execution



Thumb-2 Instruction Set Architecture

- Blend of 16 and 32-bit instructions that delivers significant benefits in code size and performance

3-stage pipeline + branch speculation

- The decode stage generates a speculative instruction fetch for each branch instruction encountered,

Memory Map...

Internal Memory

Stellaris Internal Memories

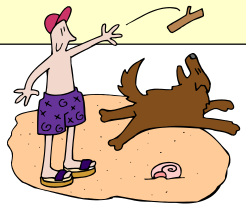
- **FLASH**
  - Up to 256 KB
  - **Single cycle up to 50MHz**
  - 1KB sector
  - Flash memory protection
  - Flash programming
- **SRAM**
  - Up to 96KB
  - Up to 80MHz
  - Bit-band capability
  - Code execution
- **ROM**
  - Stellaris® Boot Loader
  - Stellaris® Peripheral Driver Library
  - AES cryptography tables
  - CRC error detection functionality

0x0000 0000 Internal FLASH
0x0100 0000 On-chip ROM
0x2000 0000 Bit-banded on-chip SRAM
0x2200 0000 Bit-band alias of 0x2000 0000
0x4000 0000 Peripherals
0x4200 0000 Bit-band alias of 0x4000 0000
0x6000 0000 EPI0 mapped peripheral and RAM
0xE000 0000 Private peripherals bus Instrumentation Trace Macrocell Data Watchpoint and Trace Flash Patch and Breakpoint Nested Vectored Interrupt Controller Trace Port Interface Unit

Code Fetch …

Code Fetch Process

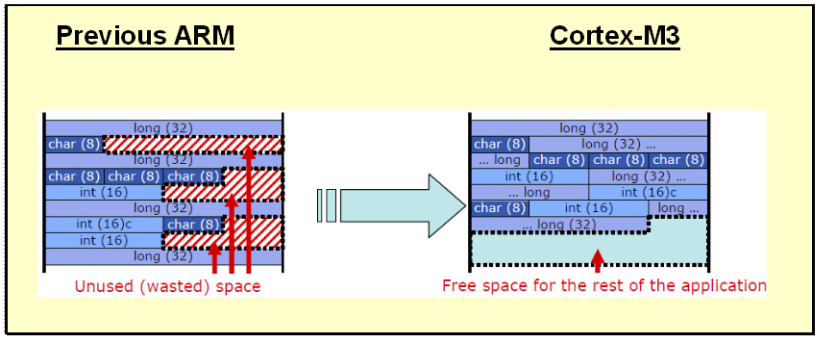
The Flash memory controller prefetch buffer is automatically used when the CPU frequency > 50 MHz.
In this mode, Flash memory operates at half of the system clock. The prefetch buffer fetches two 32-bit words per clock allowing instructions to be fetched with no wait states while code is executing linearly. The fetch buffer includes a branch speculation mechanism that recognizes a branch and avoids extra wait states by not reading the next word pair. Short loop branches often stay in the buffer. As a result, some branches can be executed with no wait states. Other branches incur a single wait state.



Memory Alignment…

Memory Alignment

Data memory accesses can be defined as aligned or unaligned improving data constant and RAM utilization



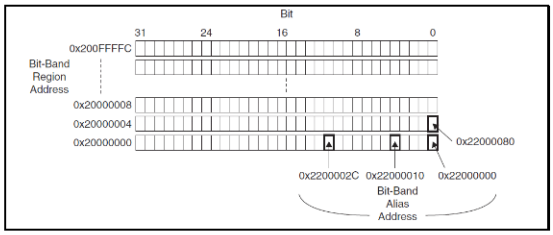
Bit-banding...

Bit-Banding

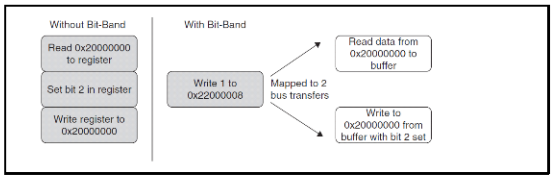
Bit Banding

Bit-banding provides an address alias to individual RAM bits …

- When the bit-band alias address is used, each individual bit can be accessed separately in the least significant bit (LSB) of each word-aligned address



- For example, to set bit 2 in word data in address 0x20000000



MPU…

Memory Protection Unit

Memory Protection Unit

Benefits:

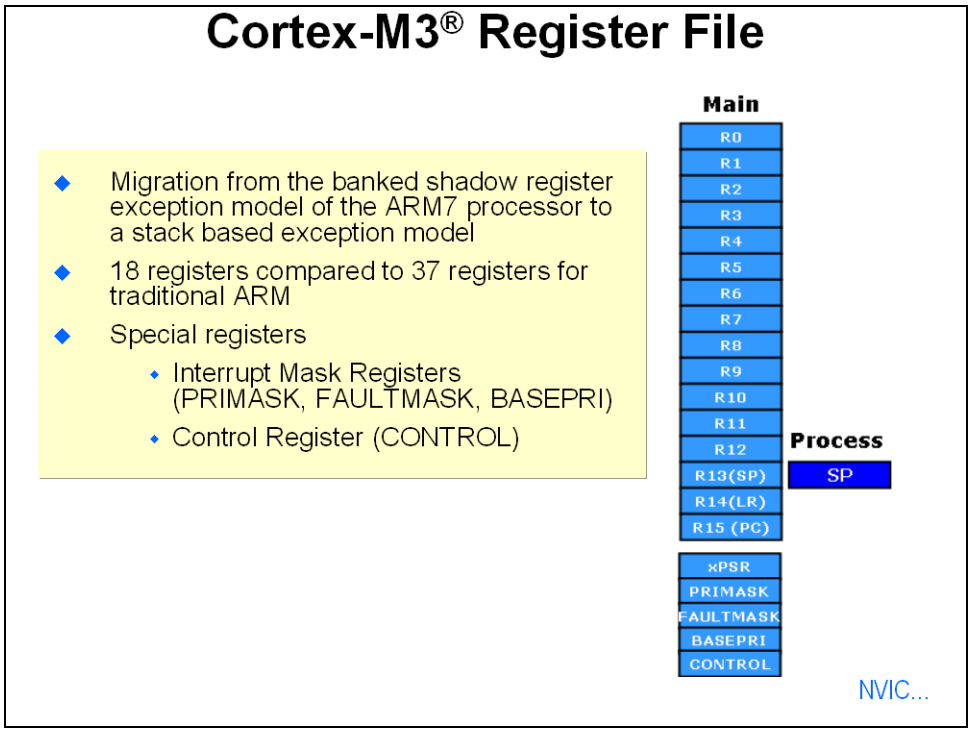
- Enforce privilege rules
- Separate processes
- Enforce access rules

Features

- 8 Protection regions from 32B to 4GB range
- 8 sub-regions within each region
- Every region and sub-region permits:
  - Code execution - allow/disallow
  - Data access – none, read-only, read/write access
- Any violation causes a memory management fault, activating the fault handler

Register File ...

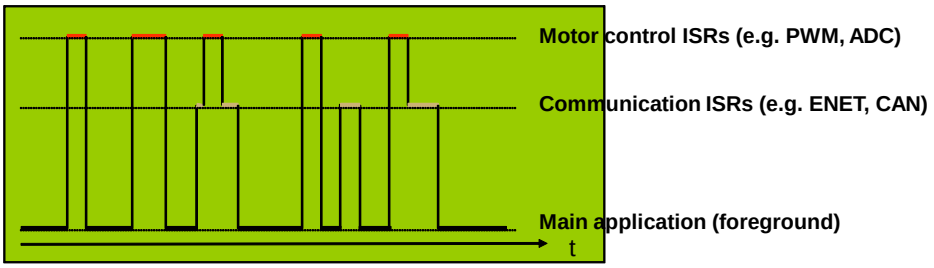
Cortex-M3® Register File



NVIC

Nested Vectored Interrupt Controller (NVIC)

- No need to use software to determine which IRQ handler to serve. There is also no need to have software code to set up nested interrupt support
- Programmable interrupt priority control via a 4-bit interrupt priority value. Pre-empting and non-pre-empting prioritization.
- Reduced interrupt latency thanks to late arrival interrupt acceptance and tail-chain interrupt entry



Interrupt Handling…

Cortex-M3® Interrupt Handling

Interrupt handling is micro-coded. No instruction overhead

- **Entry**
  - Automatically pushes registers R0–R3, R12, LR, PSR, and PC onto the stack
  - In parallel, ISR is pre-fetched on the instruction bus.
    - ISR ready to start executing as soon as stack PUSH complete
  - A late-arriving interrupt will restart ISR pre-fetch, but state saving does not need to be repeated
- **Exit**
  - Processor state is automatically restored from the stack
  - In parallel, interrupted instruction is pre-fetched ready for execution upon completion of stack POP
  - Stack POP can be interrupted, allowing new ISR to be immediately executed without the overhead of state saving

Tail Chaining…

Interrupt Latency

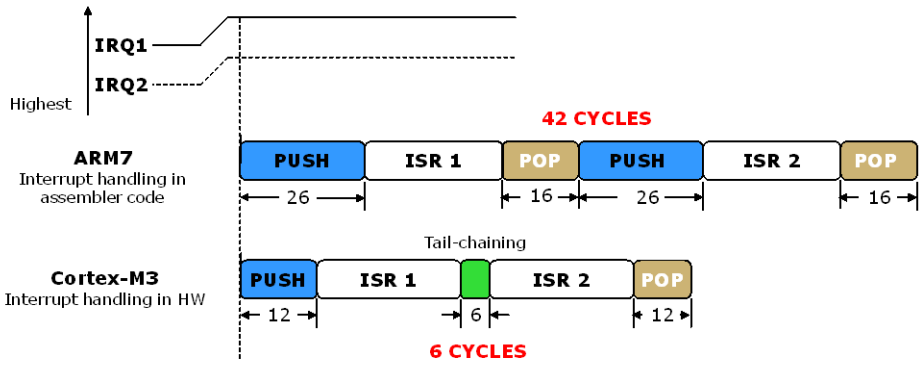
Cortex-M3® Interrupt Handling

Interrupt handling is micro-coded. No instruction overhead

- **Entry**
  - Automatically pushes registers R0–R3, R12, LR, PSR, and PC onto the stack
  - In parallel, ISR is pre-fetched on the instruction bus.
    - ISR ready to start executing as soon as stack PUSH complete
  - A late-arriving interrupt will restart ISR pre-fetch, but state saving does not need to be repeated
- **Exit**
  - Processor state is automatically restored from the stack
  - In parallel, interrupted instruction is pre-fetched ready for execution upon completion of stack POP
  - Stack POP can be interrupted, allowing new ISR to be immediately executed without the overhead of state saving

Tail Chaining...

Interrupt Latency - Tail Chaining



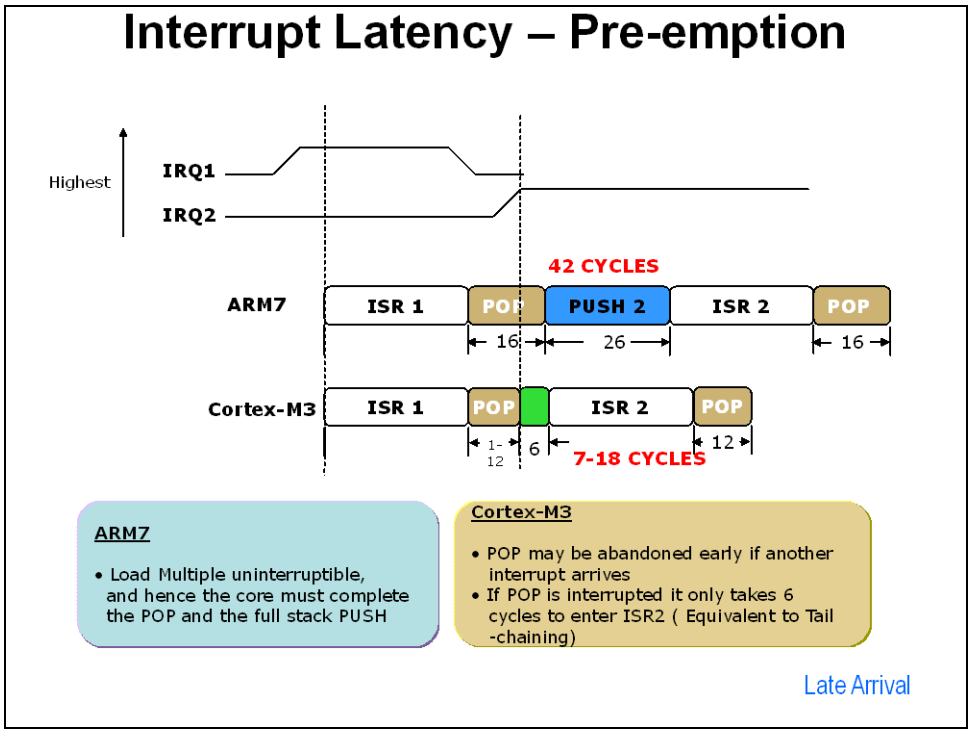
ARM7

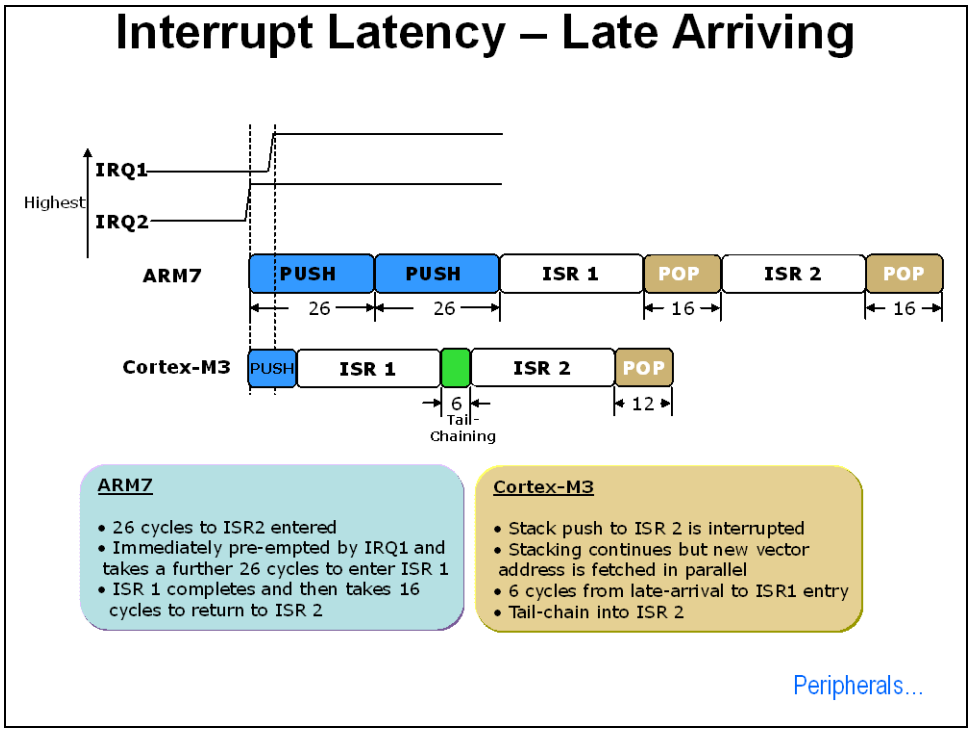
- 26 cycles from IRQ1 to ISR1 entered
  - Up to 42 cycles if LSM
- 42 cycles from ISR1 exit to ISR2 entry
- 16 cycles to return from ISR2

Cortex-M3

- 12 cycles from IRQ1 to ISR1 entered
  - 12 cycles if LSM
- 6 cycles from ISR1 exit to ISR2 entry
- 12 cycles to return from ISR2

Pre-emption...


Interrupt Latency – Pre-emption
Highest
IRQ1
IRQ2
42 CYCLES
ARM7
ISR 1
POP
PUSH 2
ISR 2
POP
16
26
16
Cortex-M3
ISR 1
POP
ISR 2
POP
1-12
6
12
7-18 CYCLES
ARM7
• Load Multiple uninterruptible, and hence the core must complete the POP and the full stack PUSH
Cortex-M3
• POP may be abandoned early if another interrupt arrives
• If POP is interrupted it only takes 6 cycles to enter ISR2 (Equivalent to Tail -chaining)
Late Arrival


Interrupt Latency – Late Arriving
Highest
IRQ1
IRQ2
ARM7
PUSH
PUSH
ISR 1
POP
ISR 2
POP
26
26
16
16
Cortex-M3
PUSH
ISR 1
ISR 2
POP
6
Tail-Chaining
12
ARM7
• 26 cycles to ISR2 entered
• Immediately pre-empted by IRQ1 and takes a further 26 cycles to enter ISR 1
• ISR 1 completes and then takes 16 cycles to return to ISR 2
Cortex-M3
• Stack push to ISR 2 is interrupted
• Stacking continues but new vector address is fetched in parallel
• 6 cycles from late-arrival to ISR1 entry
• Tail-chain into ISR 2
Peripherals…

Peripherals

Stellaris Family Peripherals

ARM® Cortex™-M3 v7-M Processor Core
- Up to 80 MHz
- Up to 100 MIPS (at 80 MHz)

On-chip Memory
- 256 KB Flash; 96 KB SRAM
- ROM loaded with Stellaris DriverLib, BootLoader, AES tables, and CRC

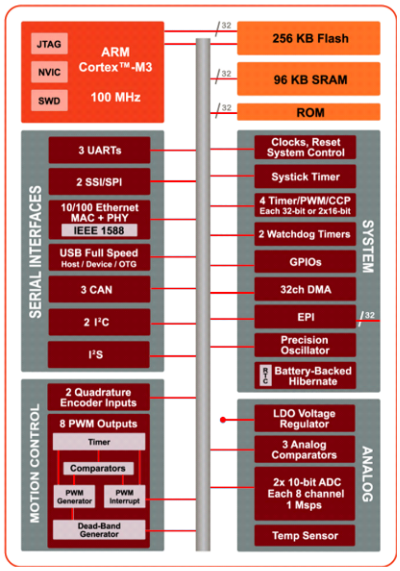
External Peripheral Interface (EPI)
- 32-bit dedicated parallel bus for external peripherals
- Supports SDRAM, SRAM/Flash, M2M

Advanced Serial Integration
- 10/100 Ethernet MAC and PHY
- 3 CAN 2.0 A/B Controllers
- USB (full speed) OTG / Host / Device
- 3 UARTs with IrDA and ISO 7816 support*
- 2 I^2Cs
- 2 Synchronous Serial Interfaces (SSI)
- Integrated Interchip Sound (I^2S)

System Integration
- 32-channel DMA Controller
- Internal Precision 16MHz Oscillator
- Two watchdog timers with separate clock domains
- ARM Cortex Systick Timer
- 4 32-bit timers (up to 8 16-bit) with RTC capability
- Lower-power battery-backed hibernation module
- Flexible pin-muxing capability

Advanced Motion Control
- 8 advanced PWM outputs for motion and energy applications
- 2 Quadrature Encoder Inputs (QEI)

Analog
- 2x 8-ch 10-bit ADC (for a total of 16 channels)
- 3 analog comparators
- On-chip voltage regulator (1.2V internal operation)



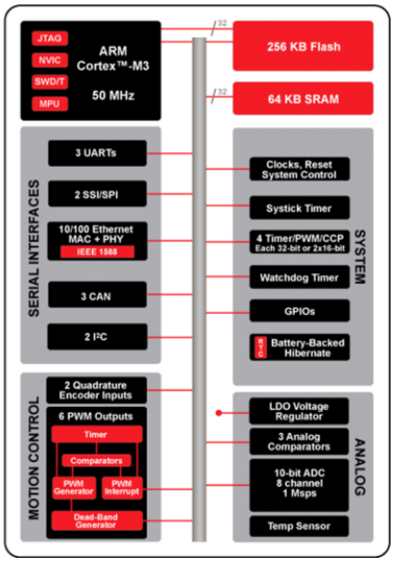
CPU...

* One UART features full modem controls

LM3S8962

LM3S8962 Microcontroller

- 50MHz 32-bit ARM® Cortex™-M3 v7M core
- Thumb2 16/32-bit instruction set
- 256KB Flash / 64KB SRAM
- 5-42 GPIOs
- 4 General Purpose Timer Modules
- ARM FiRM-compliant Watchdog Timer
- 4 input ADC with sequencers and FIFO
- UART/SSI/I²C/CAN
- 10/100 Ethernet MAC/PHY
- Analog Comparator
- 3 PWM Generator Blocks
- 2 Quadrature Encoder Input modules
- On-chip LDO voltage regulator



Eval Kit ...

LM3S8962 Ethernet+CAN Evaluation Kit

- LM3S8962 w/ Integrated 10/100 Ethernet controller and CAN MAC
- OLED graphics display
- Speaker
- microSD card slot
- USB Debugger interface
- Maze Game QuickStart application

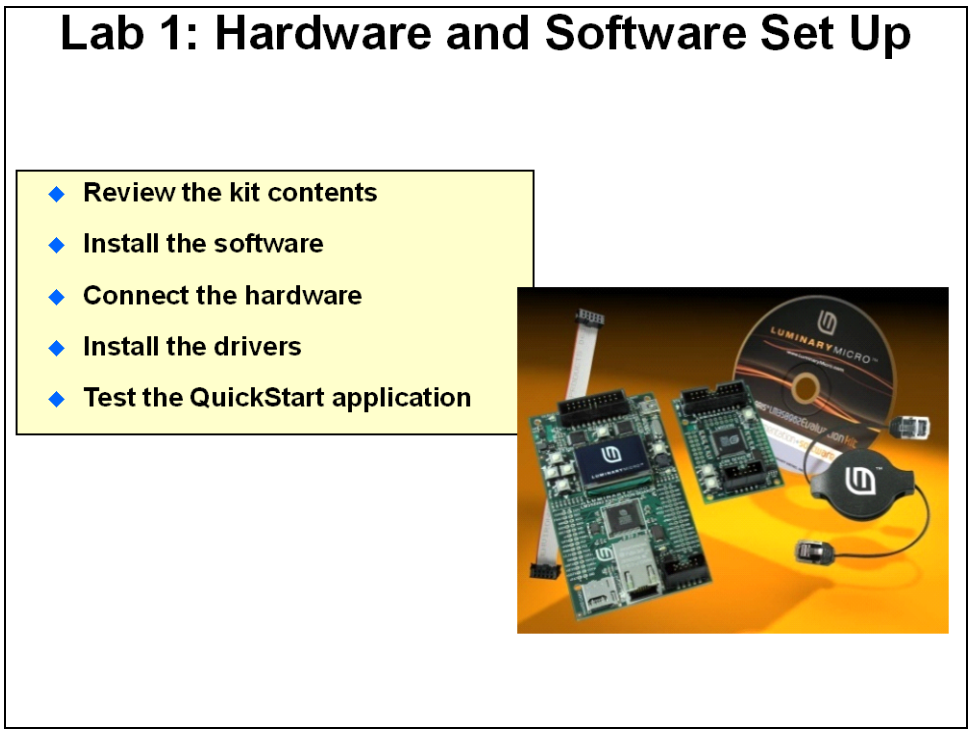
Lab1 ...

Lab1: Hardware and Software Set Up

Objective

The objective of this lab exercise is to download and install Code Composer Studio, as well as download the various other support documents and software to be used with this workshop. Then we will review the contents of the evaluation kit and verify its operation with the pre-loaded demo program. These development tools will be used throughout the remaining lab exercises in this workshop.

The instructions in this workshop use the 8962 evaluation kit, but, with some adjustments, you should be able to use almost any other Stellaris evaluation board in most of the labs.



Procedure

Install the Software

1. Download and run the latest version of Code Composer Studio 5.x web installer from http://processors.wiki.ti.com/index.php/Download_CCS . This version of the workshop was built using build number 5.1.0.09000. For this and the next two steps, you will need a my.TI account. You should note that the free, code size limited version of CCS will not work with Stellaris devices. The full version of CCS will operate with full functionality for free while connected to a Stellaris evaluation board. Those boards can also operate as an emulator interface for your target system, though this function requires a licensed version of CCS.

   When you reach the "Processor Support" page of the installation process, make sure the select "Stellaris Cortex M MCUs" from the list. If you are also attending the MSP430 workshop, select "MSP430 Low Power MCU's" as well. Select others also if you like, but bear in mind that the installation size and time will go up as you select more processors. Use the default selections for all other pages of the setup.

2. Download and install the latest full version of StellarisWare (SW-LM3S) from: http://focus.ti.com/mcu/docs/mcuorphantoolsw.tsp?sectionId=632&orphantabId=8 This workshop was built using release number 8264. Please install StellarisWare into the default "C:/StellarisWare" folder.

3. Download and install the latest LM Flash Programmer (LMFLASHPROGRAMMER) from http://focus.ti.com/mcu/docs/mcuorphantoolsw.tsp?sectionId=632&orphantabId=8. This workshop was built using version number 1368.

4. To properly display the web pages served up by the QuickStart application you should also have the latest version of the Java Runtime Engine installed on your PC. You can find it at www.java.com

5. In section 7, "Graphic Library" we'll need a photo editing tool capable of working with PNM format files. GIMP can do this. Download and install GIMP from www.gimp.org

6. Windows7 no longer has HyperTerminal. To regain that capability, download and "install" TeraTerm or another terminal program of your choice.

7. Download a copy of the workbook pdf file from the wiki site. It will be handy to cut and paste code snippets from.

Helpful Documents

8. There are many helpful documents that you should download, but at a minimum you should have the following documents at your fingertips.

   Go here and download the Peripheral Driver User's Guide

   Go here and download the 8962 Data Sheet

Initial Board Set-Up

9. **Connect and power the EVB**

   Connect the LM3S8962 Evaluation Board to the LM3S2110 CAN Device Board using the 10-way ribbon cable included in the kit. The cable inserts into the header labeled CAN on each board.

   Note: Do <u>not</u> connect the larger 20-pin ribbon cable between the two boards. This will enable JTAG debugging of the LM3S2110. You could inadvertently overwrite the code in the devices flash memory.

   Using the included USB cable, connect your eval board to a free USB port on your PC. Connect the mini-b (smaller) end of the USB cable to the connector labeled **P4** on the EVB. Connect the other end (Type A) to a free USB port on your host PC. A PC's USB port is capable of sourcing up to 500 mA for each attached device, which is sufficient for the evaluation board. If connecting the board through a USB hub, it must be a powered hub.

10. The Stellaris Evaluation Board USB port is a composite port and consists of three connections:

    Stellaris Virtual COM Port
    Stellaris Evaluation Board A
    Stellaris Evaluation Board B

    The Windows "Found New Hardware" notification should appear and all three connections should be installed automatically.

Quickstart Application

The LM3S8962 Evaluation Board comes preprogrammed with a quickstart application. Once you have powered the board, this application runs automatically. You probably already noticed this running as you installed the drivers. A splash screen appears on the OLED display for a few seconds before the application begins.

The quickstart application is a game in which you navigate a character through a maze. Use the directional push buttons to move the character, and the user pushbutton (**SELECT**) to fire a missile to destroy the monsters. Score accumulates for maze progress and the number of monsters destroyed. The game lasts for only one character "life". The score displays at the end of the game.

Since the OLED display on the evaluation board has burn-in characteristics similar to a CRT, the application also contains a screen saver. The screen saver only becomes active if two minutes have passed without the user pushbutton being pressed while waiting to start the game (that is, the screen saver never appears during game play).

After two minutes of running the screen saver, the display turns off and the user LED blinks. Exit either mode of screen saver by pressing the user pushbutton (**SELECT**). Press the button again to start the game.

The LM3S8962 Evaluation Board uses the CAN module on the LM3S8962 to communicate with the included LM3S2110 CAN device board. During game play, the volume is adjusted by using the **up** and **down** buttons on the CAN device board. The Status LED on the CAN device board will blink during various portions of the quick start application.

The LM3S8962 microcontroller contains an integrated Ethernet controller, which is also used by the game to display the entire maze on the PCs screen. The game is programmed to obtain an IP address from a DHCP server or, if it fails to do that, it will default to one that has been pre-programmed.

11. PC Networking Changes

Connect the Ethernet port of the LM3S8962 evaluation board to your computer's Ethernet port with the included cable. These cables aren't the highest quality. If you don't see the yellow and green LEDs on the port flickering, you may want to use another cable.

Press the RESET button on the board. After 45 seconds or so, you should see an IP address at the bottom of the OLED display. This is the IP address of the LM3S8962 Ethernet interface. Our board defaulted to an address of 169.254.254.108, yours may be different.

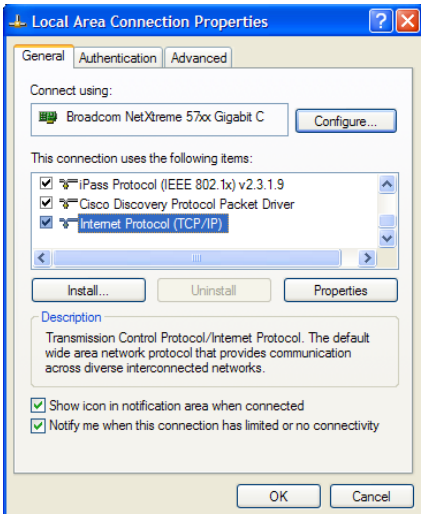
If you have a 802.11, AirCard™, 3G or other wireless Internet connection, **disable** it at this time. Also disable any VPN connections. We want to guarantee that your browser makes its connection attempt through the Ethernet cable.

If you have problems connecting to the board in later steps, you may also need to disable your firewall software.

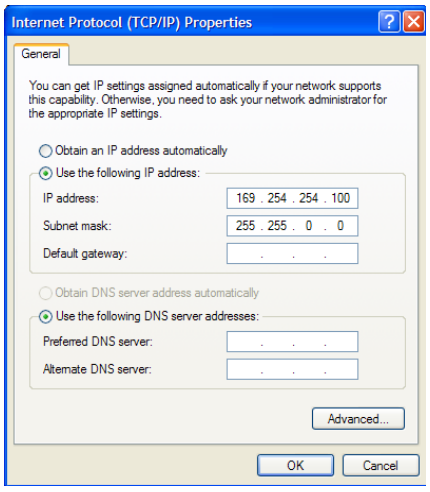
Since there is no DHCP server present to allocate an address to the Ethernet port on the LM3S8962 board, the **lwIP** stack has defaulted to a preprogrammed IP address. In order to communicate, we need to provide the PCs Ethernet port a suitable address. Depending on the OS that you're using, the following procedure may be slightly different.

In Windows XP:

Go to **Start → Control Panel → Network Connections → Local Area Connection**. Click the **Properties** button. When the **Local Area Connection Properties** window appears, select **Internet Protocol (TCP/IP)** like shown below:



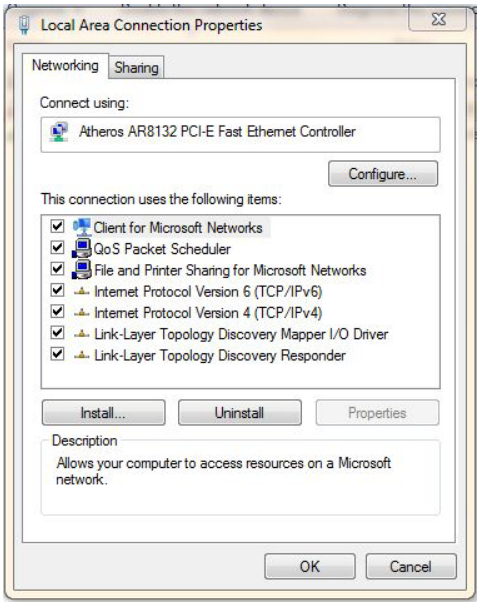
Click the **Properties** button. When the **Internet Protocol (TCP/IP) Properties** window appears, make a mental (or other) note of the settings you see, then make the selections shown below (if your boards' IP address was different, simply make sure that the first three fields are the same as shown on the OLED, and the fourth is different). Since our board defaulted to an IP address of 169.254.254.108, we're going to set our computer's Ethernet port to an address of 169.254.254.100. Again, your settings may be different.



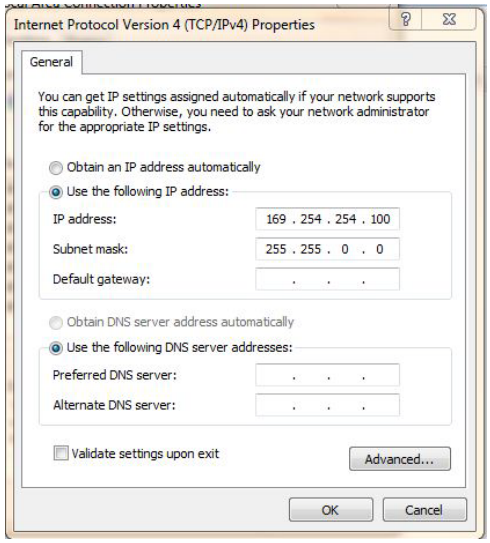
Click **OK**. Close the **Local Area Connection Properties** and **Local Area Connection Status** windows.

In Windows 7:

Go to **Start → Control Panel → Network and Internet → Network and Sharing Center** and click on **Change adapter settings** on the left. Click on **Local Area Connection**. Click on **Internet Protocol Version 4 (TCP/IPv4)** in the list and click **Properties**.



When the **Internet Protocol Version 4 (TCP/IPv4) Properties** window appears, make a mental (or other) note of the settings you see, then make the selections shown below (if your boards' IP address was different, simply make sure that the first three fields are the same as shown on the OLED, and the fourth is different). Since our board defaulted to an IP address of 169.254.254.108, we're going to set our computer's Ethernet port to an address of 169.254.254.100. Again, your settings may be different.



Click **OK**. Close the **Networking** windows.

12. **Start your browser**
    If your board has gone to sleep, wake it up by clicking the SELECT button. Start your web browser, and **enter** the address shown on your OLED display (ours was 169.254.254.108) and press **Enter**. A web page, like the one below, served from the LM3S8962 should appear.

You should be able to see your position, and the positions of the bad guys in the maze, This makes it easier to win, especially since you can now see the exit from the maze.

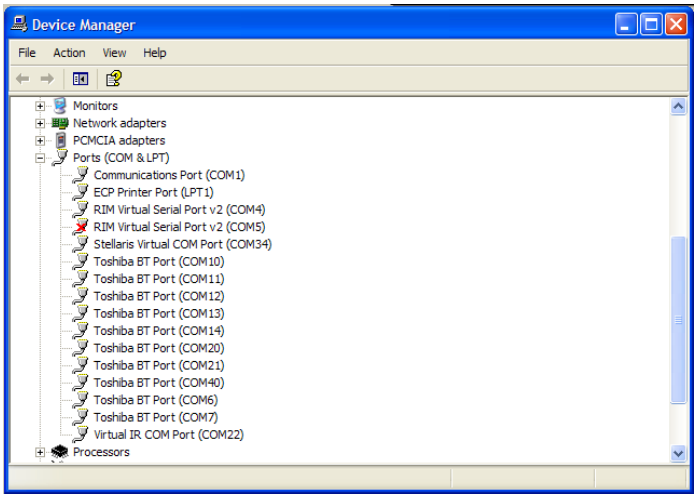
If the maze image above appears as a gray box on your browser, you likely do not have Java installed on your PC. Make sure that you have the Java runtime engine installed, as shown in step 4 of this lab procedure.

13. **UART Connection**

While the game is being played, a running tally of the score is output through UART0 of the LM3S8962 microcontroller. UART0 is connected to the FTDI's second serial channel. This serial channel is available to Windows as a Virtual COM port. To determine which COM# Windows has assigned to the Virtual COM port on the LM3S8962 board, follow these steps:

From the **Start Menu**, click **Run …** then type **devmgmt.msc** in the dialog box and click **OK**.
Click on the **+** symbol to expand the **Ports (COM & LPT)** group.

Stellaris Virtual COM Part (COM#) is listed as shown in the figure below. This COM# is the device you connect to using your terminal application. In this example, the COM port is **COM34**.



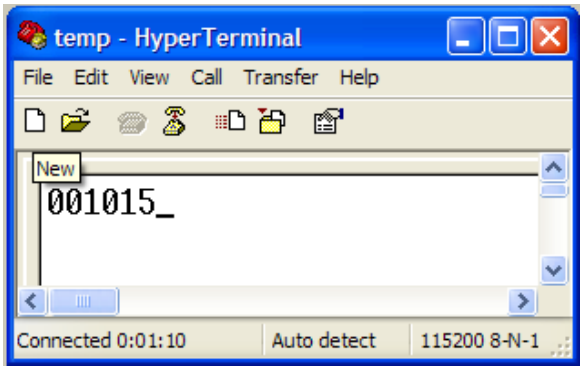
In Windows XP:
To view the score, open up **HyperTerminal** (usually found under **Start → All Programs → Accessories → Communications**).

Give the connection a name like **Temp**, then connect using COM#, where # is the number Windows has assigned the Virtual COM port. Set the serial connection to a baud rate of **115200**, **8** data bits, **no** parity, **1** stop bit, and **no** flow control. The game must be running to record a score.

In Windows 7:
To view the score, open up your terminal program (TeraTerm, for example).

Make a serial connection to COM#, where # is the number Windows has assigned the Virtual COM port. Set the serial connection to a baud rate of **115200**, **8** data bits, **no** parity, **1** stop bit, and **no** flow control. The game must be running to record a score.

Your HyperTerminal window should look something like this:

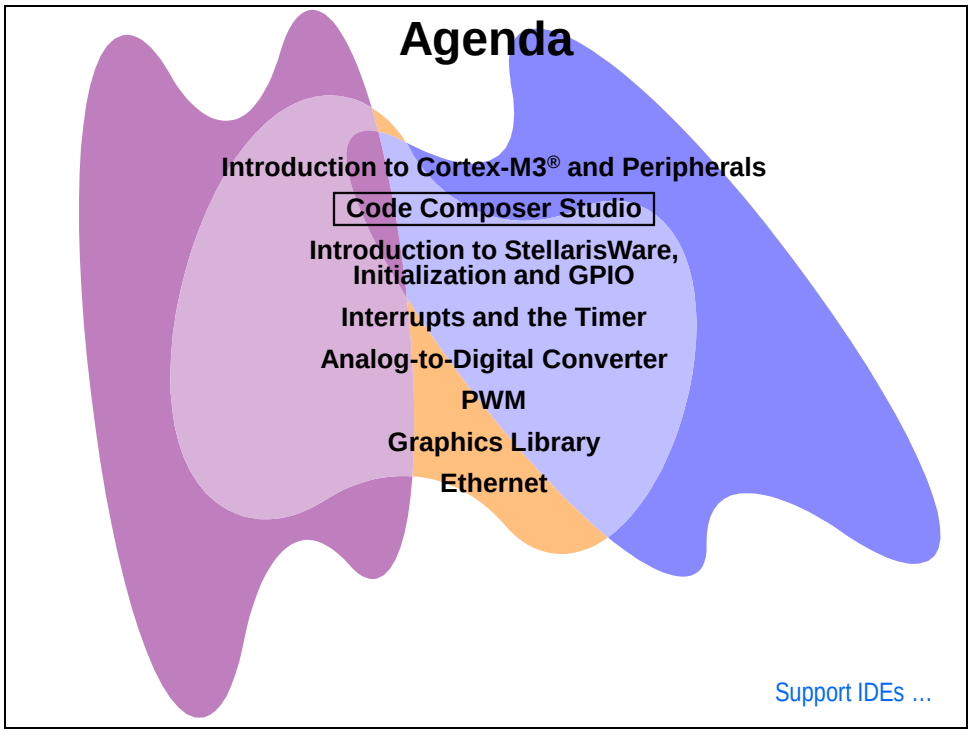


14. Close all of the windows on your desktop.

You're done.

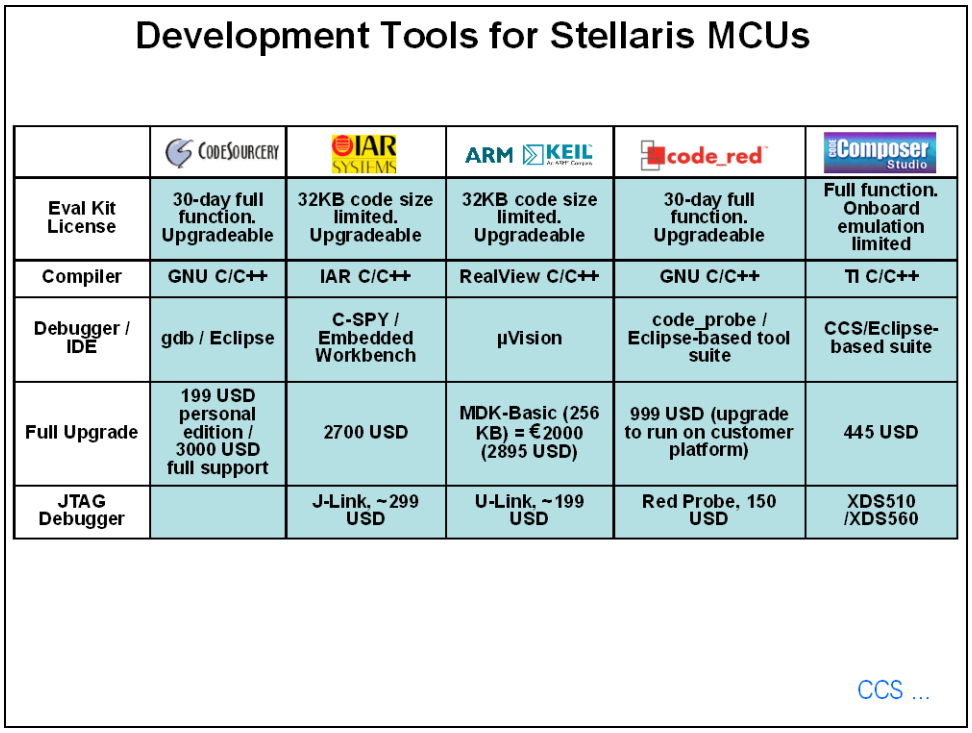
Introduction

This module will introduce you to the basics of Code Composer Studio. In the lab, we will set up a project from a blank page and try out some Code Composer features.



Module Topics

Stellaris Development Tools



	CodeSourcery	IAR Systems	ARM KEIL	code_red	TI Composer Studio
Eval Kit License	30-day full function. Upgradeable	32KB code size limited. Upgradeable	32KB code size limited. Upgradeable	30-day full function. Upgradeable	Full function. Onboard emulation limited
Compiler	GNU C/C++	IAR C/C++	RealView C/C++	GNU C/C++	TI C/C++
Debugger / IDE	gdb / Eclipse	C-SPY / Embedded Workbench	µVision	code_probe / Eclipse-based tool suite	CCS/Eclipse-based suite
Full Upgrade	199 USD personal edition / 3000 USD full support	2700 USD	MDK-Basic (256 KB) = €2000 (2895 USD)	999 USD (upgrade to run on customer platform)	445 USD
JTAG Debugger		J-Link, ~299 USD	U-Link, ~199 USD	Red Probe, 150 USD	XDS510 /XDS560

Code Composer Studio

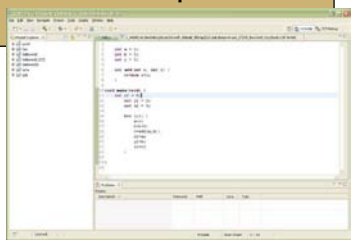
What is Code Composer Studio?

- Integrated development environment for TI embedded processors
  - Includes debugger, compiler, editor, simulator, OS…
  - The IDE is built on the Eclipse open source software framework
  - Extended by TI to support device capabilities
- CCSv5 is based on "off the shelf" Eclipse (version 3.7 in CCS 5.1)
  - Future CCS versions will use unmodified versions of Eclipse
    - TI contributes changes directly to the open source community
  - Drop in Eclipse plug-ins from other vendors or take TI tools and drop them into an existing Eclipse environment
  - Users can take advantage of all the latest improvements in Eclipse
- Integrate additional tools
  - OS application development tools (Linux, Android…)
  - Code analysis, source control…
- Linux support soon
- Low cost!

User Interface Modes…

User Interface Modes

- Simple Mode
  - By default CCS will open in simple/basic mode
  - Simplified user interface with far fewer menu items, toolbar buttons
  - TI supplied Edit and Debug Perspectives
- Advanced Mode
  - Uses default Eclipse perspectives
  - Very similar to what exists in CCSv4
  - Recommended for users who will be integrating other Eclipse based tools into CCS
- Possible to switch Modes
  - Users can decide that they are ready to move from simple to advanced mode or vice versa



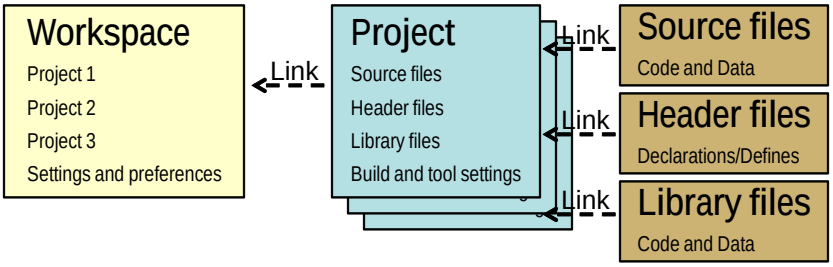
Common Tasks…

Common tasks

- **Creating New Projects**
  - **Very simple to create a new project for a device using a template**
- **Build options**
  - **Many users have difficulty using the build options dialog and find it overwhelming**
  - **Updates to options are delivered via compiler releases and not dependent on CCS updates**
- **Sharing projects**
  - **Easy for users to share projects, including working with version control (portable projects)**
  - **Setting up linked resources has been simplified**

Workspaces and Projects…

Workspaces and Projects

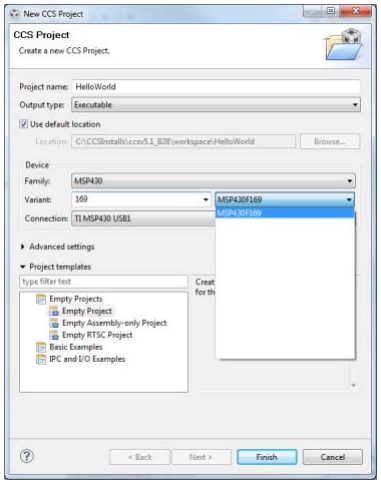


A workspace contains your settings and preferences, as well as links to your projects. Deleting projects from the workspace deletes the links, not the files

A project contains your build and tool settings, as well as links to your input files. Deleting files from the workspace deletes the links, not the files

Project Wizard…

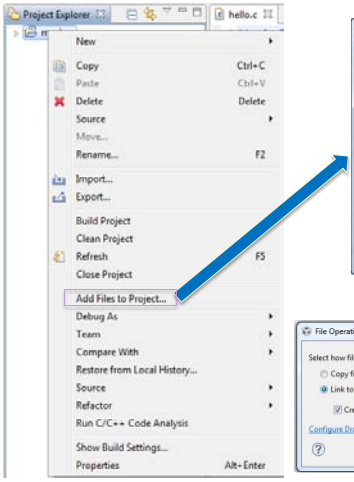
Project Wizard

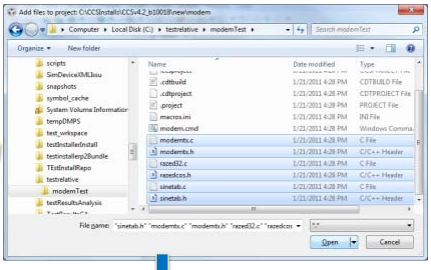


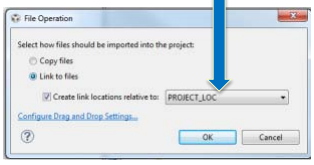
- **Single page wizard for majority of users**
  - Next button will show up if a template requires additional settings
- **Debugger setup included**
  - If a specific device is selected, then user can also choose their connection, ccxml file will be created
- **Simple by default**
  - Compiler version, endianness… are under advanced settings

Add Files…

Adding Files to Projects







- Add Files to Project allows users to control how the file is added to the project
- Linking Files using built-in macros allows easy creation of portable projects

Lab 2…

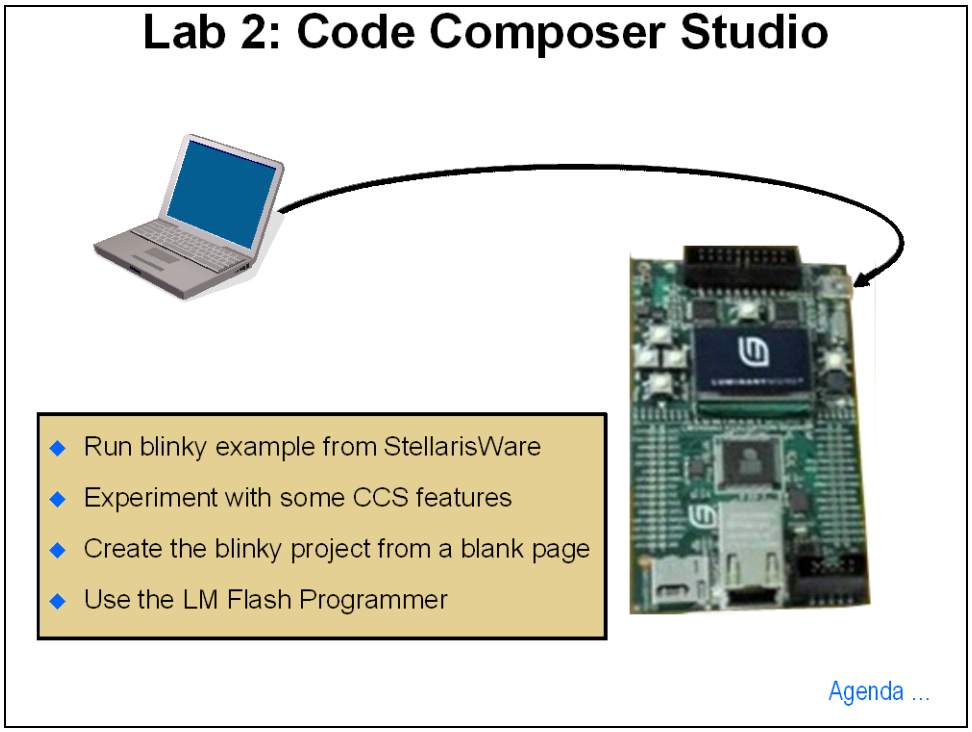
Lab2: Code Composer Studio

Objective

The objective of this lab exercise is to learn how to create a portable, easy to use Code Composer Studio project and utilize some of the CCS features.

This will not be a lab covering the code itself.

In any IDE, many of the underlying details are hidden from view. Sometimes that's a good thing, and sometimes it's not. In this lab we'll try to pull back the curtains and reveal the underlying issues involved.



Test the Example blinky Project

Open Code Composer Studio

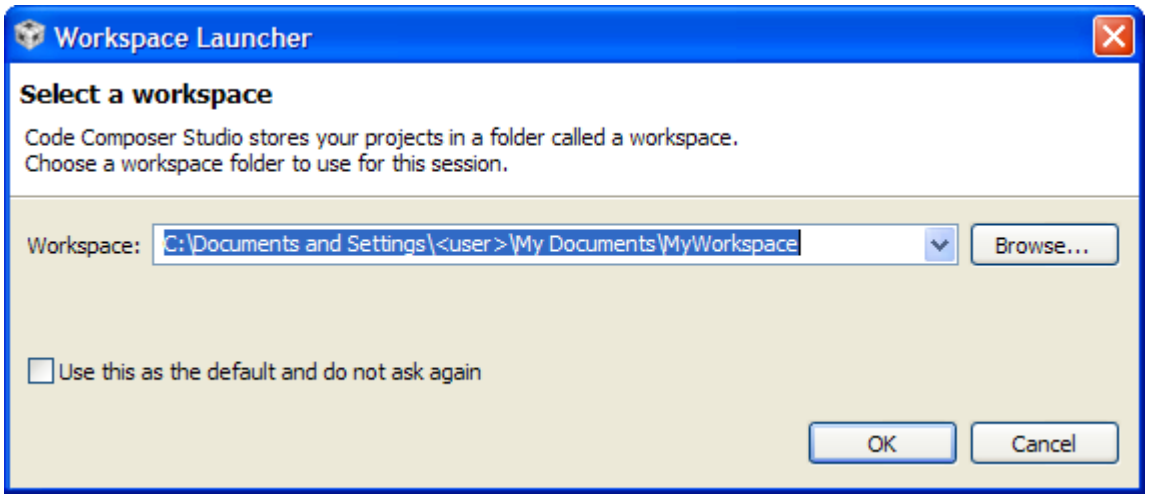
1. Double click on the Code Composer shortcut on your desktop to start CCS. 

   When the “Select a workspace” dialog appears, type

   ```
   C:\Documents and Settings\<user>\My Documents\MyWorkspace
   ```

   Obviously, replace <user> with your own username.

   Do not check the “Use this as the default and do not ask again” checkbox. (If at some point you accidentally check this box, see the Workshop Appendix for hints) Click OK. The location of the workspace folder is not critical, but to make the project portable, we want to locate the workspace outside the StellarisWare directory.



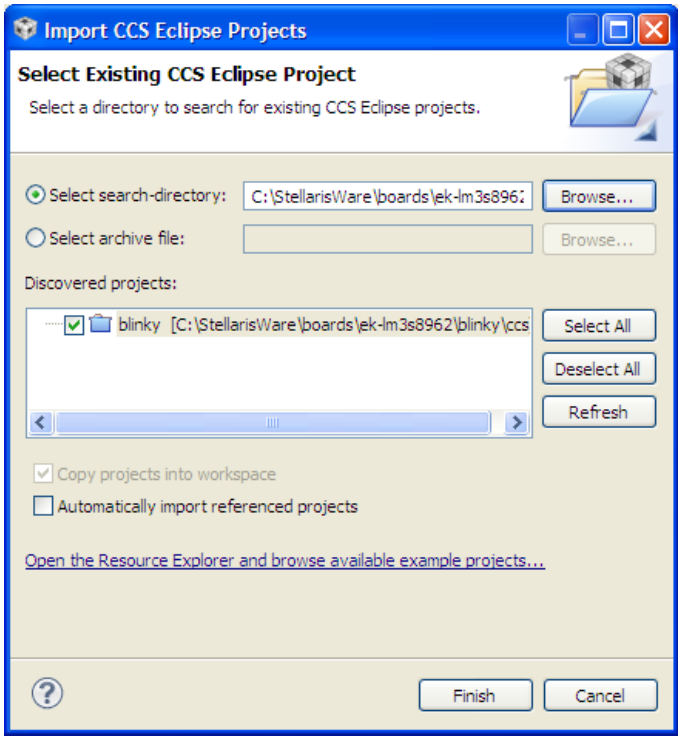
   In the next few steps you will be asked to license Code Composer. When that happens, select “Free License”. As long as your PC is connected to the evaluation board (not as a target emulator), Code Composer will have full functionality, free of charge.

   If the “TI Resource Explorer” window appears, close the tab. At this time it only offers MSP430 resources.

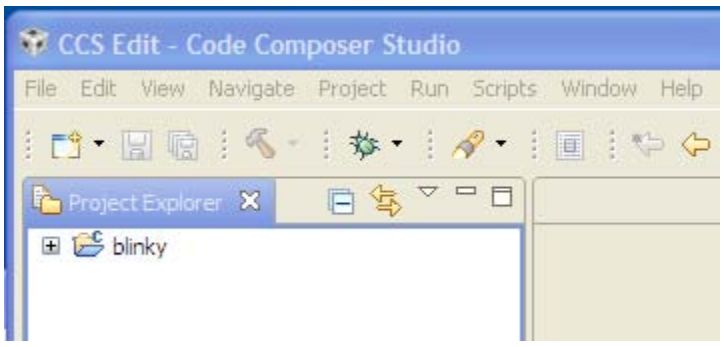
   Maximize the Code Composer window.

Import blinky Project

2. On the CCS menu bar select Project → Import Existing CCS/CCE Eclipse Project. In the search-directory box, browse to `C:\StellarisWare\boards\ek-lm3s8962\blinky` and click OK. Click Finish.



Your Code Composer project pane should look like the below:



Set Active Project

3. Click on blinky in the Project Explorer pane to make the project Active.

Debug, Download and Run

4. Make sure that your evaluation board is plugged in, then click the Debug  button on the CCS menu bar to build and download the `blinky` project. When the process completes, CCS will be in the Debug perspective. (Note the two tabs in the upper right of your screen) You can create as many additional perspectives as you like for your specific needs. Only the Debug and Edit perspectives are pre-defined.

5. Click the Run  button on the CCS menu bar to run the code. Observe the LED blinking on right side of your evaluation board.

Some CCS Features

6. In the code window in the middle of your screen, find the `while(1)` loop starting around line 66. There are four lines of code in the loop that blinks the LED. Click the Halt  button on the CCS menu bar. Pick a line of code inside the while loop and double-click in the gray area to the left of the line number to set a breakpoint. Click the Run  button to restart the code. The program will stop at the breakpoint and you will see an arrow on the left of the line number, indicating that the program counter has stopped on this line of code. **Note that the current driver for the evaluation board does not support adding/removing breakpoints while the processor is running.** Click the Run button a few times or press the F8 key to run the code. Observe the LED on the eval board as you do this.

   Remove all the breakpoints you have set at once by clicking Run → Remove All Breakpoints. Again, breakpoints can only be removed when the processor is not running.

7. Click on View → Registers to see the core and peripheral register values. Resize the windows if necessary. Click on the plus sign left of Core Registers to view the registers. Only the peripherals that are enabled can be read. In this project you can view Core Registers, GPIO_PORTF (where the LED is located), HIB, FLASH_CTRL, SYSCTL and NVIC.

8. Click on View → Memory Browser to examine processor memory. Type 0x00 in the entry box and press Enter. You can page through memory and you can click on a location to directly change the value in that memory location.

9. Double click on the variable `ulLoop` in the code window around line 88. Right click on the selected variable and select Add Watch Expression and then click OK. The window on the upper right will switch to the Watch view and you should see the variable listed. It should report "identifier not found" if your code is running. Halt your code and the watch will update.

10. Click on Terminate  to return to the editor perspective. Right-click on blinky in the Project Explorer pane and select Close Project.

Starting a Project from a Blank Page

An easy way to start a new project can be to simply take an existing project, like blinky, and modify it. But that can hide issues that may become a problem later, like fixed paths and portability.

So, engineers often want to start with a new project and just bring in the parts of the example they need to make their design work. This can lead to some common problems – so here is how to avoid them.

Create a New Project

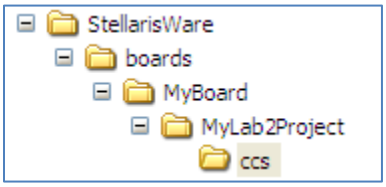
1. We need to create some folders on the hard drive where our project and files will reside. If you take a moment and look at the folder structure of existing StellarisWare boards and projects, you'll see that this arrangement mirrors it precisely.

   Open Windows Explorer and navigate to `C:\StellarisWare\boards`. Right-click in the open space of the right-hand pane and select New → Folder. Name the new folder `MyBoard` and press the Enter key.

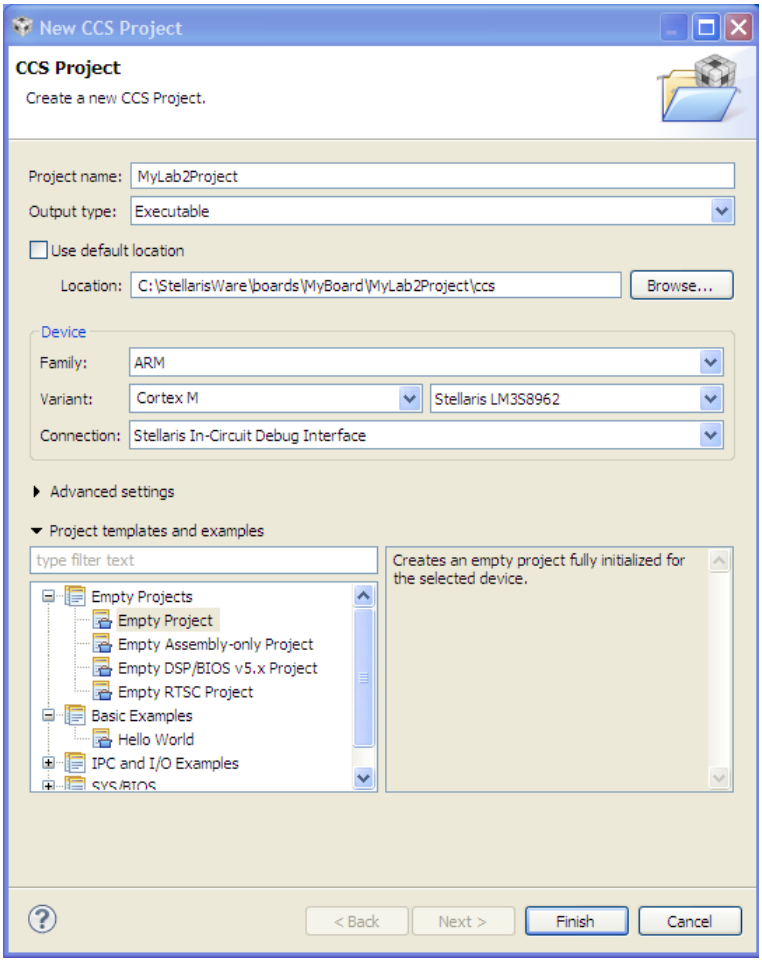
   Double click on `MyBoard` to enter the folder and then right-click in the wide open right-hand pane. Select New → Folder. Name the new folder `MyLab2Project` and press the Enter key.

   Double click on `MyLab2Project` to enter the folder and then right-click in the wide open right-hand pane. Select New → Folder. Name the new folder `ccs` and press the Enter key.

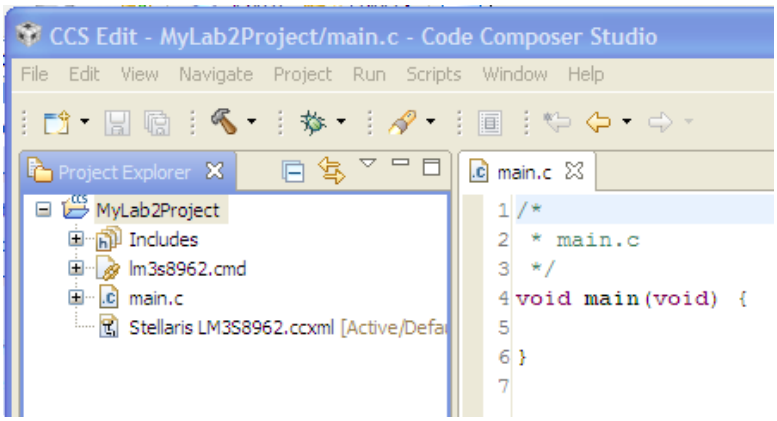
   Some folders were removed to make the screen shot below easier to see:



2. On the CCS menu bar select File → New → CCS Project. Make the selections shown below. Make sure to uncheck the "Use default location" checkbox and select the correct path to the "ccs" folder you created. This step is critical to making your project functional and portable. Click Finish.



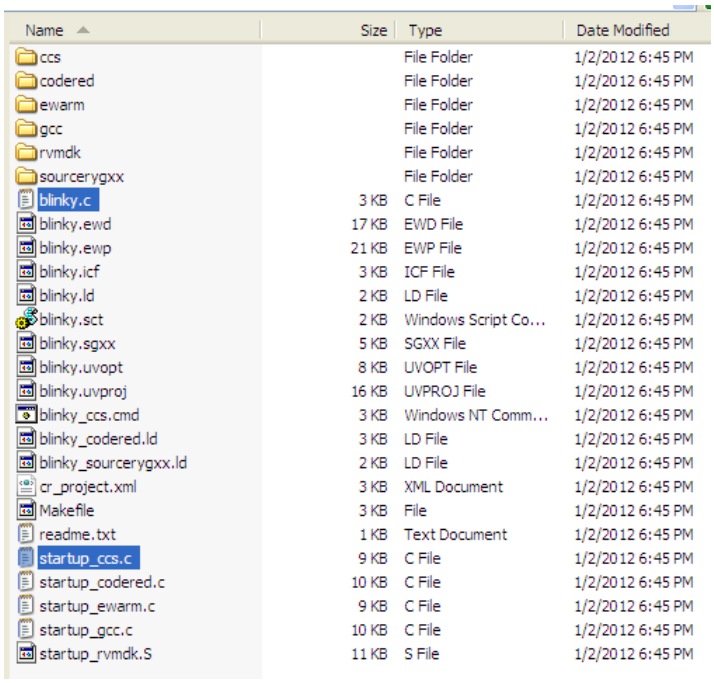
Code Composer will add your project to the workspace, add a file called main.c and open it for editing:

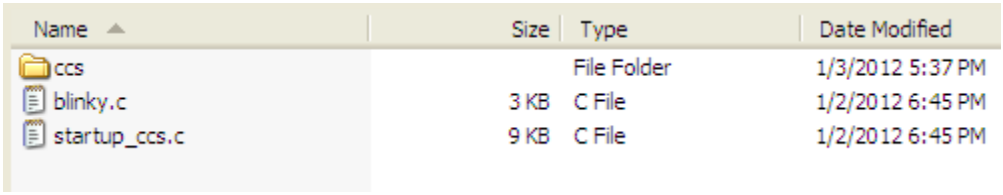


Add New Source Files

3. The `main.c` file was added by the New Project wizard to make it easy to start programming. Unfortunately the file is in the wrong location for our purposes. Right-click on `main.c` in the Project Explorer pane and delete the file. Click on MyLab2Project to make the project active.

4. Since we're not doing any programming, let's do this the easy way by cutting and pasting our two source files; the code file and the CCS startup file. Open Windows Explorer and navigate to `C:\StellarisWare\boards\ek-lm3s8962\blinky`. Copy `blinky.c` and `startup_ccs.c` and paste them into your `C:\StellarisWare\boards\MyBoard\MyLab2Project` folder.

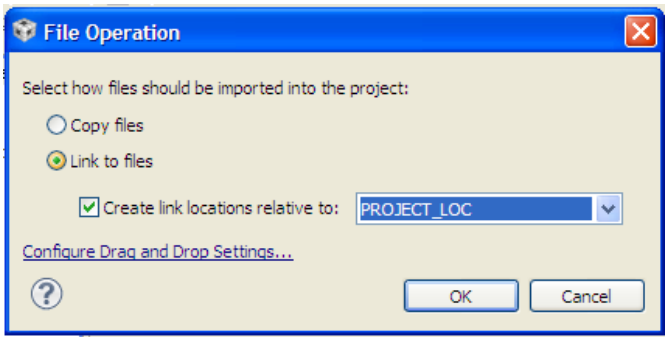




5. When you add a file to your project in any IDE, ask yourself; “how is this file located by the tool?” Is your file located with an absolute path or a relative path? If the answer is an absolute path, that will mean some problems will occur when you try to share your project with a colleague. Let’s start out by adding the files that you pasted into the `MyLab2Project` folder.

   Right-click on MyLab2Project in the Project Explorer pane and select Add Files… . File Naviagate to the `C:\StellarisWare\boards\MyBoard\MyLab2Project` folder and, using the Ctrl key, select both `blinky.c` and `startup_ccs.c`. Click Open.

   You will see a File Operation dialog. This dialog allows you to select how the file should be imported into the project. Let’s link them relative to the location of the project in `C:\StellarisWare\boards\MyBoard\MyLab2Project\ccs`. Make the selections as shown below and click OK.

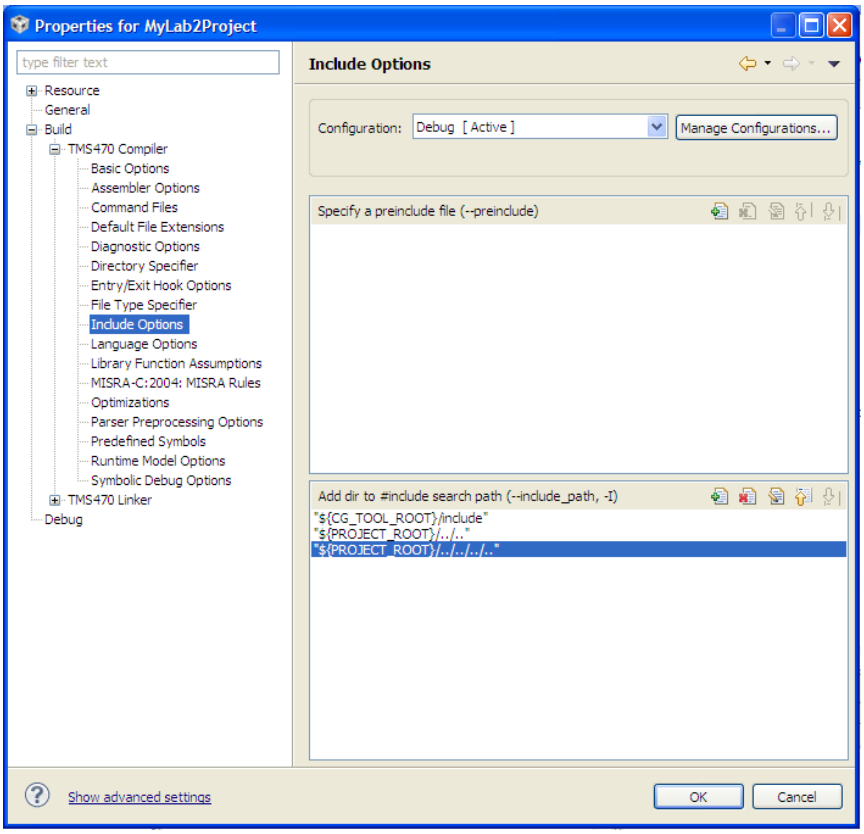


   If you want to link files into your project, this is how it should be done, so that the linked files can be packaged quickly for portability.

Set the Build Options

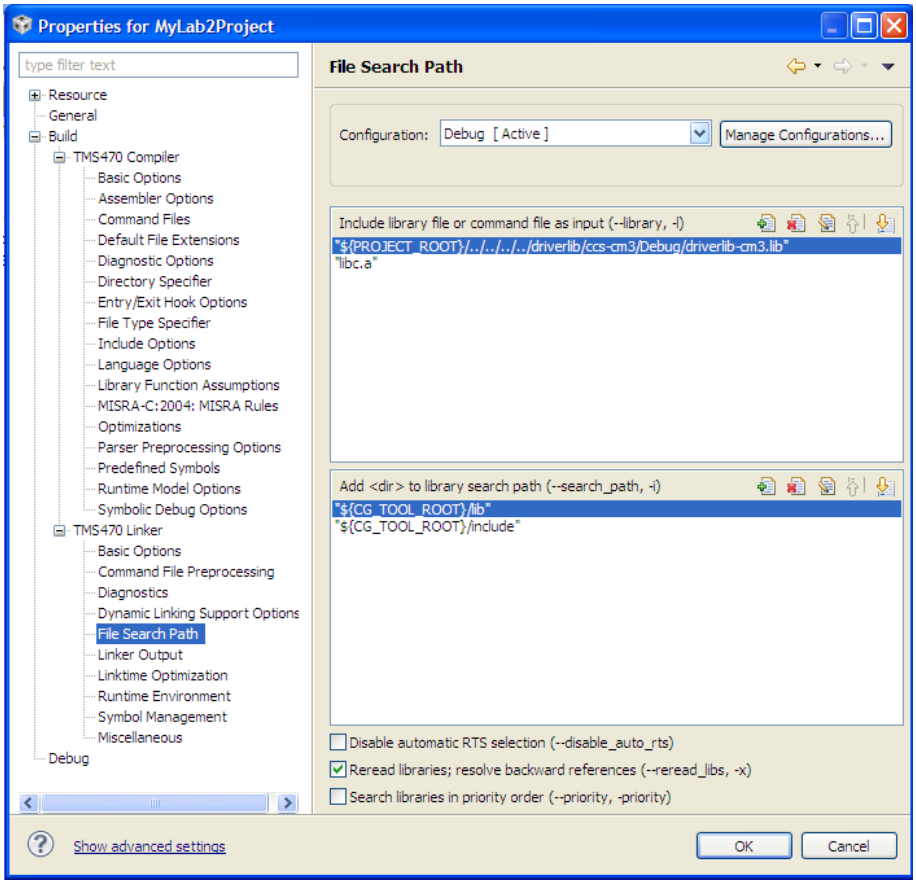
6. Right-click on MyLab2Project in the Project Explorer pane and select Properties. Click the + left of TMS470 Compiler and click on Include Options. In the bottom, include search path pane, click the Add button  and, one at a time, add the following two include search paths. You may want to copy/paste from the workbook pdf for the next few steps.

   ${PROJECT_ROOT}/../..
   ${PROJECT_ROOT}/../../../..



7. Click **File Search Path** under **TMS470 Linker**. Add the following include library file to the top window:

${PROJECT_ROOT}/../../../../driverlib/ccs-cm3/Debug/driverlib-cm3.lib



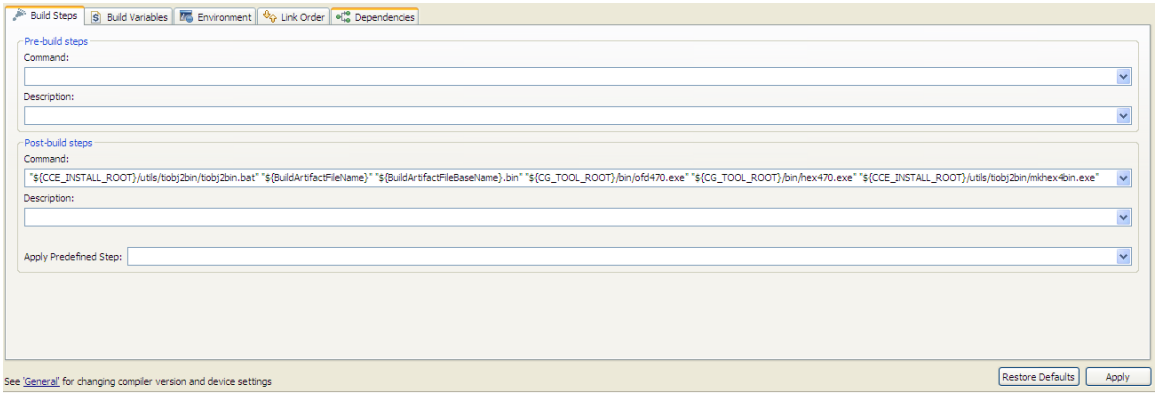
LM Flash Programmer

8. To use the LM Flash Programmer, Code Composer needs to generate a compatible output file. If you are just using Code Composer to load the program to your device, which is generally the case during development, this step is not needed.

   On the left, click **Build**. Add the following to the **Post-build steps** Command entry box:

```
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin.bat"
"${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.bin"
"${CG_TOOL_ROOT}/bin/ofd470.exe"
"${CG_TOOL_ROOT}/bin/hex470.exe"
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin.exe"
```

Note that there is a space between the quotation marks of each statement. These changes would look like the screen capture below:



Click **OK** to save all your changes.

Build

9. Run a test build to make sure that the project builds cleanly, with no errors, by clicking Project → Build All on the CCS menu bar. Keep an eye on the Console window that appears at the bottom of your screen. Any errors will appear there.

   Alternately, you could have clicked the Debug  button, which would have built the project, and if there were no errors, automatically loaded it to your device.

Debug

10. Click the Debug button. Since the project has already been built, the switch to debug mode will happen very quickly. Code Composer will open a connection and download the executable program to the Flash memory on your device.

11. Click the Run button to run the program. If you've done everything correctly, the LED should be flashing just like the blinky project did. Click on Terminate to return to the editor perspective.

12. Close any open editor windows. Then right-click on `MyLab2Project` in the Project pane and select Close Project. Minimize CCS.

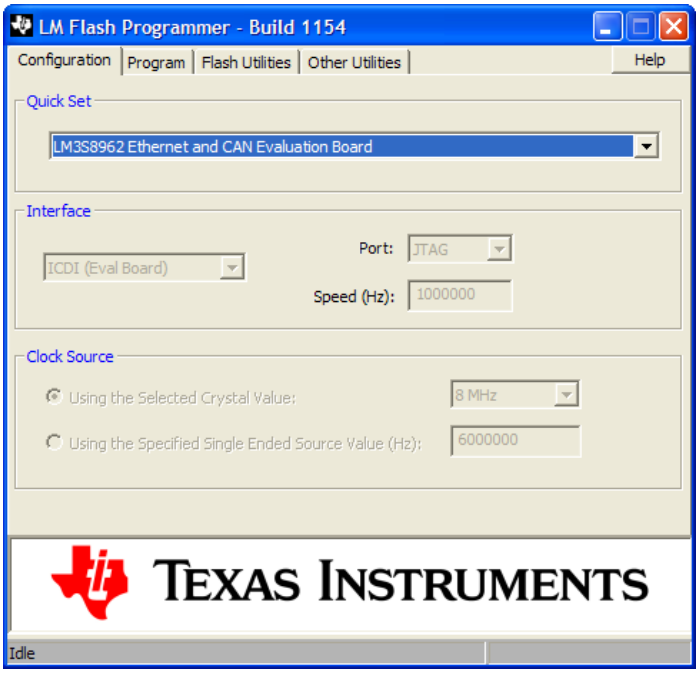
LM Flash Programmer

13. If you have not done so already, install the LM Flash Programmer onto your PC.

14. Make sure that Code Composer Studio is not actively debugging … otherwise CCS and the Flash Programmer may conflict for control of the USB port.

    There should be a shortcut to the **LM Flash Programmer** on your desktop, double-click it to open the tool. If the shortcut does not appear, go to Start → All Programs → Texas Instruments → LM Flash Programmer and click on LM Flash Programmer.

15. In order to know that we're programming something different than what is already in the devices Flash memory, let's program the board with the original QuickStart application. Select your evaluation board (in our case, the **LM3S8962 Ethernet and CAN Evaluation Board**) from the **Quick Set** pull-down menu under the **Configuration tab**. You can also manually configure the tool for targets that are not evaluation boards.

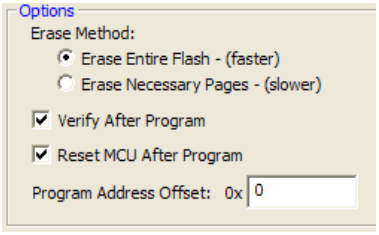


16. Click on the Program tab. Then click the Browse button and navigate to:

    `C:\StellarisWare\boards\ek-lm3s8962\qs_ek-lm3s8962\ccs\Debug\qs_ek-lm3s8962.bin`

    This is the QuickStart application that was programmed into the flash memory of the LM3S8962 at the factory. If you are using a different board, use the QuickStart bin file for that board.

    Note that there are applications here which have been built with each supported IDE. Make sure that the following checkboxes are selected:



17. Assure that your board is properly connected to your computer's USB port and the evaluation boards Debug port. Click the Program button.

    You should see the programming and verification status at the bottom of the window. After these steps are complete, the QuickStart application should be running on your evaluation kit.

18. Select the .bin file that you created in this lab. Click the Browse button and navigate to:

    `C:\StellarisWare\boards\MyBoard\MyLab2Project\ccs\Debug`

    and select `MyLab2Project.bin`. Click Open.

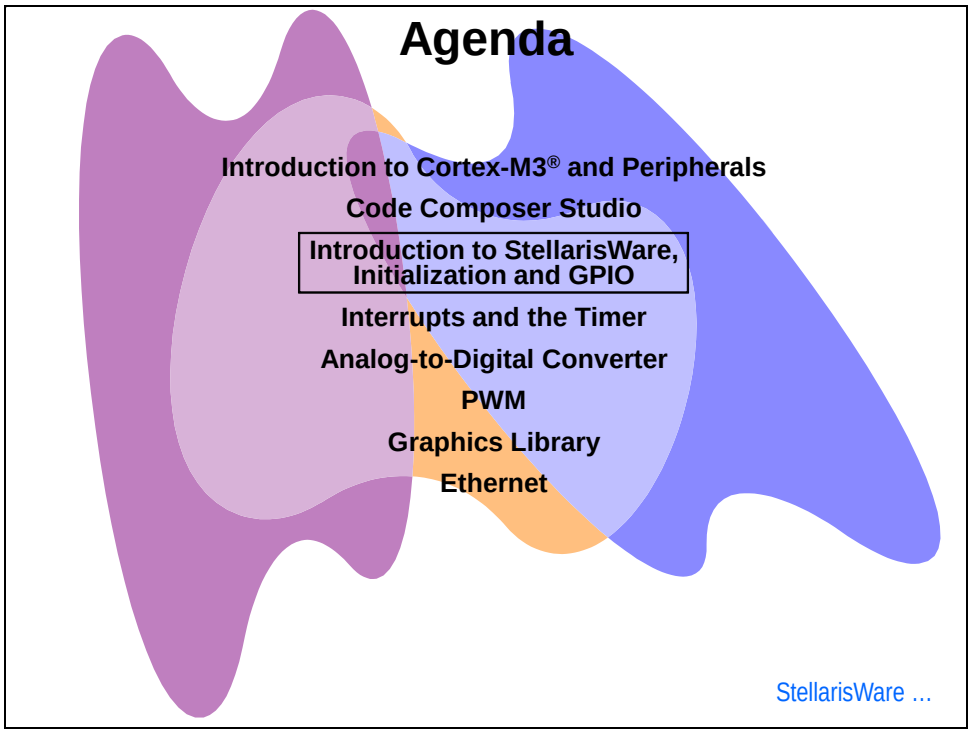
19. Program the device with your newly built .bin file by clicking the Program button. The LED on the board should blink as before. At the conclusion of this workshop, you can return the board to its originally programmed state by following the steps above to reprogram the QuickStart application into the Flash memory of the device.

    Close the LM Flash Programmer..

You're done.

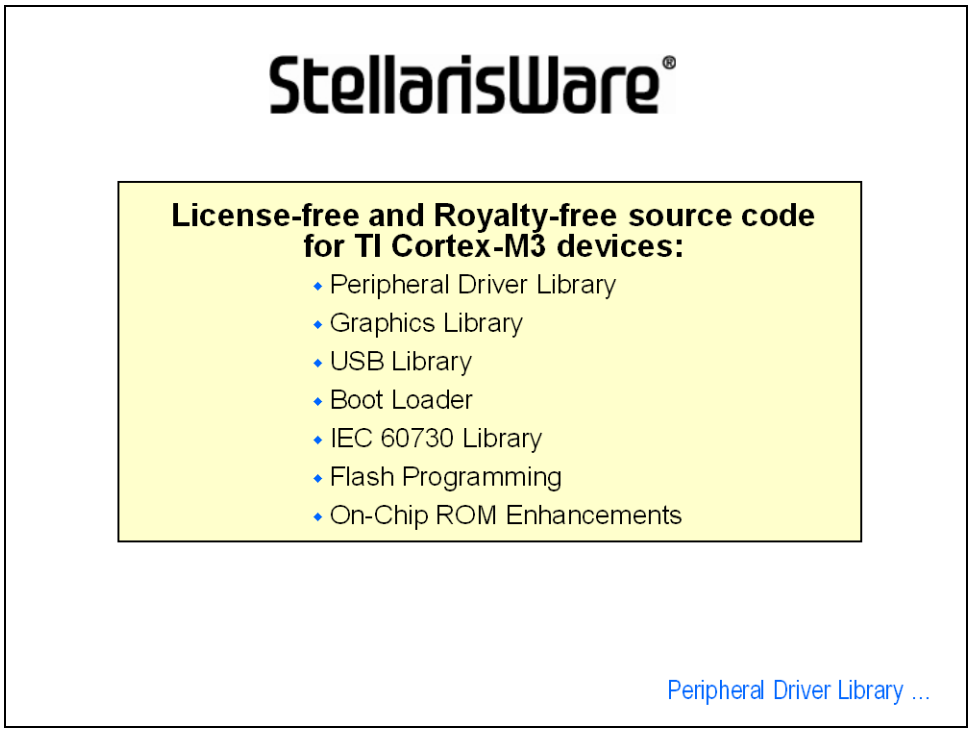
Introduction

This module will introduce you to StellarisWare. We will use several APIs from the driver library to set up the clock and the GPIO peripheral. We'll use a different API to control the GPIO pins.



Module Topics

StellarisWare



Peripheral Driver Library

Peripheral Driver Library

- High-level API interface to complete peripheral set
- License-free and royalty-free use for TI parts
- Simplifies and speeds development of applications
- Can be used for application development or as a programming example
- Available as object library and as source code
- Compiles on ARM/Keil, IAR, Code Red, CCS and GNU tools
- Peripheral driver library functions are preprogrammed in ROM on select Stellaris MCUs

Graphics Library ...

Graphics Library

Graphics Library

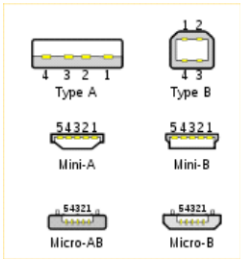
- Set of graphics primitives and widgets for use on Stellaris MCUs.
- Three subsequent layers of functionality:
  - Display Driver Layer
  - Graphics Primitives Layer
  - Widget Layer
  - Each API in each layer is directly callable
- Written entirely in C (except where not possible), self-contained, easy-to-understand, efficient.
- Compiles on ARM/Keil, IAR, Code Red, CCS and GNU tools.
- Computations that can be performed at compile time whenever possible.
- Graphics Primitives:
  - Point, Line, Rectangle, Circle, Font, Image, Context, Buffer
  - 134 Computer Modern predefined fonts available
  - Up to 24-bit color (~150 common colors conveniently referenced in GraphicsLib)
- Widgets:
  - Canvas, Checkbox, Container, Push Button, Radio Button, Slider, ListBox
- Special Utilities
  - *ftrasterize*: render your own font to be recognized by GraphicsLib
  - *lmi-button*: predefined button shape with shadow and 3-D
  - *pnmtoc*: Convert a NetPBM image file into a format recognized by GraphicsLib

USB Library ...

USB Library

USB Library Stacks and Examples

- **USB-IF Compliance**
  - Stellaris has passed USB Device and Embedded Host compliance testing
- **Device Examples:**
  - HID Keyboard
  - HID Mouse
  - CDC Serial
  - Generic Bulk
  - Audio class
  - Device Firmware Upgrade
  - Oscilloscope
- **Host Examples:**
  - Mass Storage
  - HID Keyboard
  - HID Mouse
- **Windows INF for supported classes**
  - Points to base Windows drivers
  - Sets config string
  - Sets PID/VID
  - Precompiled DLL saves development time
- **Device framework integrated into USBLib**



FREE Vendor ID/ Product ID sharing program

VID Request for embedded USB products

IEC60730 ...

IEC 60730

Safe At Home With IEC 60730

The International Electrotechnical Commission (IEC)

- IEC: World's authority in international standards for household appliances
- StellarisWare extension provides support for IEC 60730 Class B safety requirements
- Class B covers most home appliances, such as washers/dryers, refrigerators, freezers, and cookers/stoves
- Free license and royalty-free use for use on Stellaris MCUs
- Library supports both startup and periodic testing requirements of IEC 60730

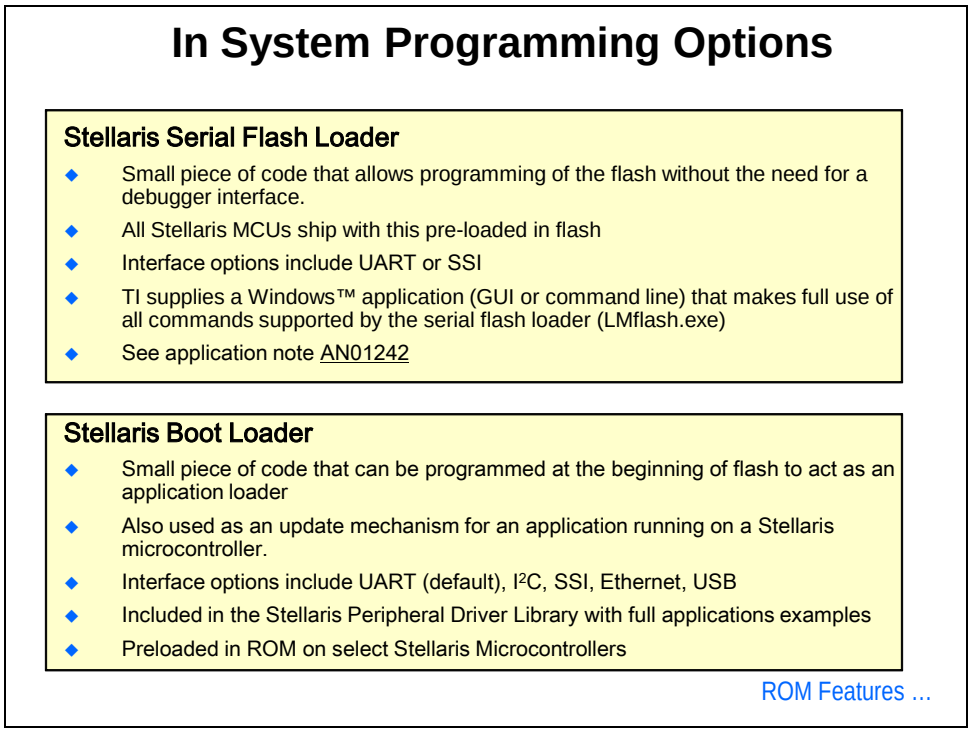
http://www.iec.ch/index.html

	Module	Description
StellarisWare™ Software	Reset Handler	Performs basic register and memory test out of reset.
	CPU Test	Performs stuck bit testing on the CPU PC and registers.
	SRAM Test	Performs stuck bit testing on the SRAM.
	Flash Test	Performs a CRC test on the Flash.
	ADC Test	Performs a conversion test on an ADC channel connected to a known voltage reference.
		Performs ADC temperature sensor test.
	GPIO Test	Performs GPIO input/output plausibility test.
	Clock/Interrupt Test	Performs tests to check the clock frequency, interrupt handling, and execution.
Stellaris® Hardware	Nested Vector Interrupt Controller	Deterministic, fast interrupt processing for execution certainty.
	Automotive-grade Flash Memory	High reliability non-volatile memory for robust environments.
	Cyclical Redundancy Check in ROM	Especially useful in verifying the contents of memory in a Stellaris microcontroller.
	2 Watchdog Timers	Clocked with precision oscillator, a second WDT takes advantage of the non-maskable interrupt (NMI) handler safety feature of the ARM Cortex-M3 processor.
	Precision Oscillator	Supplies an accurate, independent time base when periodic safety tests are executed.
	Advanced Motion Control with Multiple Fault Conditioning Inputs	Provides quick motor shutdown in low latency situations.
	Quadrature Encoder Inputs	Provides precise, closed loop control of motors.
	Integrated Analog Comparators	Used to trigger Stellaris' accurate ADC and to trigger an interrupt when needed, which is useful for infrequent out-of-range events such as a current or voltage spike.
		Eliminates the performance-wasting requirement of constant CPU polling.
	Internal Temperature Sensor	Used to monitor and shut down an appliance if the appliance overheats.
	10/100 Ethernet MAC/PHY with IEEE 1588 PTP	Offers highly synchronized connectivity features for precision internetworking.
	Controller Area Network (CAN) 2.0 MACs	

Note: Watchdog timers are completely independent hardware timers

In System Programming ...

In System Programming



ROM Features

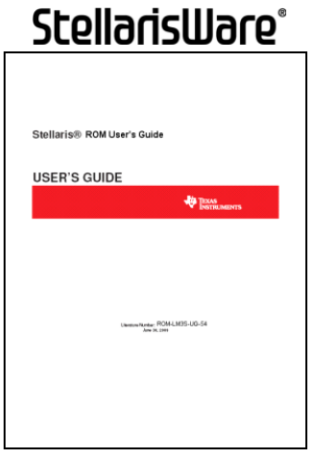
ROM Features

StellarisWare® DriverLib

- High-level API interface to complete peripheral set.
- Simplifies and speeds development of applications.
- Saves user flash by storing peripheral setup and configuration code
- Allows programmer focus to be on the application—not setup

Other flash memory-saving options

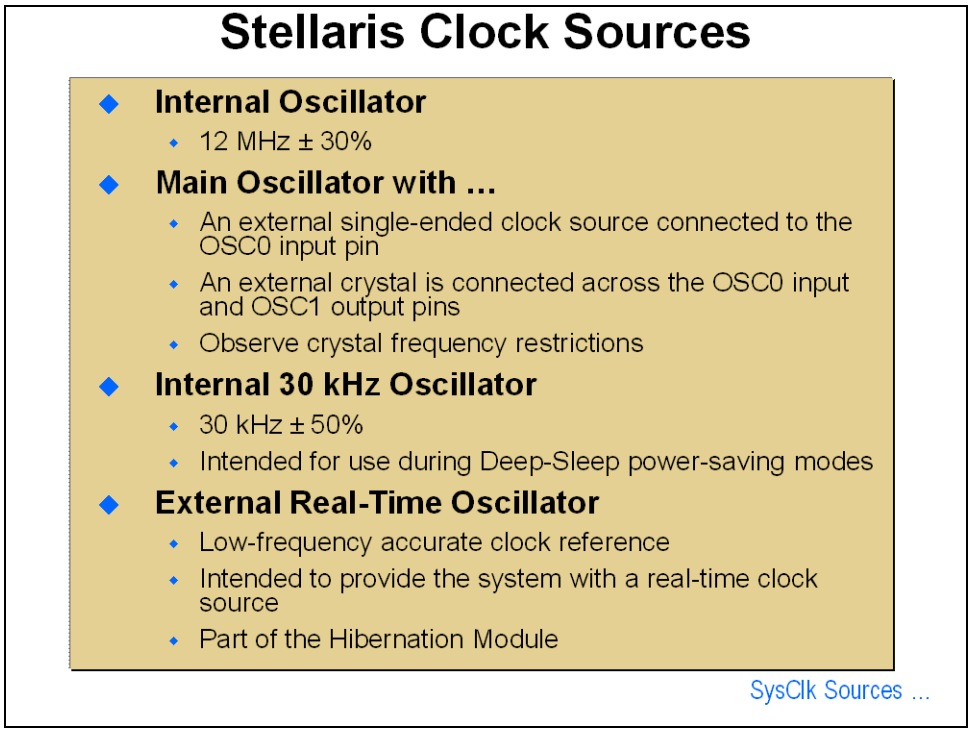
- Advanced Encryption Standard (AES) cryptographic tables
  - Supported by the current AES example application
  - 128, 192 and 256-bits
- Cyclic Redundancy Check (CRC) functionality – for error detection



Stored in ROM on select Stellaris MCUs

Clock Sources ...

Stellaris Clock Sources



System (CPU) Clock Sources

The CPU can be driven by any of the clock sources…

- Internal 12 MHz
- Main
- Internal 30 kHz
- External Real-Time

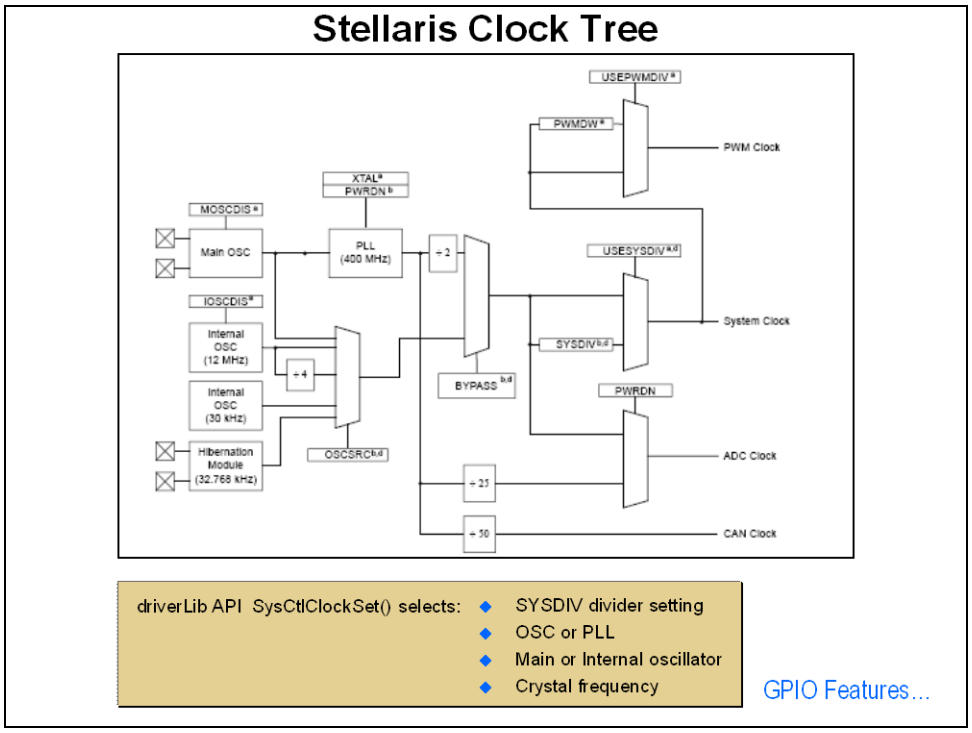
- Plus -

- The main internal PLL
- The internal 12 MHz oscillator divided by four (3 MHz ± 30%)
- PLL clock reference restrictions apply

Clock Source	Drive PLL?	Used as SysClk?
Internal 12 MHz	No	Yes
Internal 12 Mhz/4	No	Yes
Main Oscillator	Yes	Yes
Internal 30 kHz	No	Yes
External Real-Time Osc	No	Yes
PLL	-	Yes

Clock tree…

Clock Tree


Stellaris Clock Tree
USEPWMDIV a
PWMDIV a
PWM Clock
XTAL a
PWRDN b
MOSCDIS a
Main OSC
PLL
(400 MHz)
÷ 2
USESYSDIV a,d
IOSCDIS a
Internal
OSC
(12 MHz)
÷ 4
SYSDIV b,d
System Clock
BYPASS b,d
Internal
OSC
(30 kHz)
PWRDN
Hibernation
Module
(32.768 kHz)
OSCSRC b,d
ADC Clock
÷ 25
÷ 50
CAN Clock
driverLib API SysCtlClockSet() selects:
SYSDIV divider setting
OSC or PLL
Main or Internal oscillator
Crystal frequency
GPIO Features...

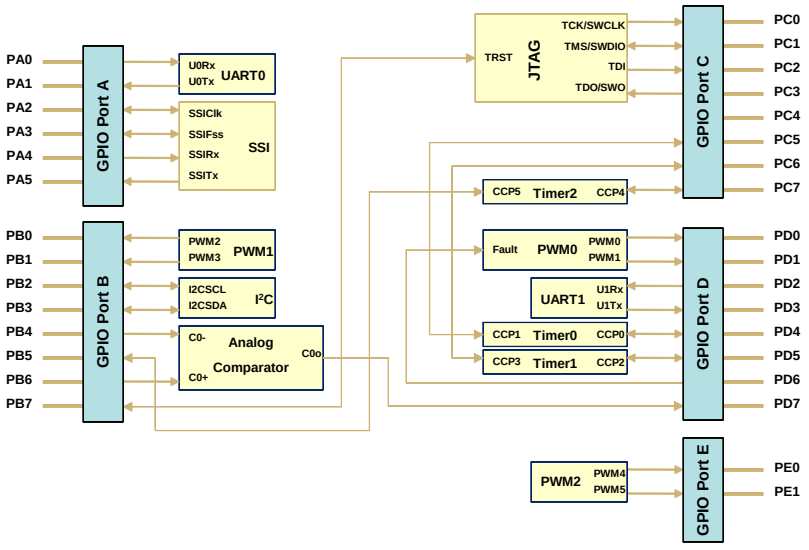
GPIO

GPIO - Features

- **Up to 42 GPIO pins**
- **Inputs …**
  - are 5V tolerant
  - are Schmitt Triggered
  - can be configured as Interrupts with
    - Edge triggering (Rising, Falling or Both)
    - Level triggering (High or Low)
  - can initiate ADC conversion
  - configurable with weak pull-up/pull-down resistors
- **Outputs …**
  - have 2mA, 4mA or 8mA drive capability with slew control for 8mA setting
  - have open-drain enable

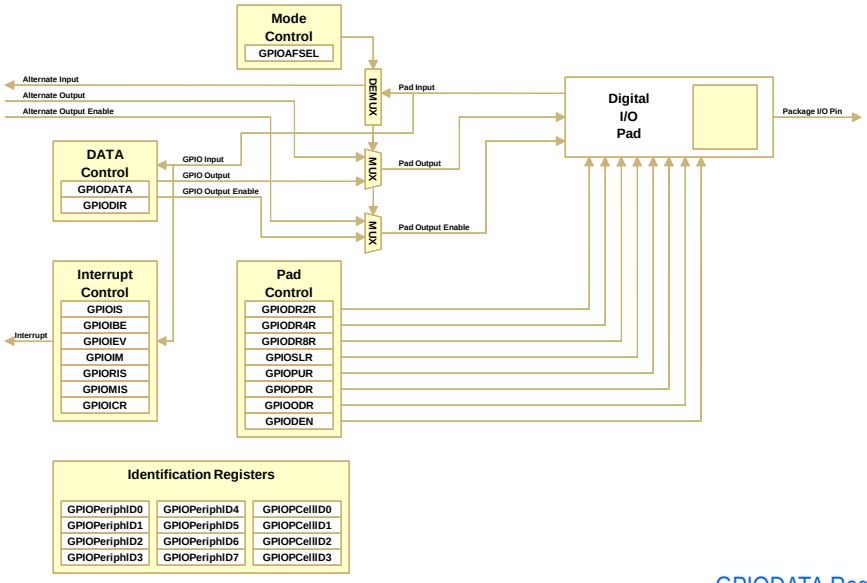
Module block diagram…

GPIO - Module Block Diagram



Port block diagram …

GPIO Port Block Diagram



GPIODATA Reg...

GPIO – Data Register Operation

GPIO – Data Register Operation

- The GPIO ports allow for the modification of individual bits in the **GPIO Data (GPIODATA)** register by using bits [9:2] of the address bus as a mask (256 locations)
- Software can modify individual GPIO pins in a single instruction, without affecting the state of the other pins
- Only bits with ADDR[9:2] that are equal to 1 will be read or written. For example: A mask of 0xFF (ADDR[9:0] = 0x3FF) would affect all bits on the port

GPIO Write...

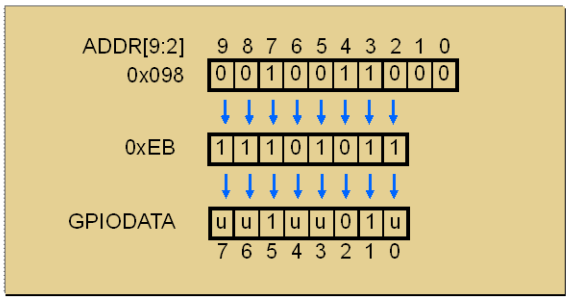
GPIO – Write

Write Example:

- If the address bit associated with the data bit is set to 1, the GPIODATA register bit is altered. If the address bit is cleared to 0, the GPIODATA register bit is unchanged.

Example: Set GPIODATA bits 5 and 1 HIGH. Clear bit 2. Leave the rest unchanged (u)

Process: Write a value of 0xEB to address GPIODATA + 0x098



GPIO Read...

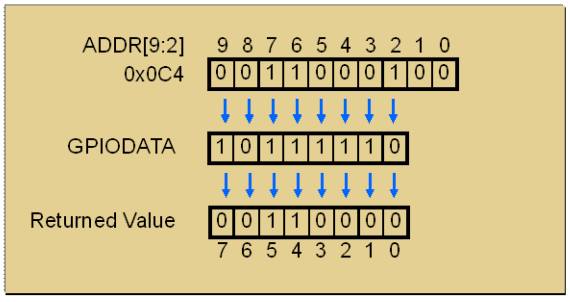
GPIO – Read

Read Example:

- If the address bit associated with the data bit is set to 1, the GPIODATA register bit is read. If the address bit is cleared to 0, the GPIODATA register bit is read as a zero.

Example: Read GPIODATA bits 5, 4 and 0

Process: Read address GPIODATA + 0x0C4

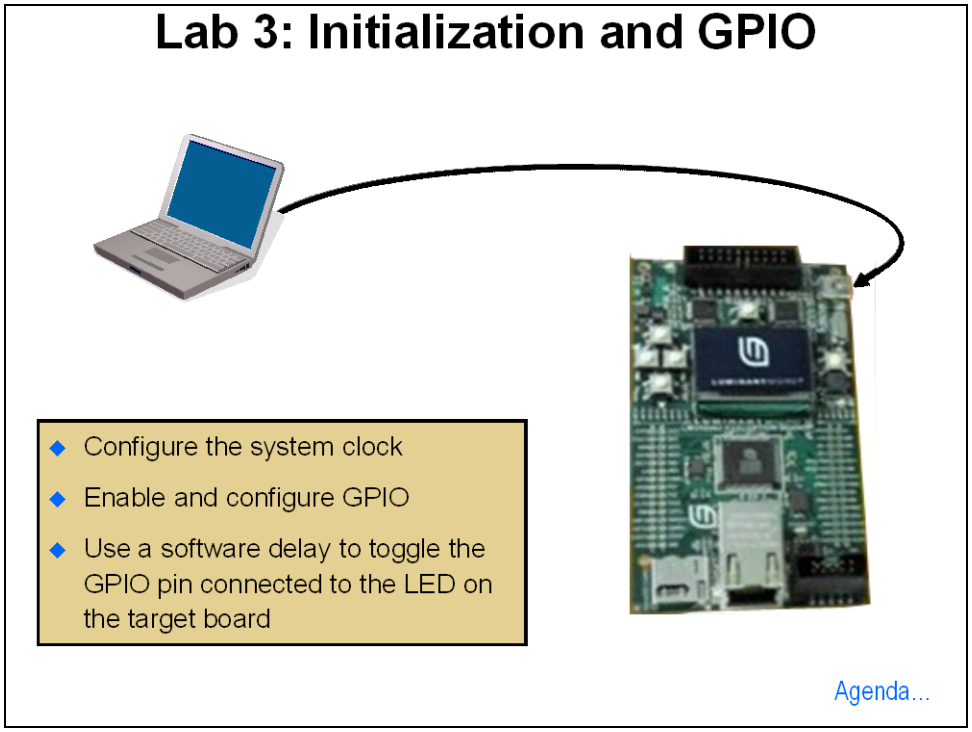


Lab 3...

Lab 3: Initialization and GPIO

Objective

In this lab we'll learn how to initialize the clock system and the GPIO peripheral. We'll then use the GPIO output to blink an LED on the evaluation board.



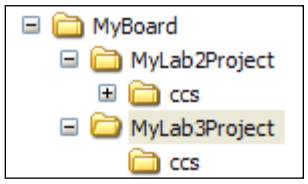
Procedure

Create New Project Folders

1. We need to create some folders to hold Lab 3.

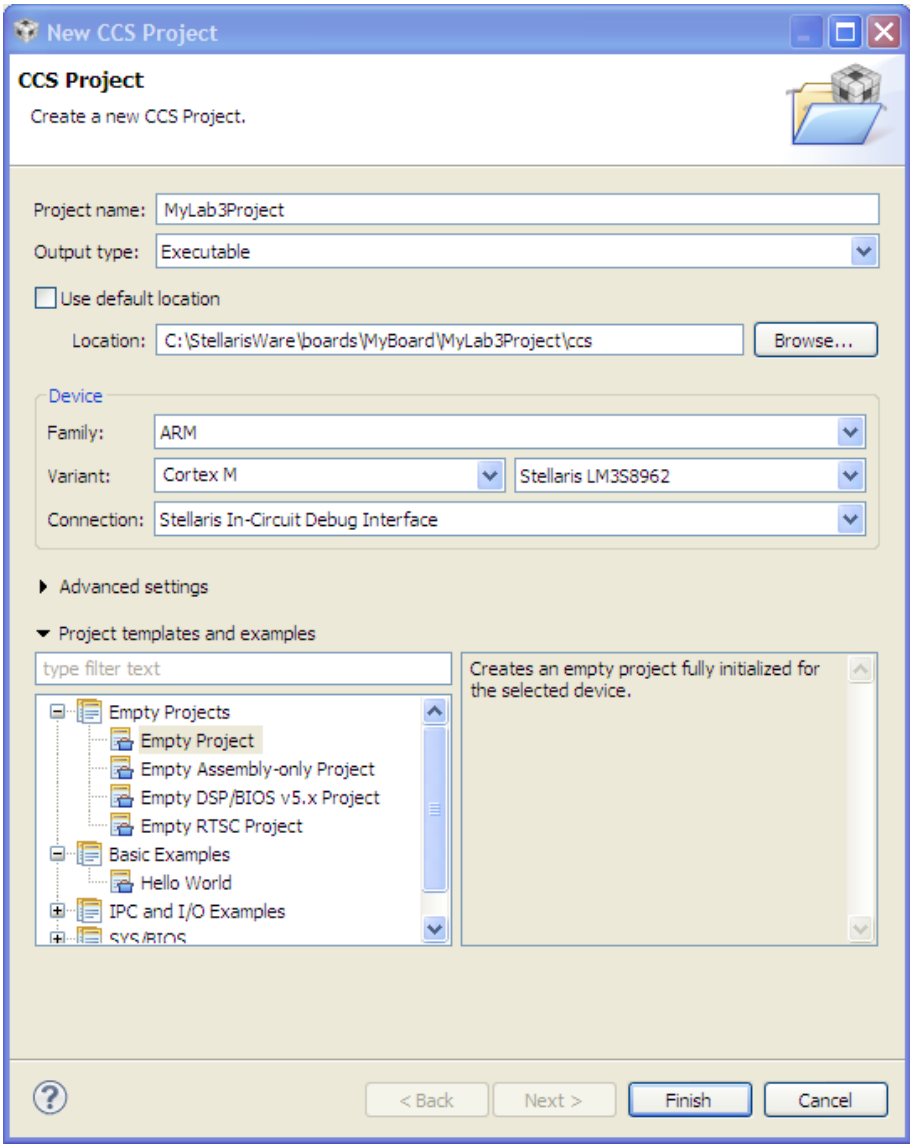
   Open Windows Explorer and navigate to `C:\StellarisWare\boards\MyBoard`. Right-click in the open space of the right-hand pane and select New → Folder. Name the new folder `MyLab3Project` and press the Enter key.

   Double click on `MyLab3Project` to enter the folder and then right-click in the wide open right-hand pane. Select New → Folder. Name the new folder `ccs` and press the Enter key.



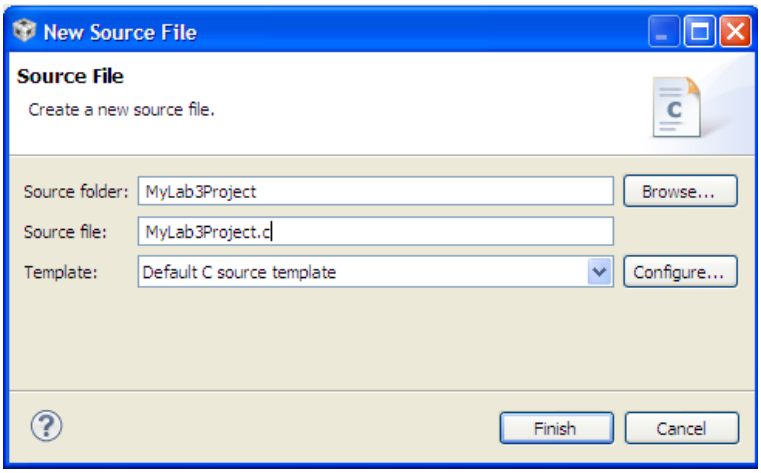
Create `MyLab3Project` Project

2. Maximize Code Composer. On the CCS menu bar select File → New → CCS Project. Make the selections shown below. Make sure to uncheck the "Use default location" checkbox and select the correct path to the "ccs" folder you created. This step is important to making your project portable. Click Finish.

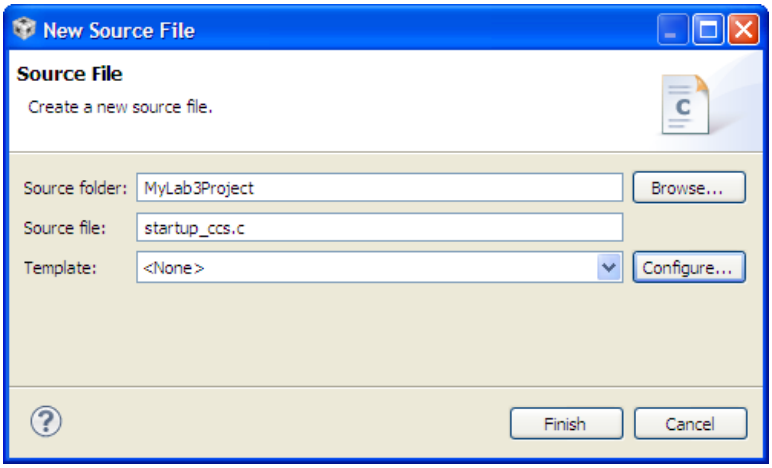


Add Source Files to Project

3. Delete `main.c` from MyLab3Project.
4. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the main C code file and click Finish.



5. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the startup file and click Finish.



The previous two steps create and link the files into our project. Eclipse places a restriction on the process though … the file must be in the project directory. That means these two files will land in C:\StellarisWare\boards\MyBoard\MyLab3Project\ccs. That's okay for the purposes of the workshop. If you want more control over the placement and linking of your files, use the steps in the previous lab.

Header Files

6. Type the following four lines into `MyLab3Project.c` to include the header files needed to access the StellarisWare APIs :

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
```

If you're prone to typing errors, it might be a good idea to cut/paste the code from the pdf file.

`hw_memmap.h` : Macros defining the memory map of the Stellaris device. This includes defines such as peripheral base address locations such as GPIO_PORTF_BASE

`hw_types.h` : Defines common types and macros such as tBoolean and HWREG(x).

`sysctl.h` : Defines and macros for System Control API of DriverLib. This includes API functions such as SysCtlClockSet and SysCtlClockGet.

`gpio.h` : Defines and macros for GPIO API of DriverLib. This includes API functions such as GPIOPinTypePWM and GPIOPinWrite.

Main() Function

7. Next, create your main application function and declare any variables that you will use later in the function:

```
int main(void)

{
  unsigned long ulPeriod;
  unsigned long ulDelay;

}
```

These two variables will be used later in calculating the period and delay needed for a 10 Hz signal. The return type needed for a main function varies depending on the tool chain. Since StellarisWare source can be built using several different tool chains, using the return type "`int`" is a good choice for flexibility in your code later.

Clock Setup

8. The system clock will default to running directly off of the internal oscillator after reset, which is not a precision oscillator. Configure the system clock to run directly from the Main Oscillator at 8 MHz (the crystal frequency on the 8962 eval board) with the following call.

   Leave a blank line for spacing and enter this single line of code inside `main()` after the variable declarations above:

```
SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
```

GPIO Configuration

9. Before calling any peripheral specific DriverLib function, we must enable the clock for that peripheral (RCGCn register). If you fail to do this, it will result in a Fault ISR (address fault).This is a common mistake for new Stellaris users. The second statement configures the GPIO as an output. Check the User's manual for your evaluation board to find out which GPIO pin is connected to the LED. Leave a line for spacing, then enter these two lines of code inside `main()` after the line in the previous step.

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0);
```

Period/Delay Calculation

10. To toggle a GPIO pin at 10 Hz and a 50% duty cycle, you need to generate a delay that is ½ of the period. We can perform this calculation at run time …

    First, calculate the number of clocks cycles required for a 10 Hz period by calling SysCtlClockGet() and dividing it by your desired frequency.

    Next, calculate the delay by dividing the period by two to get ½ of the period. Divide that result by 3 since the SysCtlDelay(ulCount) function takes 3 cycles per count. Finally, subtract an arbitrary amount from the delay to compensate for the delay added by the code in the rest of the loop. If you want, you can calculate this out based on the code the compiler generates, or take your best guess and then test it. Leave a line for spacing and enter the following two lines after the lines in the previous step:

```
ulPeriod = SysCtlClockGet() / 10;
ulDelay = ((ulPeriod / 2) / 3) - 4 ;
```

Toggle Loop

11. Finally, create a while (1) loop to send a "1" and "0" to the GPIO pin, with an equal delay between the two. To write the GPIO pin, use the GPIO API function call GPIOPinWrite. Make sure to read and understand how the GPIOPinWrite function is used. The third data argument is not simply a 1 or 0, but represents the entire 8-bit data port. The second argument is a bit-packed mask for data being written.

    In our example below, we are writing only bit 0 of Port F. This looks rather simple and user's make incorrect assumptions on how the function works. Now might be a good time to go to www.ti.com/stellaris , click on the Documentation tab, find and download the Datasheet for your StellarisWare device. Check out the GPIO chapter to understand the unique way the GPIO data register is designed and the advantages of this approach.

    Leave a line for spacing, and then add this code after the code in the previous step.

```
while(1)

{
    // Turn on the LED
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0x01);

    // Delay for a bit
    SysCtlDelay(ulDelay);

    // Turn off the LED
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0x00);


    // Delay for a bit
    SysCtlDelay(ulDelay);

}
```

12. Click the Save button to save your work. Your code should look something like this:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

int main(void)
{
  unsigned long ulPeriod;
  unsigned long ulDelay;

  SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);

  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0);

  ulPeriod = SysCtlClockGet() / 10;
  ulDelay = ((ulPeriod / 2) / 3) - 4 ;

  while(1)
    {
        // Turn on the LED
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0x01);

        // Delay for a bit
        SysCtlDelay(ulDelay);

        // Turn off the LED
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0x00);

        // Delay for a bit
        SysCtlDelay(ulDelay);
    }

}
```

Sorry about the small font here, but any larger font made the SysCtlClockSet() instruction look funny. If you're having problems, you can cut/paste this code into `MyLab3Project.c`.

Startup Code

13. In addition to the main file you have created, you will also need a startup file specific to the tool chain you are using. This file contains the vector table, startup routines to copy initialized data to RAM and clear the bss section, and default fault ISRs.

    Since this application does not use any interrupts, you can easily copy the "startup_ccs.c" file from the hello or blinky example and use that file. This also makes a good template to start with for more complex application that might require interrupts and complex fault handling.

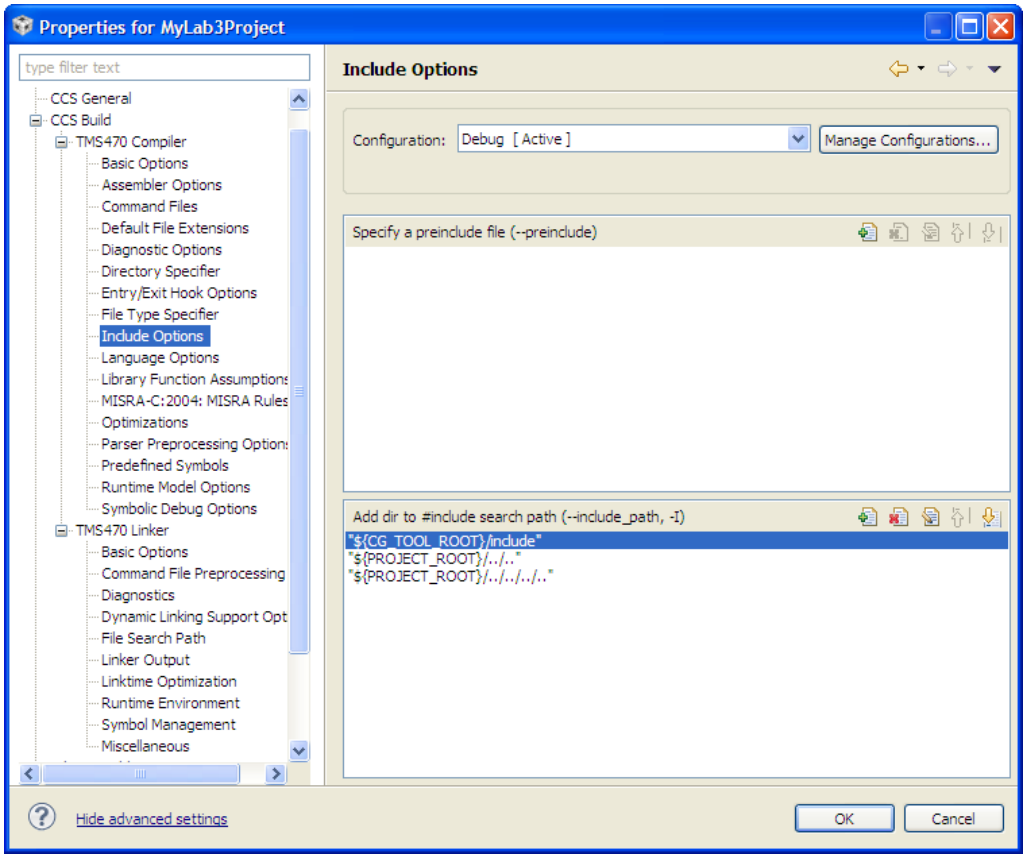
    From the CCS menu bar, click File → Open File and navigate to `C:\StellarisWare\boards`. Select the board that you are using (the ek-lm3s8962 in our case) and click on the `blinky` folder. Click on `startup_ccs.c` and click Open.

    Copy and paste the entire contents of the reference file you just opened into your blank `startup_ccs.c` file. Close the reference file. Click the Save button.

Set the Build Options

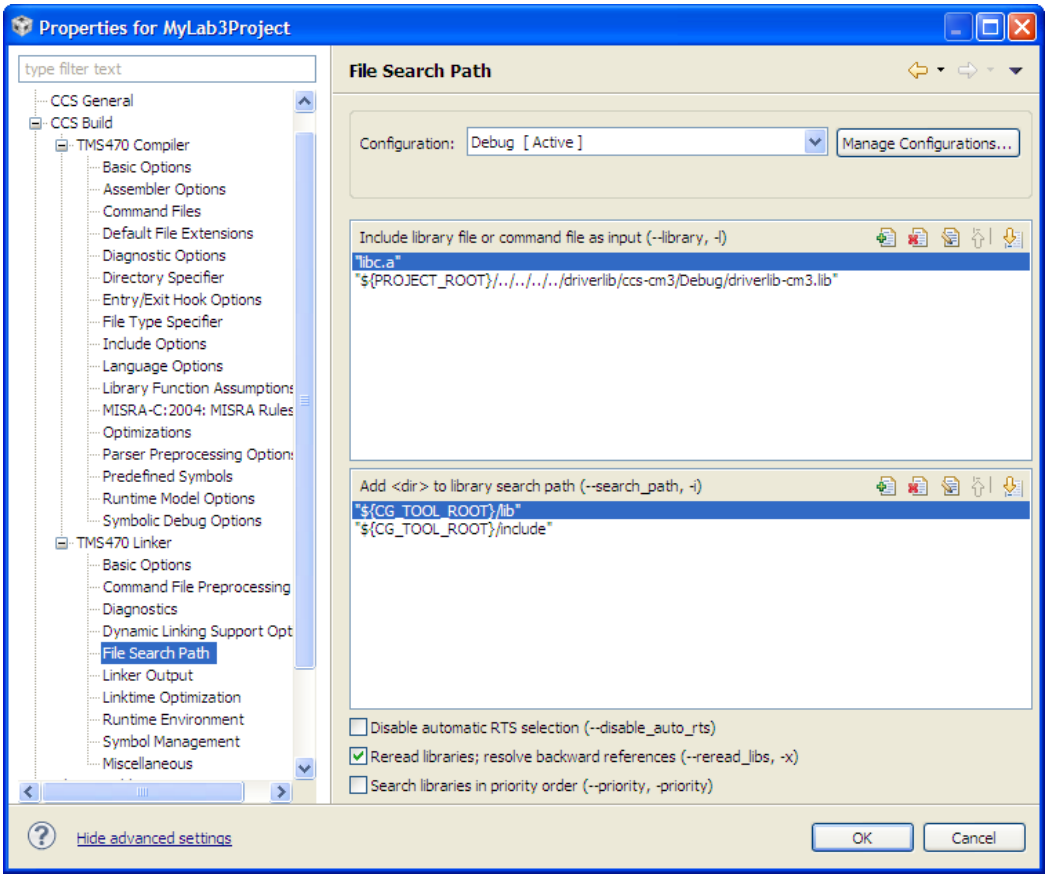
14. Right-click on MyLab3Project in the Project Explorer pane and select Properties. Click the + left of TMS470 Compiler and click on Include Options. In the bottom, include search path pane, click the Add button  and, one at a time, add the following two include search paths. You may want to copy/paste from the workbook pdf for the next few steps.

${PROJECT_ROOT}/../..
${PROJECT_ROOT}/../../../..



15. Click **File Search Path** under **TMS470 Linker**. Add the following include library file to the top window:

${PROJECT_ROOT}/../../../../driverlib/ccs-cm3/Debug/driverlib-cm3.lib



Click OK to save your changes.

Run the Code

16. Compile and download your application by clicking the Debug button  on the menu bar. If you have any issues, correct them, and then click the Debug button again. After a successful build, the CCS Debug perspective will appear.

    Click the Run button  to run the program that was downloaded to the flash memory of your device. The program will generate a 10 Hz square wave on Pin 0 of GPIO Port F. This pin is connected to the STATUS LED on the EK-LM3S8962.

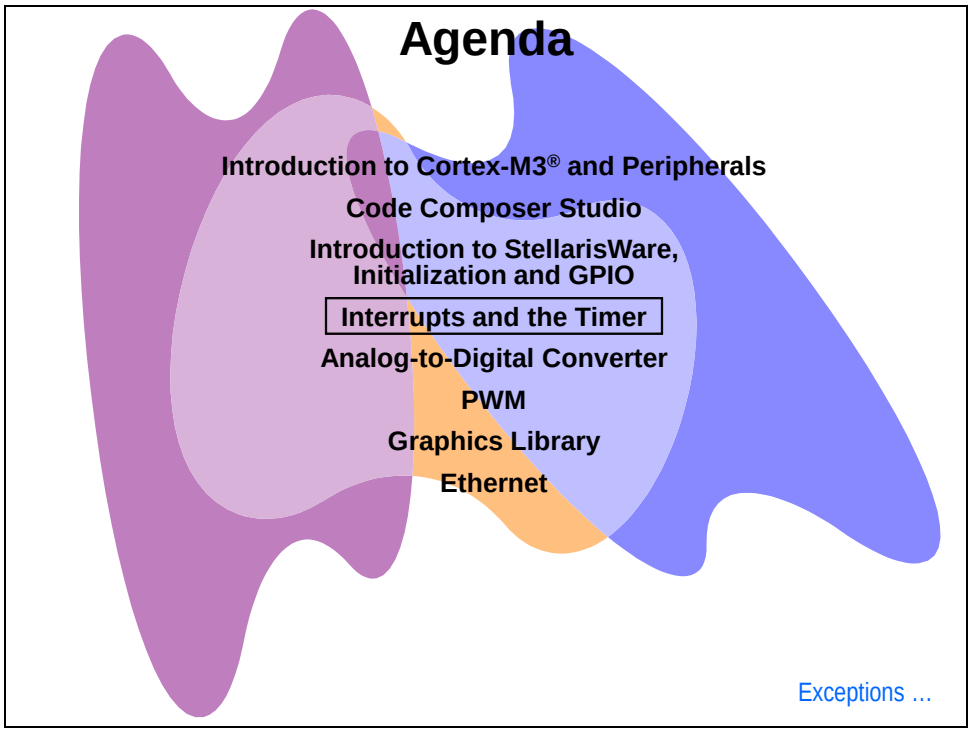
    When you're done, click the Terminate  button to return to the Editing perspective

17. Close any open editor windows. Then right-click on `MyLab3Project` in the Project pane and select Close to close the project. Minimize Code Composer Studio.

You're done.

Introduction

This module will introduce you to the use of interrupts on the Cortex-M3® and the general purpose timer module (GPTM). The lab will use the timer to generate interrupts which the Interrupt Service Routine code we write will respond to … blinking the LED.



Module Topics

Cortex-M3® Exceptions and Vector Table

Cortex-M3® Exceptions

N°	Exception Type	Priority	Vector address	Descriptions
1	Reset	-3	0x04	Reset
2	NMI	-2	0x08	Non-Maskable Interrupt
3	Hard Fault	-1	0x0C	Error during exception processing
4	Memory Management Fault	Configurable	0x10	MPU violation
5	Bus Fault	Configurable	0x14	Bus error (Prefetch or data abort)
6	Usage Fault	Configurable	0x18	Exceptions due to program errors
11	SVCall	Configurable	0x2C	SVC instruction
12	Debug Monitor	Configurable	0x30	Exception for debug
14	PendSV	Configurable	0x38	
15	SysTick	Configurable	0x3C	System Tick Timer
16 and above	Interrupt (IRQ)	Configurable	0x40	External interrupt

Vector Table...

Cortex-M3® Vector Table

- After reset, vector table is located in address 0
- Each entry contains the address of the function to be executed
- Address 0x00 is used as the starting value of Main Stack Pointer (MSP)
- Vector table is relocatable
- Open startup_ccs.c to see vector table coding

address	Vector
0x00	Initial Main SP
0x04	Reset
0x08	NMI
0x0C	Hard Fault
0x10	Memory Management Fault
0x14	Bus Fault
0x18	Usage Fault
0x2C	SVCall
0x30	Debug Monitor
0x38	PendSV
0x3C	SysTick
0x40	Interrupt (IRQ)
...	Other IRQs

GPTM Features...

General Purpose Timer Module (GPTM)

General Purpose Timer Module (GPTM) - Features

- GPTM, System Timer (SysTick) & PWM timer are all timing resources
- GPTM contains 4 timers (0-3)
- Each timer provides two 16-bit timer/counters (A/B)
- Timers can operate:
  - independently as timers or event counters
  - together as a 32-bit timer
  - together as a 32-bit Real-Time Clock (RTC)
- Timers can also be used for:
  - Pulse Width Modulation
  - To trigger analog to digital conversions

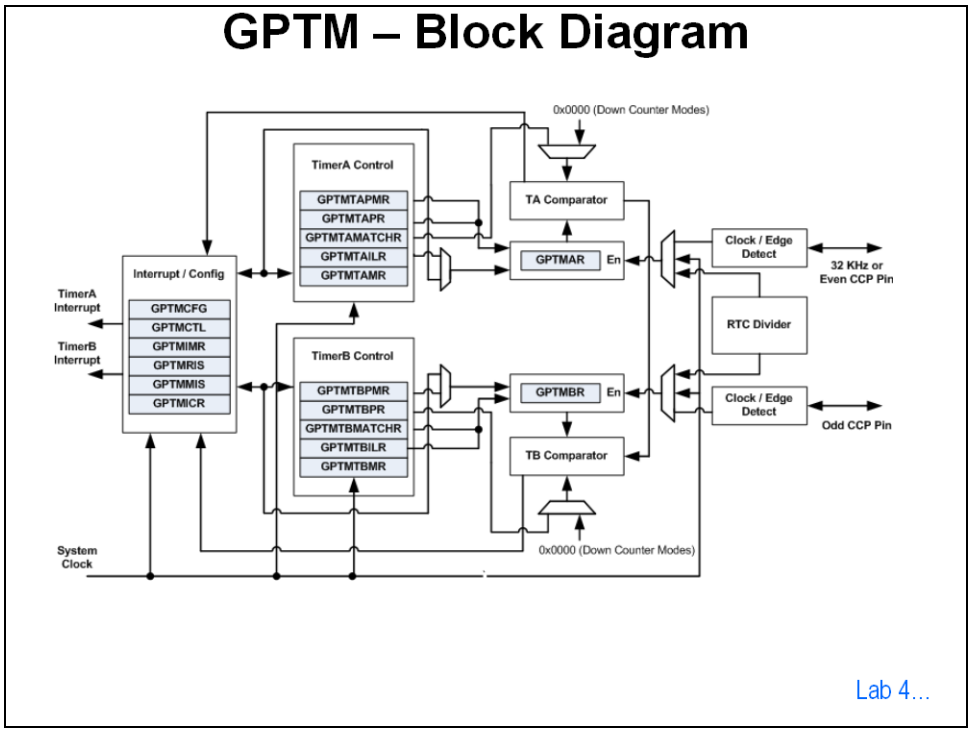
GPTM Modes...

GPTM - Modes

- 16 and 32-bit Timer modes
  - Programmable one-shot
  - Programmable periodic
  - Real-Time Clock when external 32768 Hz clock is the input
  - User-enabled stalling when controller asserts CPU Halt flag during debug
  - ADC event trigger
- 16-bit Capture modes
  - Input edge count capture
  - Input edge time capture
- 16-bit PWM mode
  - Simple PWM with s/w output inversion

GPTM Block Diagram...

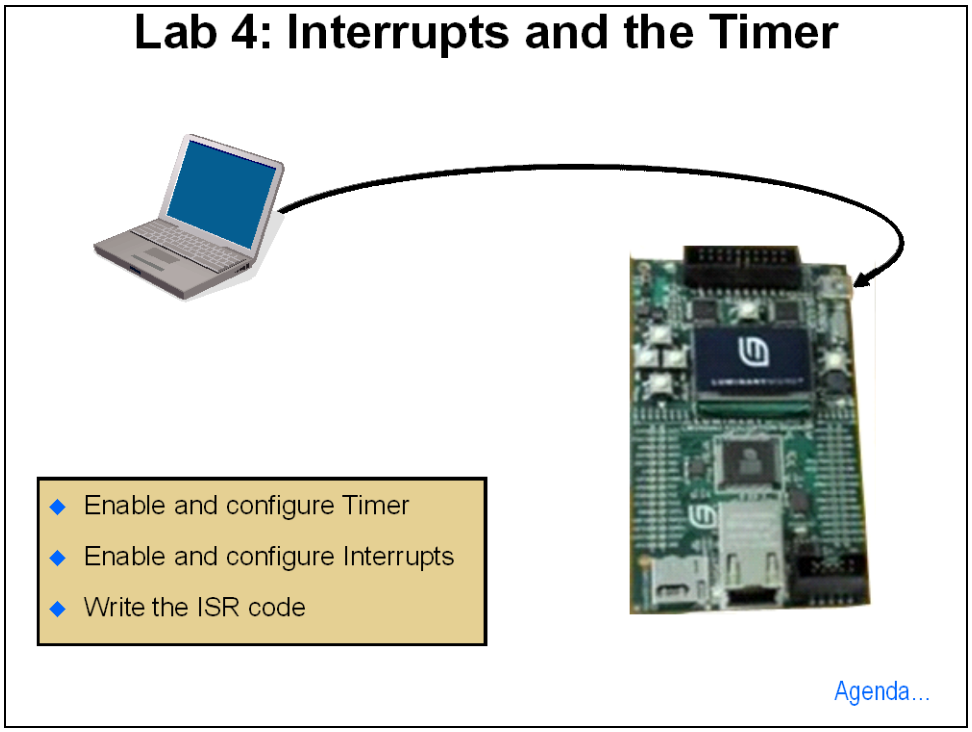
GPTM Block Diagram



Lab 4: Interrupts and the Timer

Objective

In this lab we'll set up the timer to generate interrupts, and then write the code that responds to the interrupt … blinking the LED at 10 Hz.



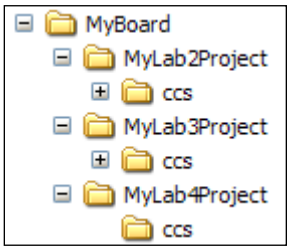
Procedure

Create New Project Folders

1. We need to create some folders to hold Lab 4.

   Open Windows Explorer and navigate to `C:\StellarisWare\boards\MyBoard`. Right-click in the open space of the right-hand pane and select New → Folder. Name the new folder `MyLab4Project` and press the Enter key.

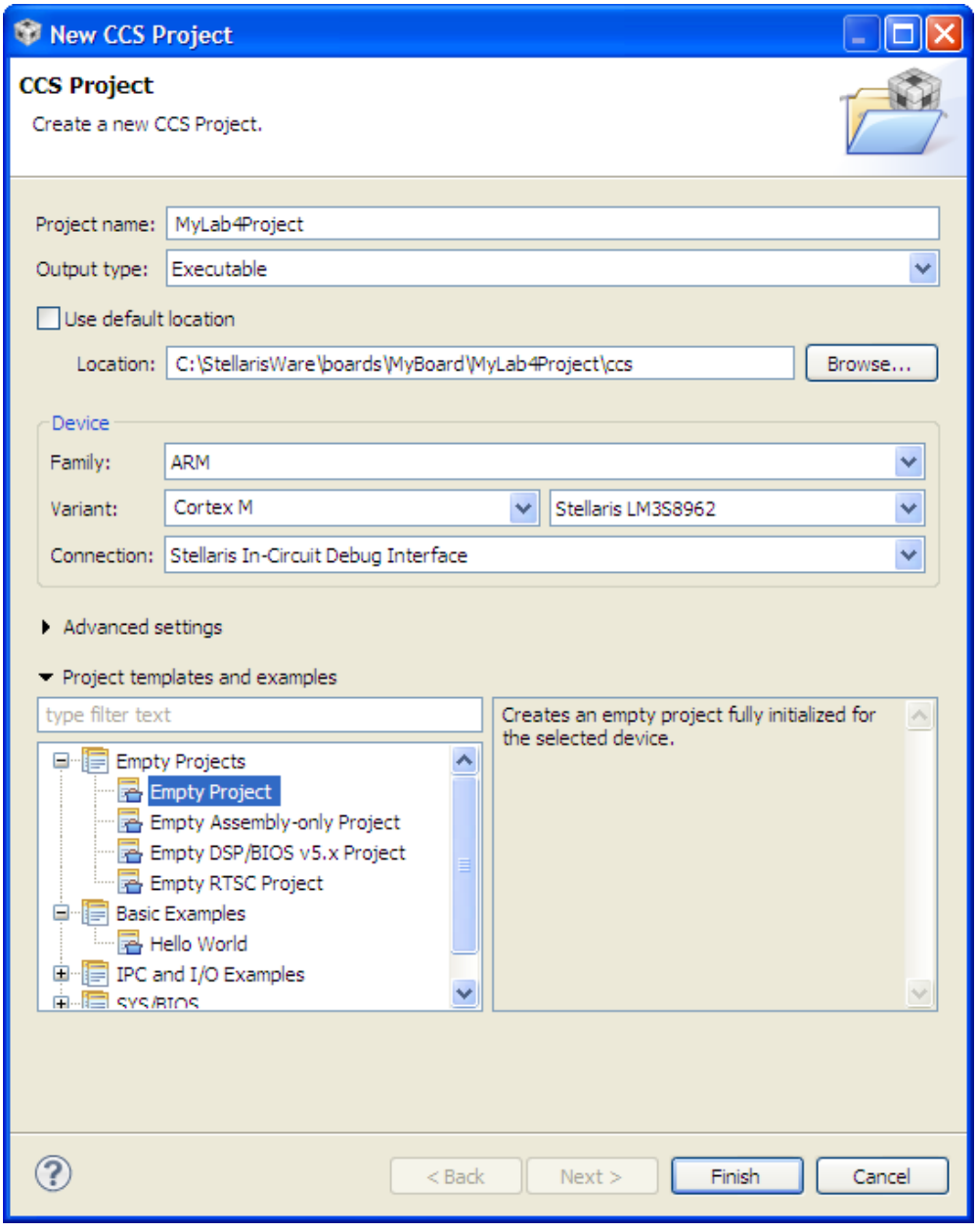
   Double click on `MyLab4Project` to enter the folder and then right-click in the wide open right-hand pane. Select New → Folder. Name the new folder `ccs` and press the Enter key.



.

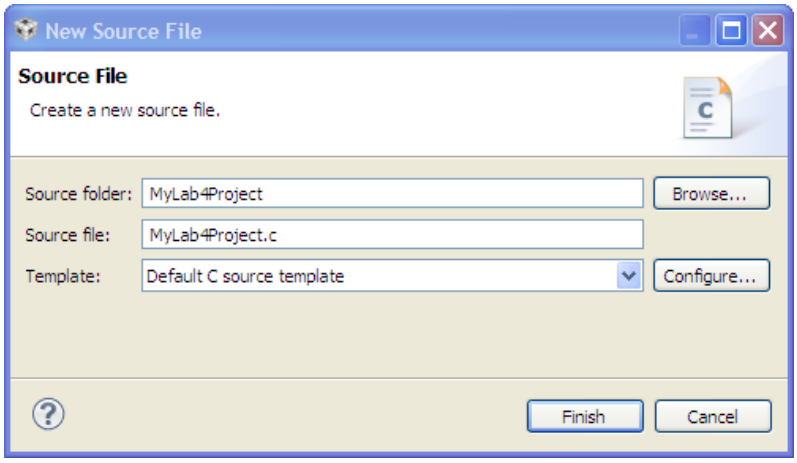
Create `MyLab4Project` Project

2. Maximize Code Composer. On the CCS menu bar select File → New → CCS Project. Make the selections shown below. Make sure to uncheck the "Use default location" checkbox and select the correct path to the "ccs" folder you created. This step is important to making your project portable. Click Finish.

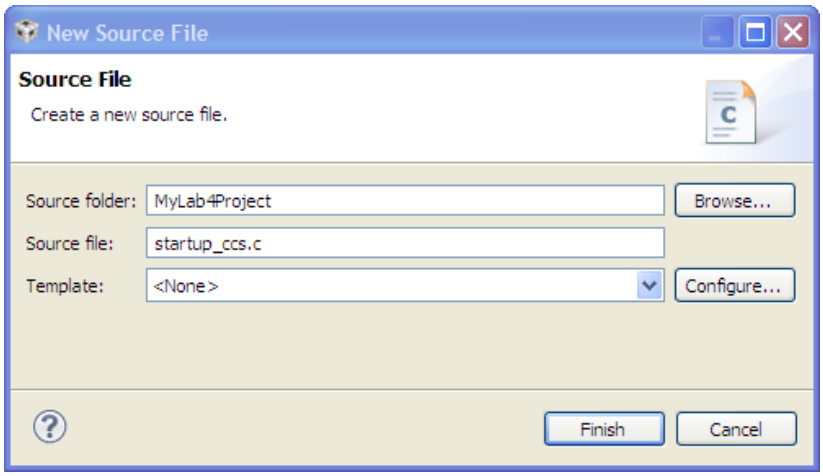


Add Source Files to Project

3. Delete `main.c` from MyLab4Project.
4. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the main C code file and click Finish.



5. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the startup file and click Finish.



Header Files

6. Type (or cut/paste) the following seven lines into `MyLab4Project.c` to include the header files needed to access the StellarisWare APIs :

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
```

`hw_ints.h` : Macros that define the interrupt assignment on Stellaris device (NVIC)

`hw_memmap.h` : Macros defining the memory map of the Stellaris device. This includes defines such as peripheral base address locations such as GPIO_PORTF_BASE

`hw_types.h` : Defines common types and macros such as tBoolean and HWREG(x)

`sysctl.h` : Defines and macros for System Control API of DriverLib. This includes API functions such as SysCtlClockSet and SysCtlClockGet.

`interrupt.h` : Defines and macros for NVIC Controller (Interrupt) API of DriverLib. This includes API functions such as IntEnable and IntPrioritySet.

`gpio.h` : Defines and macros for GPIO API of DriverLib. This includes API functions such as GPIOPinTypePWM and GPIOPinWrite.

`timer.h` : Defines and macros for Timer API of DriverLib. This includes API functions such as TimerConfigure and TimerLoadSet.

Main() Function

7. Next, create your main application function and declare any variables that you will use later in the function. Leave a line for spacing and type the following:

```
int main(void)

{
  unsigned long ulPeriod;

}
```

This variable will be used later in calculating the period and delay needed for a 10 Hz signal.

Clock Setup

8. The system clock defaults to the internal oscillator as its source after reset. This is not a precision internal oscillator. Configure the system clock to run directly from the Main Oscillator at 8 MHz with the following call.

   Leave a blank line for spacing and enter this single line of code inside `main()` after the variable declaration above:

```
SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
```

GPIO Configuration

9. Before calling any peripheral specific DriverLib function, we must enable the clock for that peripheral (RCGCn register). If you fail to do this, it will result in a Fault ISR (address fault). The second statement configures the GPIO where the board's STATUS LED is connected as an output. Leave a line for spacing, then enter these two lines of code inside `main()` after the line in the previous step.

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0);
```

Timer Configuration

10. Again, before calling any peripheral specific DriverLib function we must enable the clock to that peripheral (RCGCn register). If you fail to do this, it will result in a Fault ISR (address fault). The second statement configures Timer 0 as a 32-bit timer in periodic mode. Note that when Timer 0 is configured as 32-bit timer, it combines the two 16-bit timers Timer 0A and Timer 0B. See the General Purpose Timer chapter of the device datasheet for more information. Add a line for spacing and type the following two lines of code after the previous ones:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
```

Period Calculation

11. To toggle a GPIO at 10Hz and a 50% duty cycle, you need to generate an interrupt at ½ of the desired period. First, calculate the number of clocks cycles required for a 10Hz period by calling SysCtlClockGet() and dividing it by your desired frequency. Then divide that by two, since we want a count that is ½ of that for the interrupt.

    This calculated period is then loaded into the Timer's Interval Load register using the TimerLoadSet function of the DriverLib Timer API. Note that you have to subtract one from the timer period since this value is directly loaded into the timer interval load register. In periodic mode, it takes one extra clock to "reload" the value.

    Add a line for spacing and type the following two lines of code after the previous ones:

```
ulPeriod = (SysCtlClockGet() / 10) / 2;
TimerLoadSet(TIMER0_BASE, TIMER_A, ulPeriod -1);
```

Interrupt Enable

12. Next, we have to enable the interrupt … not only in the timer module, but also in the NVIC (Nested Vector interrupt controller, Cortex M3's interrupt controller). IntMasterEnable is the master interrupt enable for the interrupts. IntEnable enables the specific vector associated with the Timer. TimerIntEnable, enables a specific event within the timer to generate an interrupt. In this case we are enabling an interrupt to be generated on a timeout of Timer 0A. Add a line for spacing and type the following three lines of code after the previous ones:

```
IntMasterEnable();
IntEnable(INT_TIMER0A);
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
```

Timer Enable

13. Finally we can enable the timer. This will start the timer and interrupts will begin triggering on the timeouts. Add a line for spacing and type the following line of code after the previous ones:

```
TimerEnable(TIMER0_BASE, TIMER_A);
```

Main Loop

14. The main processing loop of the code is simply an empty while(1) since the toggling of the GPIO will happen in the interrupt routine. Add a line for spacing and type the following lines of code after the previous ones:

```
while(1)
{
}
```

Timer Interrupt Handler

15. Since this application is interrupt driven, we must add an interrupt handler for the Timer. In the interrupt handler, we must first clear the interrupt source and then toggle the GPIO pin based on the current state. Add a line for spacing and type the following lines of code after the final closing brace of the program.

```
void Timer0IntHandler(void)

{

      // Clear the timer interrupt

      TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);


      // Read the current state of the GPIO pin and
      // write back the opposite state

      if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0))

      {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0);
      }

      else

      {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 1);
      }

}
```

16. Click the Save button to save your work. Your code should look something like this:

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"

int main(void)
{
  unsigned long ulPeriod;

  SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);

  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0);

  SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
  TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);

  ulPeriod = (SysCtlClockGet() / 10) / 2;
  TimerLoadSet(TIMER0_BASE, TIMER_A, ulPeriod -1);

  IntMasterEnable();
  IntEnable(INT_TIMER0A);
  TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

  TimerEnable(TIMER0_BASE, TIMER_A);

  while(1)
    {
    }
}

void Timer0IntHandler(void)
{

  // Clear the timer interrupt
  TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

  // Read the current state of the GPIO pin and
  // write back the opposite state
  if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0))
  {
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0);
  }
  else
  {
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 1);
  }
}
```

Startup Code

17. In addition to the main file you have created, you will also need a startup file specific to the tool chain you are using. This file contains the vector table, startup routines to copy initialized data to RAM, clears the bss section and contains the default fault routine ISRs. You can easily copy the "startup_ccs.c" file from the hello or blinky example and use this file as a starting template. You should note that the default fault routines in the start up code are endless while() loops. This may not be the behavior you want in a production system.

    Right-click on MyLab3Project in the Project Explorer pane and select Open Project. Double-click on startup_ccs.c. When the editor window opens, copy the contents of the file into your empty MyLab4Project startup_ccs.c file. Close the MyLab3Project and make sure your MyLab4Project is the Active project.

    You need to find the appropriate vector position and replace `IntDefaultHandler` with the name of your Interrupt handler. In this case you will add `Timer0IntHandler` to the position with the comment "Timer 0 subtimer A" as shown below:

```
  IntDefaultHandler,                      // ADC Sequence 2
  IntDefaultHandler,                      // ADC Sequence 3
  IntDefaultHandler,                      // Watchdog timer
  Timer0IntHandler,                       // Timer 0 subtimer A
  IntDefaultHandler,                      // Timer 0 subtimer B
  IntDefaultHandler,                      // Timer 1 subtimer A
```

    You will also need to declare this function at the top of this file as external. This is necessary for the compiler to resolve this symbol. Find

```
extern void _c_int00(void);
```

    and add the

```
extern void Timer0IntHandler(void);
```

    declaration as shown below:

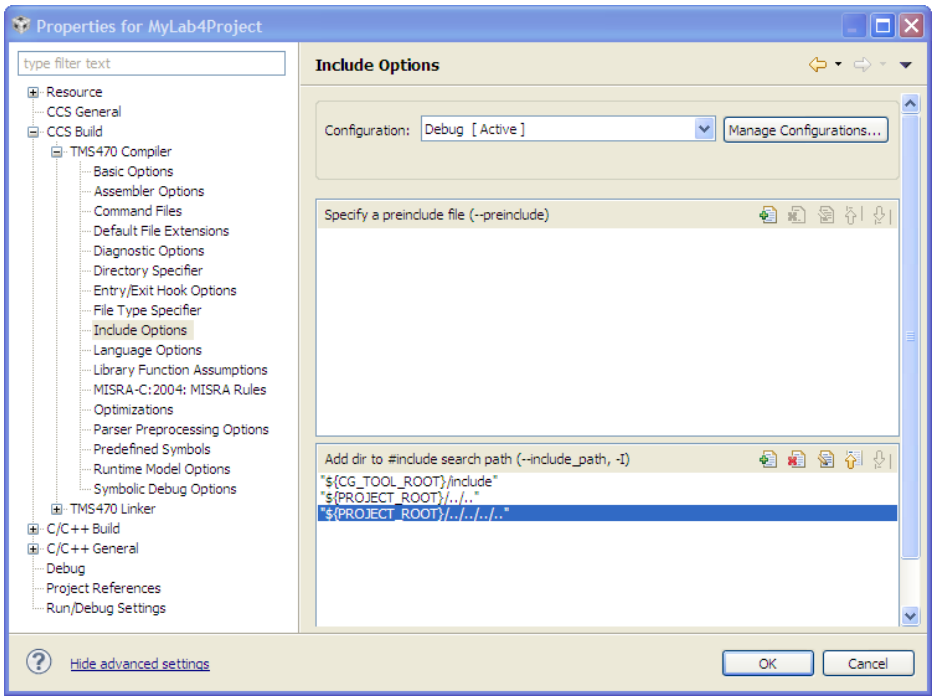
```
37 // External declaration for the reset handler that is to be called when the
38 // processor is started
39 //
40 //*****************************************************************************
41 extern void _c_int00(void);
42 extern void Timer0IntHandler(void);
43
44 //*****************************************************************************
```

    Click the Save button.

Set the Build Options

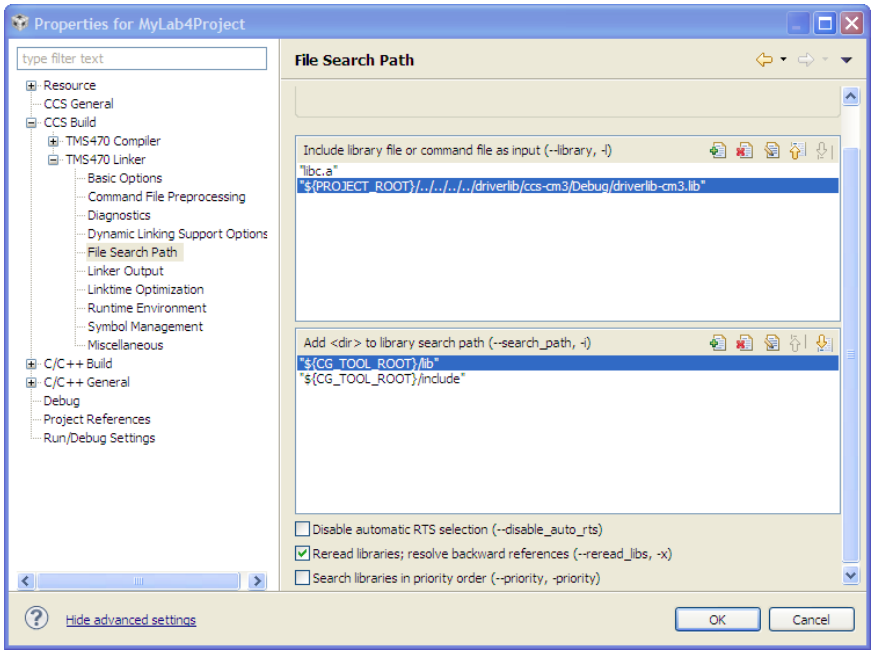
18. Right-click on MyLab4Project in the Project Explorer pane and select Properties. Click the + left of TMS470 Compiler and click on Include Options. In the bottom, include search path pane, click the Add button  and, one at a time, add the following two include search paths.

    ${PROJECT_ROOT}/../..
    ${PROJECT_ROOT}/../../../..



19. Click **File Search Path** under **TMS470 Linker**. Add the following include library file to the top window:

${PROJECT_ROOT}/../../../../driverlib/ccs-cm3/Debug/driverlib-cm3.lib



Click OK to save your changes.

Run The Code

20. Compile and download your application by clicking the Debug button on the menu bar. If you have any issues, correct them, and then click the Debug button again. After a successful build, the CCS Debug perspective will appear.

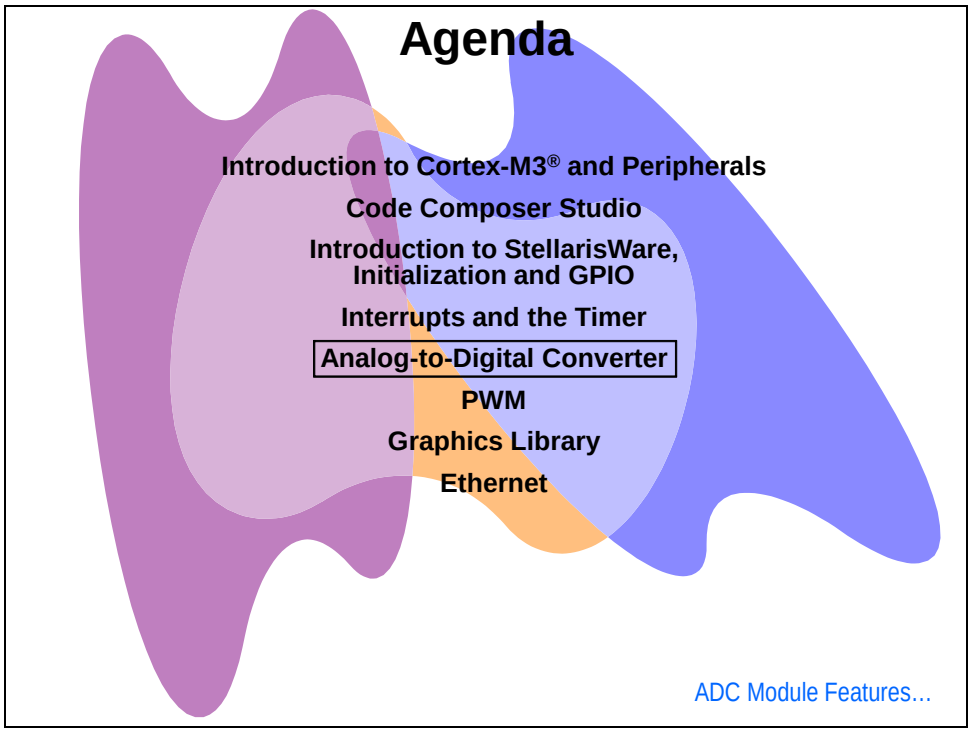
    Click the Run button to run the program that was downloaded to the flash memory of your device. The program will generate a 10 Hz PWM type signal on Pin 0 of GPIO Port F. This pin is connected to the STATUS LED on the EK-LM3S8962.

    Click the Terminate button to return to the Editing perspective, close the MyLab4Project project and minimize Code Composer Studio.

You're done.

Introduction

This module will introduce you to the use of the ADC module on Stellaris devices. In the lab we'll write the code to utilize the ADC and the sequencer to measure the value of the internal temperature sensor.



Module Topics

ADC Module Features

Stellaris ADC Module Features

The Stellaris ADC module features:

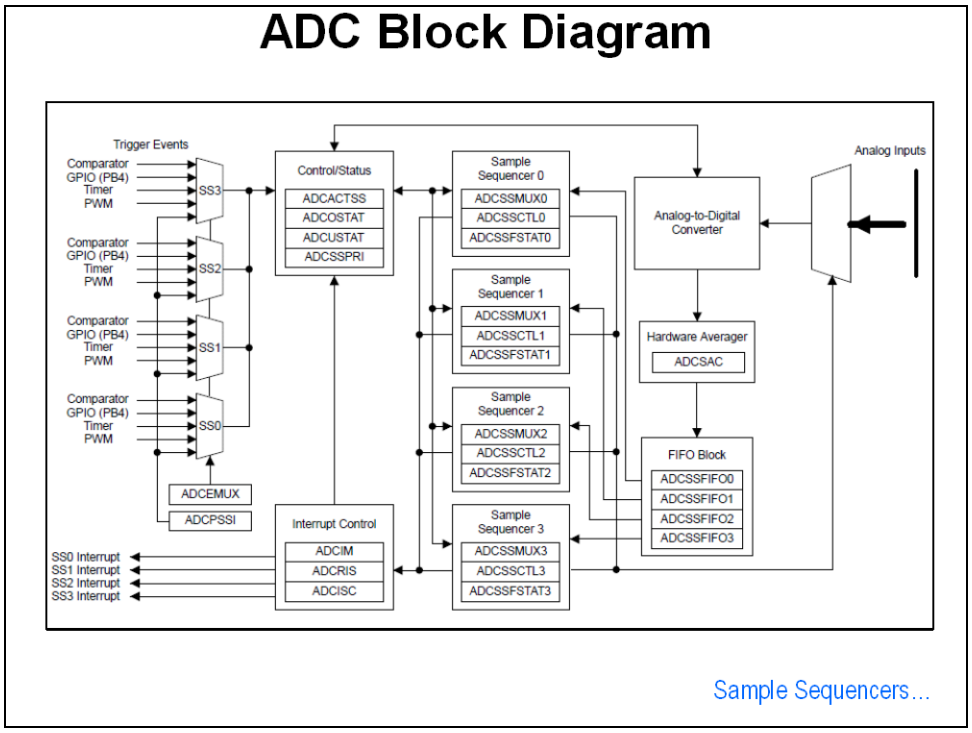
- 10-bit, 500KSPS resolution
- 4 input channels plus an internal temperature sensor
  - Single or Differential inputs
- 4 programmable sequencers
  - Configurable inputs sources
  - Trigger events
  - Interrupt generation
  - Sequence priority
  - Variable FIFO depth from 1 to 8 samples

Stellaris ADC Module Features (cont)

- Flexible trigger control
  - Software
  - Timers
  - Analog Comparators
  - PWM
  - GPIO
- Hardware averaging of up to 64 samples
- 3V internal reference
- Separate analog power and ground for greater noise immunity

ADC Block Diagram…

ADC Block Diagram



Sample Sequencers...

Sample Sequencers

Sample Sequencers

Sequencer	Number of Samples	Depth of FIFO
SS3	1	1
SS2	4	4
SS1	4	4
SS0	8	8

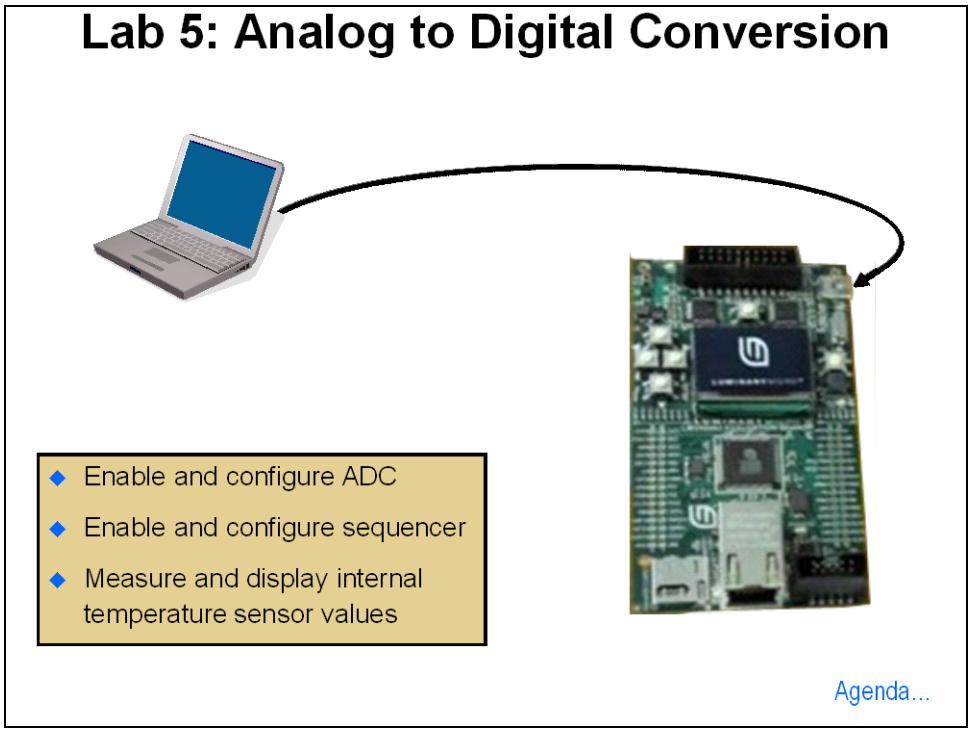
All of the sequencers are identical in implementation except for the number of samples that can be captured and the depth of the FIFO. Each FIFO entry is a 32-bit word, with the lower 10 bits containing the conversion result.

Lab 5...

Lab 5: Analog to Digital Converter

Objective

In this lab we'll set up the ADC to measure the value of the on-chip temperature sensor. We'll calculate degrees Celsius and Fahrenheit and display them in a watch window.



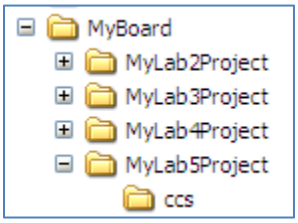
Procedure

Create New Project Folders

1. We need to create some folders to hold Lab 5.

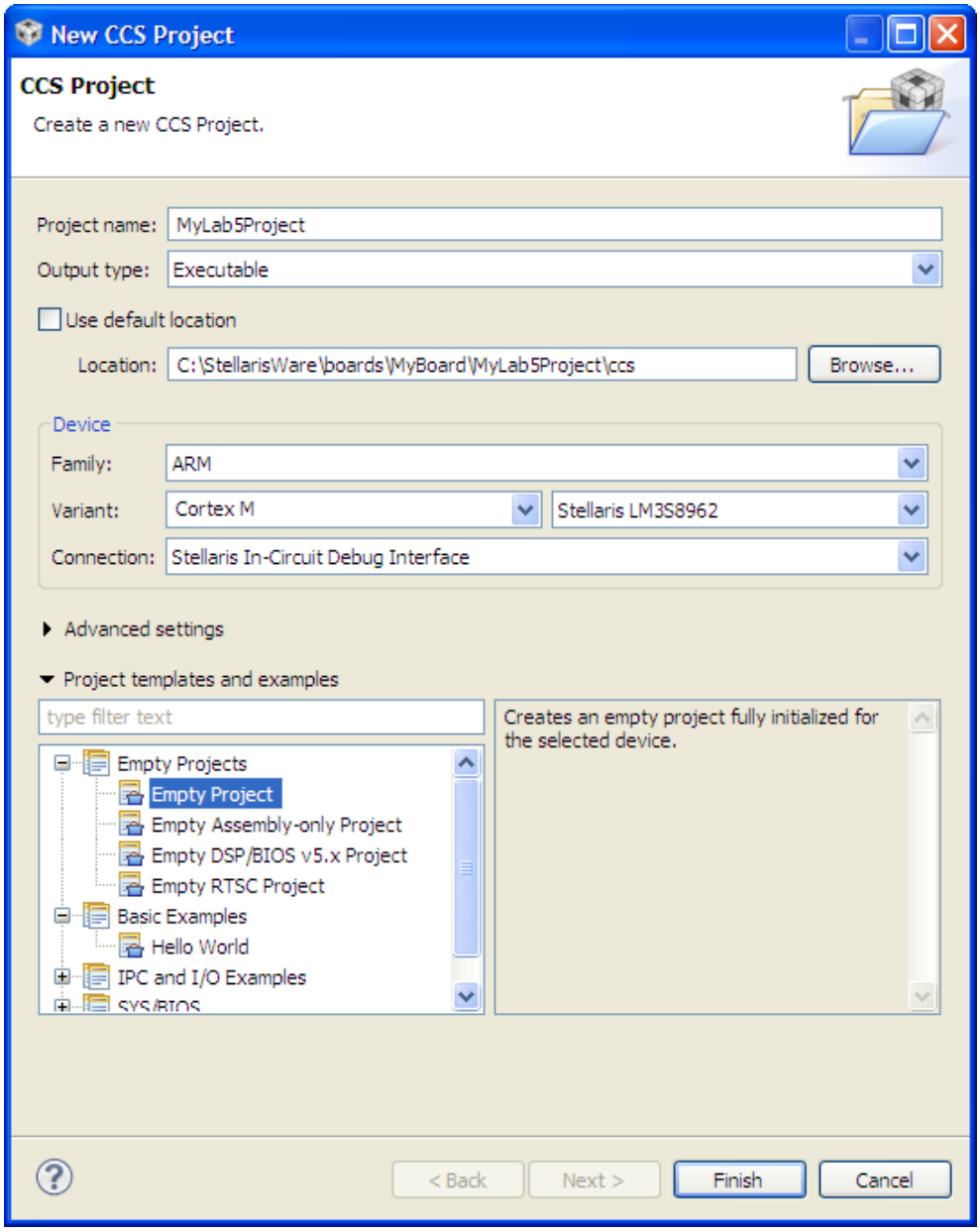
   Open Windows Explorer and navigate to `C:\StellarisWare\boards\MyBoard`. Right-click in the open space of the right-hand pane and select New → Folder. Name the new folder `MyLab5Project` and press the Enter key.

   Double click on `MyLab5Project` to enter the folder and then right-click in the wide open right-hand pane. Select New → Folder. Name the new folder `ccs` and press the Enter key.



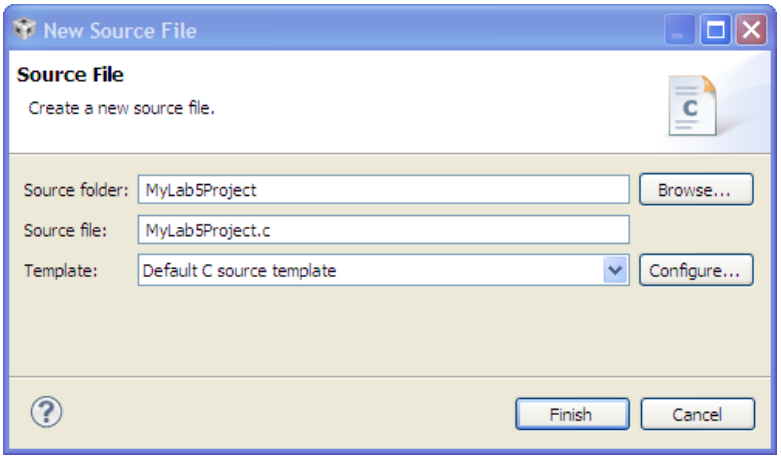
Create `MyLab5Project` Project

2. Maximize Code Composer. On the CCS menu bar select File → New → CCS Project. Make the selections shown below. Make sure to uncheck the "Use default location" checkbox and select the correct path to the "ccs" folder you created. This step is important to making your project portable. Click Finish.

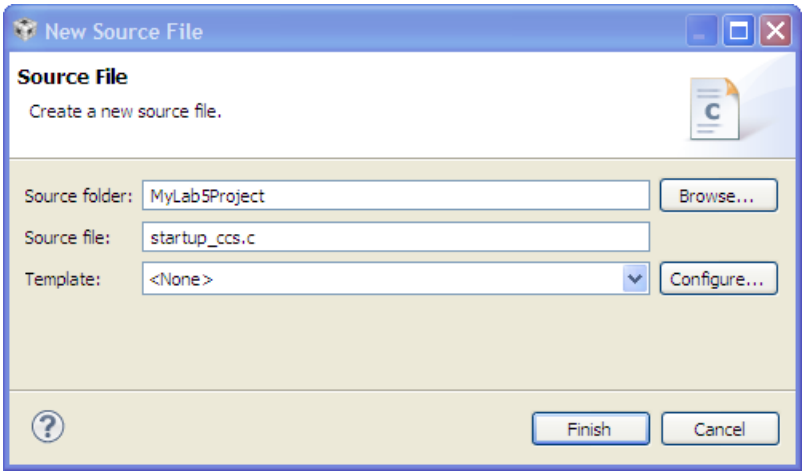


Add Source Files to Project

3. Delete `main.c` from MyLab5Project. Click on MyLab5Project to make it active.
4. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the main C code file and click Finish.



5. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the startup file and click Finish.



6. Copy the contents of startup_ccs.c in MyLab3Project (**not** MyLab4Project) into your blank startup_ccs.c file. Click Save.

Includes and Defines

7. Add (or copy/paste) the following 5 lines to the top of `MyLab5Project.c` :

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
```

`adc.h:` headers for using the ADC driver

Driver Library Error Routine

8. During the debug process, you may find that you have called a driver library API with incorrect parameters or a library function generates an error for some other reason. The following code will be called if the driver library encounters such an error.

   Leave a blank line for spacing and enter these lines of code after the lines above:

```
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif
```

Main()

9. Set up the main() routine by typing in the three lines below:

```
int main(void)
{
}
```

10. The following definition will create an array that will be used for storing the data read from the ADC FIFO. It must be as large as the FIFO for the sequencer in use. We will be using sequencer 1 which has a FIFO depth of 4. If another sequencer was used with a smaller or deeper FIFO, then the array size would have to be changed. For instance, sequencer 0 has a depth of 8. Add the following line of code as your first line in **`main()`** .

```
unsigned long ulADC0Value[4];
```

11. We'll need some variables for calculating the temperature from the sensor data. The first variable is for storing the average of the temperature. The remaining variables are used to store the temperature values for Celsius and Fahrenheit. All are declared as 'volatile' so that each variable will not be optimized out by the compiler and will be available to the 'Watch' or 'Local' window(s) at run-time. Leave a line for spacing and add these lines after that last line:

```
volatile unsigned long ulTempAvg;
volatile unsigned long ulTempValueC;
volatile unsigned long ulTempValueF;
```

12. The ADC must be clocked from the PLL or directly from a 14MHz - 18MHz clock source to operate properly. While the ADC works in a 14-18 MHz range, to maintain a 1MSPS sampling rate, the ADC must be provided a 16-MHz clock source (for 1MSPS supported devices. See the device specific datasheet for ADC speed specifications). Set the clocking to run at 20 MHz (200 MHz / 10) using the PLL. Then the SYSCTL_XTAL value must be changed to match the value of the crystal on the board. The EK-LM3S8962 has an 8.0MHz crystal and has a maximum ADC speed of 500ksps. Add a line for spacing and add this line after the last one:

```
SysCtlClockSet(SYSCTL_SYSDIV_10 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
```

13. Let's enable the ADC0 module next. Add a line for spacing and add this line after the last one:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
```

14. As an example, let's set ADC sample rate to 250 kilo-samples per second (since we're measuring temperature, a slower speed would be fine, but let's go with this). The `SysCtlADCSpeedSet()` API can also set the sample rate to additional device specific speeds. These speeds are noted in the device datasheet in the ADC section. In the case of the LM3S8962 the following additional speeds are supported: SYSCTL_ADCSPEED_500KSPS or SYSCTL_ADCSPEED_125KSPS. Add the following line directly after the last one:

```
SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS);
```

15. Before we configure the ADC sequencer settings, we should disable ADC sequencer 1. Add this line after the last one:

```
ADCSequenceDisable(ADC0_BASE, 1);
```

16. The first step in programming the ADC sequencer is to set the ADC to trigger after a processor event for sequencer 1 with priority level 0, highest priority. All sequencers can have a unique priority level. By default sequencer 0 has priority 0 (highest) and sequencer 3 has priority 3 (lowest). Add a line for spacing and add this line of code:

```
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
```

17. Next we need to configure all four steps in the ADC sequencer. Configure steps 0 - 2 on sequencer 1 to sample the temperature sensor (ADC_CTL_TS). In this example, we will be averaging all four steps of temperature sensor data on sequencer 1 to calculate the temperature, so all four sequencer steps will measure the temperature sensor. For more information on the ADC sequencers and steps, reference the device specific datasheet. Add the following three lines after the last:

```
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
```

18. The final sequencer step requires a couple of extra settings. Sample the temperature sensor (ADC_CTL_TS) and configure the interrupt flag (ADC_CTL_IE) to be set when the sample is done. Tell the ADC logic that this is the last conversion on sequencer 1 (ADC_CTL_END). Add this line directly after the last ones:

```
ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
```

19. Now we can re-enable the ADC sequencer 1. Add this line directly after the last one:

```
ADCSequenceEnable(ADC0_BASE, 1);
```

20. Still within main(), add a while loop to your code. Add a line for spacing and enter these three lines of code:

```
while(1)
{
}
```

21. Save your work. As a sanity-check, right-click on MyLab5Project.c in the Project pane and select Build Selected File(s). You should have one fatal error; that **hw_memmap.h** can't be opened. That's okay because we haven't set the build properties yet. If you are having issues, check the code on the following page:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"

#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

int main(void)
{
    unsigned long ulADC0Value[4];

    volatile unsigned long ulTempAvg;
    volatile unsigned long ulTempValueC;
    volatile unsigned long ulTempValueF;

    SysCtlClockSet(SYSCTL_SYSDIV_10 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS);
    ADCSequenceDisable(ADC0_BASE, 1);

    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);

    while(1)
    {
    }
}
```

I'm sorry about the text sizing, but that's the only way I could fit the code on a single line.

Inside the while(1) Loop

Inside the while(1) we're going to read the value of the temperature sensor endlessly.

22. The indication that the ADC conversion is complete will be the ADC interrupt status flag. It's good programming practice to make sure that the flag is cleared. Add the following line as your first line of code inside the while(1) loop:

    ```
    ADCIntClear(ADC0_BASE, 1);
    ```

23. Then we can trigger the ADC conversion. Add the following line after the last:

    ```
    ADCProcessorTrigger(ADC0_BASE, 1);
    ```

24. Then we need to wait for the conversion to complete. Obviously, a better way to do this would be to use an interrupt, rather than burn CPU cycles waiting, but that exercise is left for the student. Add a line for spacing and add the following three lines of code:

    ```
    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
    }
    ```

25. Now that the conversion is complete, we can read the ADC value from the ADC_SSFFIO1. The function we'll be using copies data from the specified sample sequencer output FIFO to a memory resident buffer. The number of samples available in the hardware FIFO are copied into the buffer, which must be large enough to hold that many samples. This will only return the samples that are presently available, which might not be the entire sample sequencer if it is in the process of being executed. Add a line for spacing and add the following line after the while loop in step 23:

    ```
    ADCSequenceDataGet(ADC0_BASE, 1, ulADC0Value);
    ```

26. Calculate the average of the temperature sensor data. The addition of 2 is for rounding. Since 2/4 = 1/2 = 0.5, 1.5 will be rounded to 2.0 with the addition of 0.5 and in the case of 1.0, when 0.5 is added to yield 1.5, this will be rounded back down to 1.0 due to the rules of integer math. Add this line after the last on a single line:

    ```
    ulTempAvg = (ulADC0Value[0] + ulADC0Value[1] + ulADC0Value[2] +
    ulADC0Value[3] + 2)/4;
    ```

27. Now that we have the averaged reading from the temperature sensor, we can calculate the Celsius value of the temperature. The equation below is the result of a combination of the Internal Temperature Sensor Characteristic mentioned in the device datasheet and the voltage representation of the sampled temperature sensor data based on a 3.0V reference. Division is performed last to avoid truncation due to integer math rules.

    Example:

    Let: A = ulTempAvg
    B = Senso (The voltage at the output terminal)
    C = ulTempValueC

    For: B = (A/1023)3.0V (1)
    B = 2.7 - [(C + 55)/75] (2)

    Then: C = ({2.7 - [(A/1023)3.0V]}75) - 55 (3)
    = {2.7*75 - [3.0V*75(A/1023]} – 55
    = {27*75/10 - [225(A/1023)]} – 55
    [2025*1023 - (225*10*A)] - 55*10*1023 = C*10*1023
    (2025 - 550)1023 - 2250*A = C*10230

    C = [(1475 * 1023) - (2250 * A)] / 10230 (4)

    Enter the following line of code directly after the last:

    ```
    ulTempValueC = ((1475 * 1023) - (2250 * ulTempAvg)) / 10230;
    ```

28. Once you have the Celsius temperature, calculating the Fahrenheit temperature is easy. Hold the division until the end to avoid truncation.

    Let: C = ulTempValueC
    F = ulTempValueF

    For: F = (C*9/5)+32 (5)

    Then: F = [(C*9) + 32*5] / 5
    F = [(C*9) + 160] /5 (6)

    Enter the following line of code directly after the last:

    ```
    ulTempValueF = ((ulTempValueC * 9) + 160) / 5;
    ```

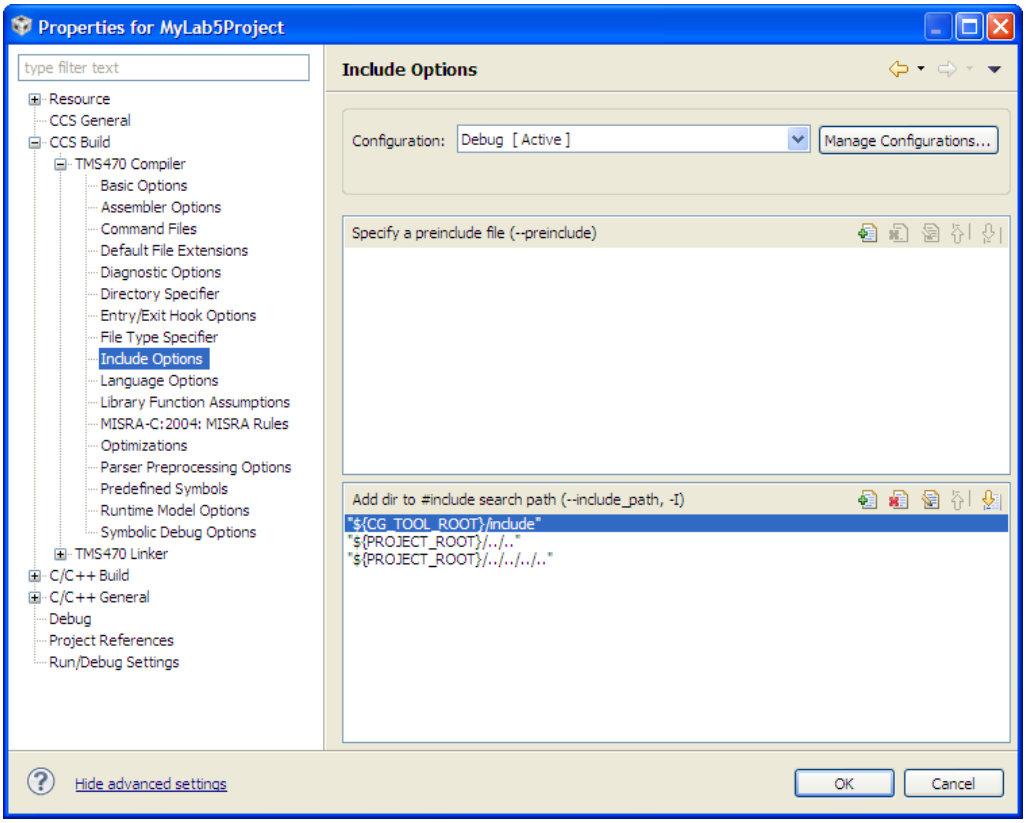
29. Save your work and compare it with our code below:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"

#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

int main(void)
{
    unsigned long ulADC0Value[4];

    volatile unsigned long ulTempAvg;
    volatile unsigned long ulTempValueC;
    volatile unsigned long ulTempValueF;

    SysCtlClockSet(SYSCTL_SYSDIV_10 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS);
    ADCSequenceDisable(ADC0_BASE, 1);

    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);

    while(1)
    {
     ADCIntClear(ADC0_BASE, 1);
     ADCProcessorTrigger(ADC0_BASE, 1);

     while(!ADCIntStatus(ADC0_BASE, 1, false))
       {
       }

     ADCSequenceDataGet(ADC0_BASE, 1, ulADC0Value);
     ulTempAvg = (ulADC0Value[0] + ulADC0Value[1] + ulADC0Value[2] + ulADC0Value[3] + 2)/4;
     ulTempValueC = ((1475 * 1023) - (2250 * ulTempAvg)) / 10230;
     ulTempValueF = ((ulTempValueC * 9) + 160) / 5;
     }
}
```

Set the Build Options

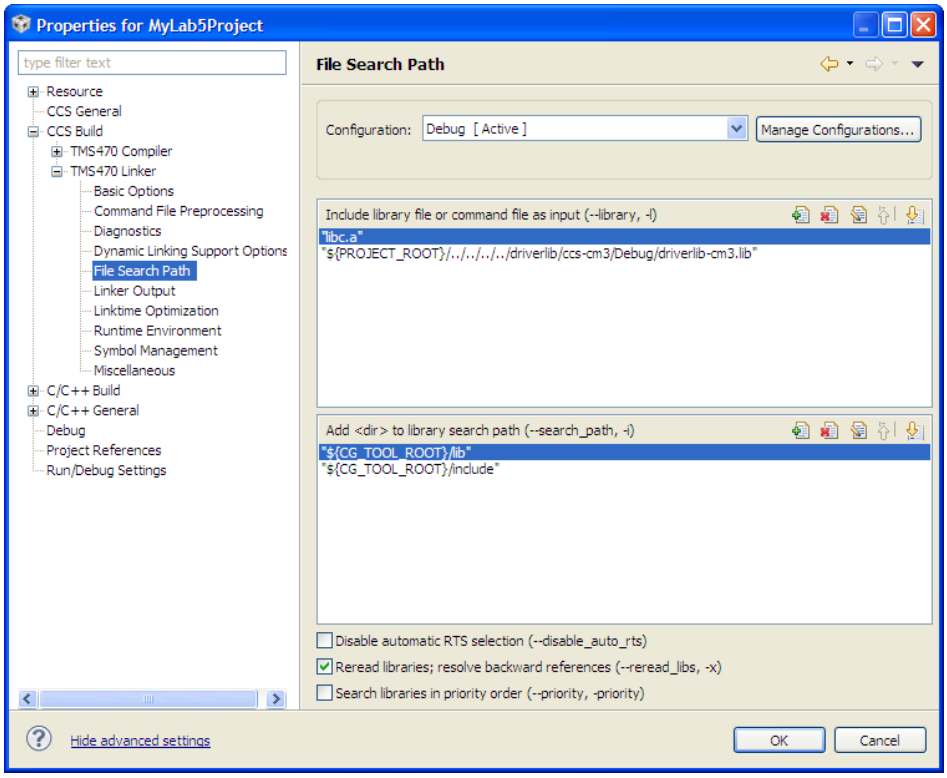
30. Right-click on MyLab5Project in the Project Explorer pane and select Properties. Click the + left of TMS470 Compiler and click on Include Options. In the bottom, include search path pane, click the Add button and, one at a time, add the following two include search paths.

${PROJECT_ROOT}/../..
${PROJECT_ROOT}/../../../..



31. Click **File Search Path** under **TMS470 Linker**. Add the following include library file to the top window:

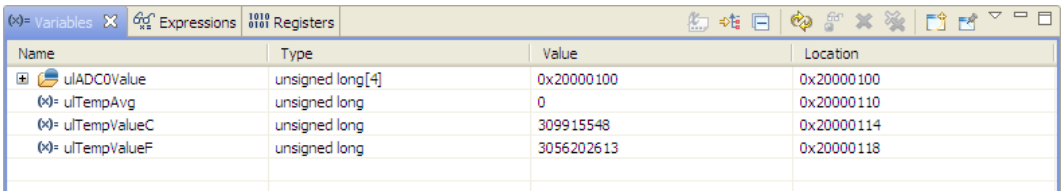
${PROJECT_ROOT}/../../../../driverlib/ccs-cm3/Debug/driverlib-cm3.lib

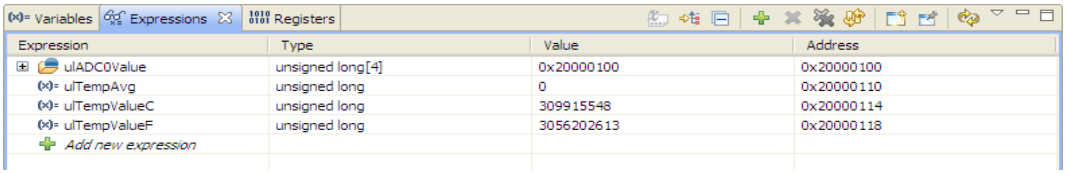


Click OK to save your changes.

Build and Run the Code

32. Compile and download your application by clicking the Debug button  on the menu bar. If you have any issues, correct them, and then click the Debug button again. After a successful build, the CCS Debug perspective will appear.
33. You should see the local variables displayed in the Variables display pane (upper right); **`ulADC0Value, ulTempAvg, ulTempValueC and ulTempValueF`**. If you do not see these, you can add them in a watch expression. Find the variables in the last four lines of code, highlight a variable, then right-click on it and select Add Watch Expression for all four variables. Both the Variables and Expressions windows are shown below:



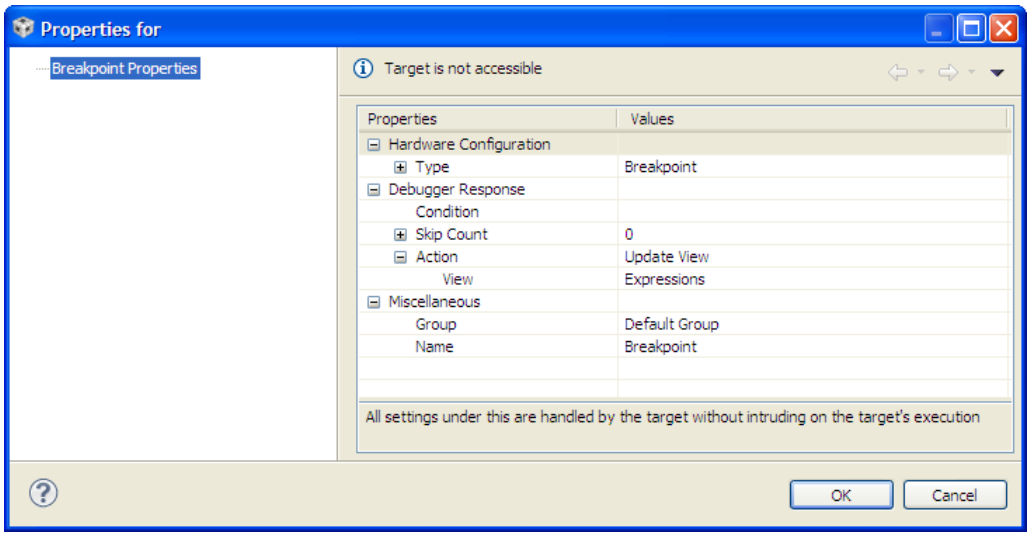


34. We'd like to set the debugger up so that it will update the Local and Watch windows each time the code runs. Since there is no line of code after the calculations, we'll choose one right before them and display the last calculation.

    Click on the first line of code in the while(1) loop;
    `ADCIntClear(ADC0_BASE, 1);` and then right-click on it. Select Breakpoint (Code Composer Studio) then Breakpoint to set a breakpoint on this line.

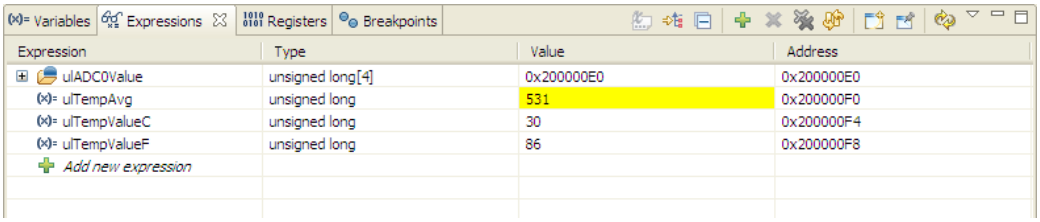
```
35      while(1)
36      {
37          ADCIntClear(ADC0_BASE, 1);
38          ADCProcessorTrigger(ADC0_BASE, 1);
39
```

    Right-click on the breakpoint symbol  and select Breakpoint Properties … Find the line that contains Action and click on the Remain Halted value. That's the normal way a breakpoint should act, but let's change it to Update View (look at the top of the list). In the value below, note that only the Expressions window will be updated. Now the variables in the Expression window will be updated and the code will continue to execute. Click OK.



35. Click the Run button  to run the program.

    You should see the measured value of U1TempAvg toggling over a couple of bits. Changed values from the previous measurement are highlighted in yellow. Use your finger (rub it briskly on your pants), then touch the LM3S-8962 device on the board to warm it. Press your fingers against a cold drink, then touch the LM3S-8962 device to cool it. You should quickly see the results on the display.



    Bear in mind that the temperature sensor is not calibrated, so the values displayed are not exact. That's okay in this experiment, since we're only looking for changes in the measurements.

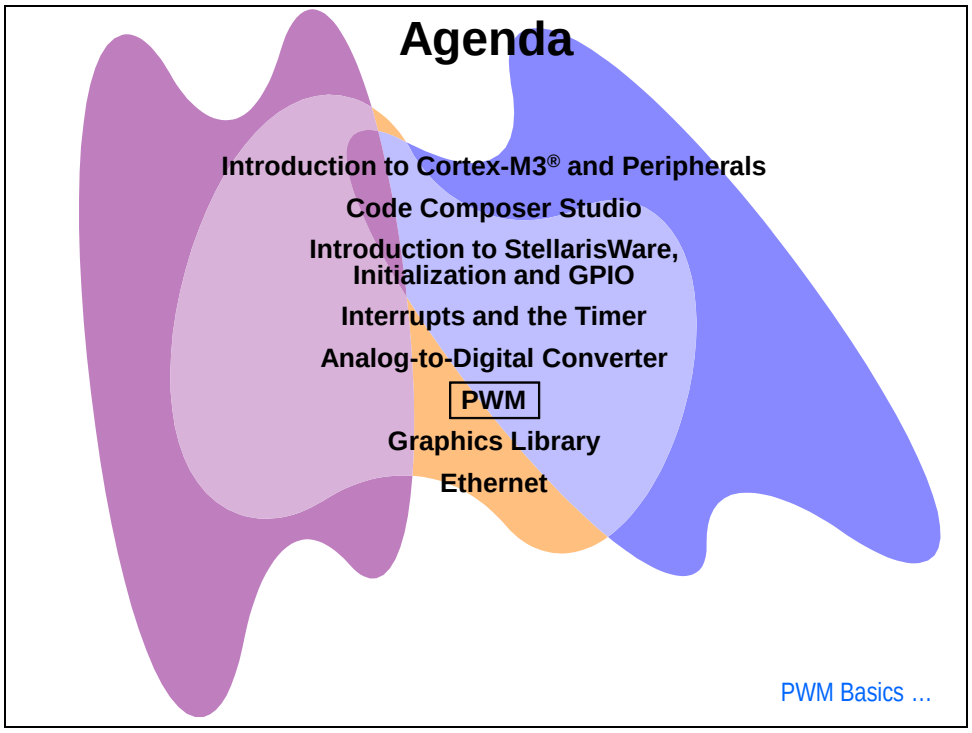
    When you're finished, click the Terminate  button to return to the Edit perspective, close the MyLab5Project project and minimize Code Composer Studio.

You're done.

Pulse Width Modulation

Introduction

This module will introduce you to the use of the PWM module on Stellaris devices. PWM is an extremely useful technique used to control lighting, motors, servos and other positioning devices. The Stellaris PWM implementation is extremely flexible and can be easily programmed to operate independently of CPU control when desired.



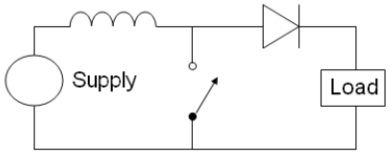
Module Topics

Pulse Width Modulation Basics

Pulse Width Modulation

Pulse Width Modulation (PWM) is a method of encoding analog signal levels. High-resolution digital counters are used to generate a square wave of a given frequency, and the duty cycle of that square wave is modulated to encode the analog signal.

Typical applications for PWM are switching power supplies, motor control, servo positioning and lighting control.



Stellaris PWM...

Stellaris PWM Module

Stellaris PWM Module

The Stellaris PWM module consists of:

- Three PWM generator blocks
- A control block which determines the polarity of the signals and which signals are passed to the pins

Each PWM generator block produces:

- Two independent output signals of the same frequency or
- A pair of complementary signals with dead-band generation (for protection of H-bridge circuits)

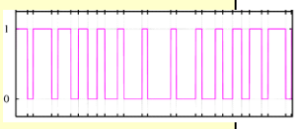
Three generator blocks can produce the full 6 signals of gate control needed for 3-phase inverter bridges

Module Features...

Stellaris PWM Module Features

Stellaris PWM Module Features

- One hardware fault input for low-latency shutdown
- One 16-bit counter
  - Down or Up/Down count modes
  - Output frequency controlled by a 16-bit load value
  - Load value updates can be synchronized
    - Produces output signals at zero and load value
- Two PWM comparators
  - Comparator value updates can be synchronized
  - Produces output signals on match
- PWM generator
  - Output PWM signal is constructed based on actions taken as a result of the counter and PWM comparator output signals
  - Produces two independent PWM signals



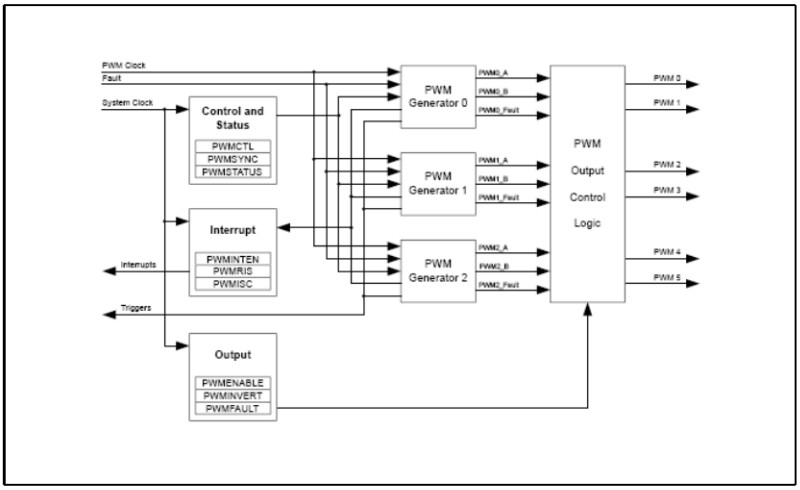
Stellaris PWM Module Features (cont)

- Dead-band generator
  - Produces two PWM signals with programmable dead-band delays suitable for driving a half-H bridge
  - Can be bypassed, leaving input PWM signals unmodified
- Flexible output control block with PWM output enable of each PWM signal
  - PWM output enable of each PWM signal
  - Optional output inversion of each PWM signal (polarity control)
  - Optional fault handling for each PWM signal
  - Synchronization of timers in the PWM generator blocks
  - Synchronization of timer/comparator updates across the PWM generator blocks
  - Interrupt status summary of the PWM generator blocks
- Can initiate an ADC sample sequence

PWM Block Diagram...

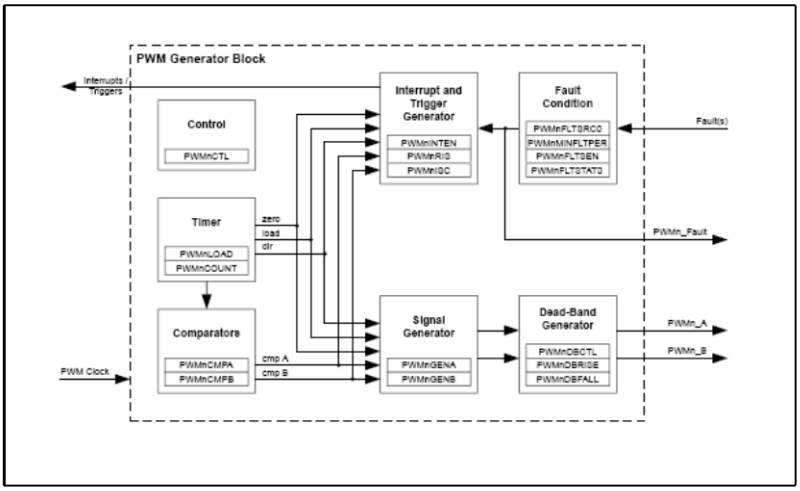
Block Diagrams

PWM Block Diagram



Generator Block Diagram...

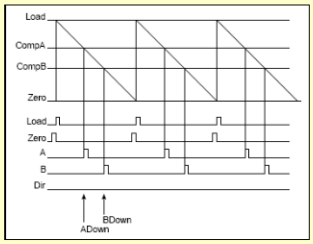
PWM Generator Block



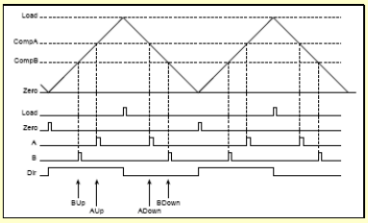
Counting Modes...

Count Modes

PWM Counting Modes



- Count-down mode



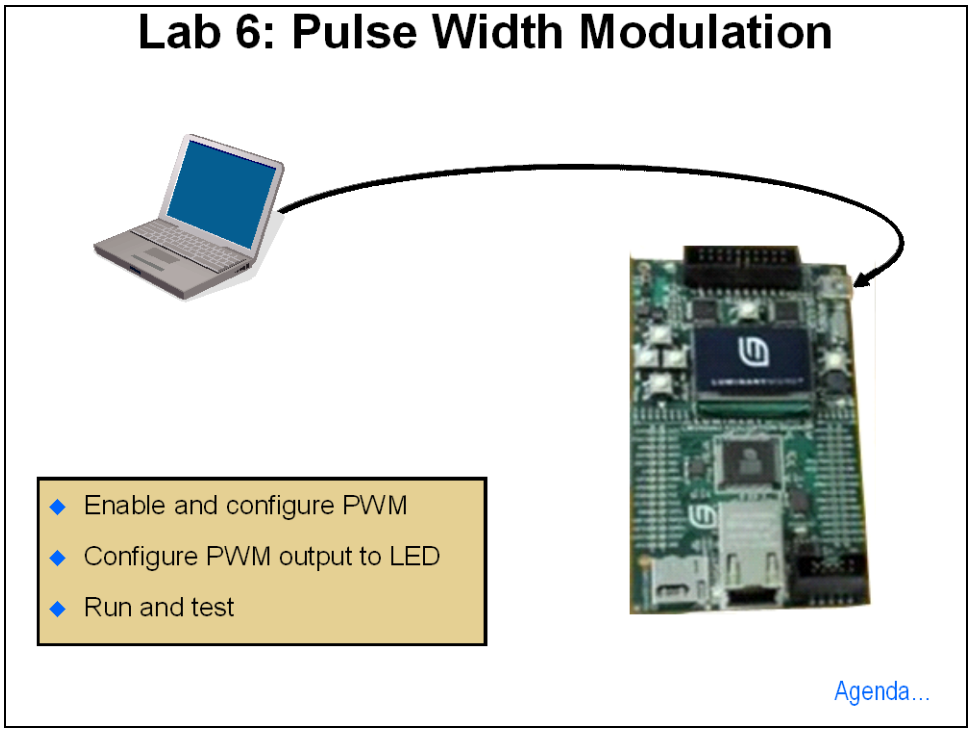
- Count-Up/Down mode

Lab 6...

Lab 6: Pulse Width Modulation

Objective

In this lab we'll set up generator 0 of the PWM module to output a 1Hz, 50% duty cycle signal to the LED.



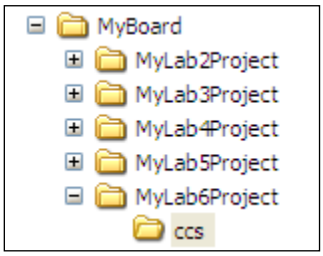
Procedure

Create New Project Folders

1. We need to create some folders to hold Lab 6.

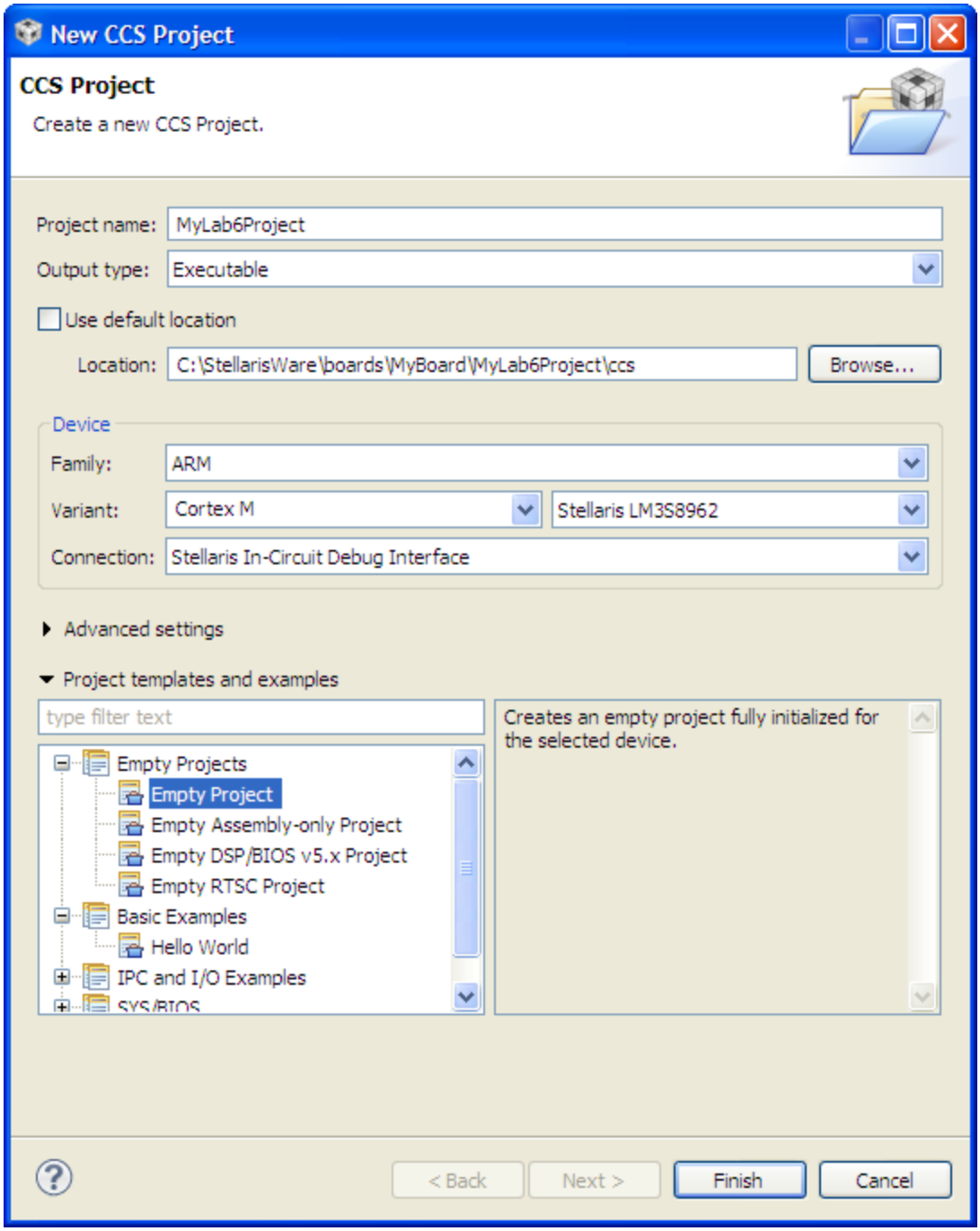
   Open Windows Explorer and navigate to `C:\StellarisWare\boards\MyBoard`. Right-click in the open space of the right-hand pane and select New → Folder. Name the new folder `MyLab6Project` and press the Enter key.

   Double click on `MyLab6Project` to enter the folder and then right-click in the wide open right-hand pane. Select New → Folder. Name the new folder `ccs` and press the Enter key.



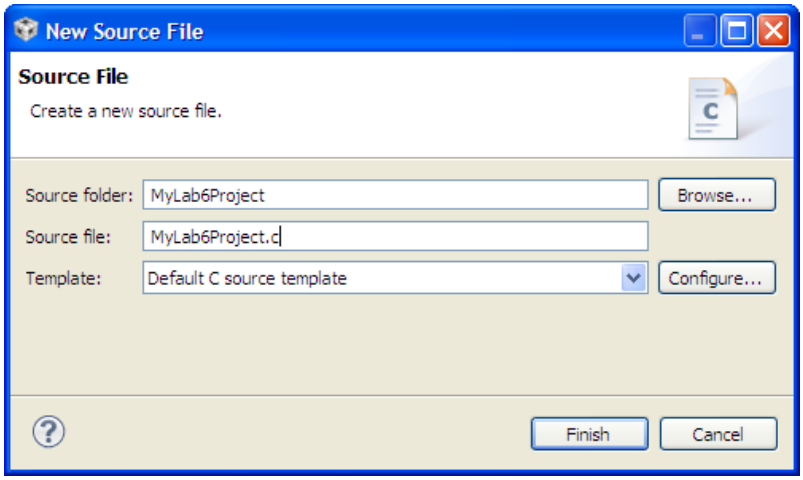
Create `MyLab6Project` Project

2. Maximize Code Composer. On the CCS menu bar select File → New → CCS Project. Make the selections shown below. Make sure to uncheck the "Use default location" checkbox and select the correct path to the "ccs" folder you created. Click Finish.

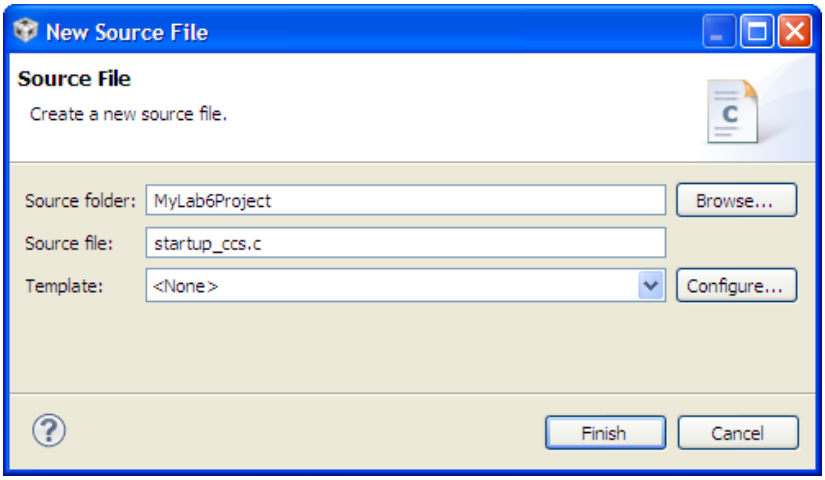


Add Source Files to Project

3. Delete `main.c` from MyLab6Project. Click on MyLab6Project to make it active.
4. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the main C code file and click Finish.



5. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the startup file and click Finish.



6. Copy the contents of startup_ccs.c in MyLab3Project (**not** MyLab4Project) into your blank startup_ccs.c file. Copy the contents of MyLab3Project.c into your MyLab6Project.c. Click Save.

Includes and Defines

7. Delete all the code inside the while(1) loop. Make sure to leave the opening and closing braces.

8. Add (or copy/paste) the following lines to the include area of `MyLab6Project.c` :

```
#include "driverlib/debug.h"
#include "driverlib/pwm.h"
```

`debug.h` : Macros for assisting debug of the driver library.

`pwm.h` : API function prototypes for Pulse Width Modulation (PWM) ports

9. We're going to start out with the LED blinking at 1Hz. Add the following definition right below the includes:

```
#define PWM_FREQUENCY         1
```

Driver Library Error Routine

10. During the debug process, you may find that you have called a driver library API with incorrect parameters or a library function generates an error for some other reason. The following code will be called if the driver library encounters such an error.

    Leave a blank line for spacing and enter these line of codes after the lines above:

```
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif
```

Main()

11. The following variables will be used to program the PWM. They are cast as "volatile" to guarantee that the compiler will not eliminate them, regardless of the optimization setting. **Replace** the first two lines in main() with these:

```
volatile unsigned long ulLoad;
volatile unsigned long ulPWMClock;
```

    Then delete the two lines shown below:

```
ulPeriod = SysCtlClockGet() / 10;
ulDelay = ((ulPeriod / 2) / 3) - 4 ;
```

12. The current clock setting runs the CPU at 8MHz. The PWM module is clocked by the system clock through a divider. That divider has a range of 2 to 64. In order for the LED to flash slowly enough for us to see it, we'll have to run the PWM fairly slowly. Backing into the system clock speed tells us that we need to run it at 1MHz. Change the divider value as shown below.

```
SysCtlClockSet(SYSCTL_SYSDIV_8 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
```

13. Since we're setting up clocks, let's set up the PWM clock with a value of 16 for the divider. That will run the PWM at 62.5kHz. Add the following line right after the previous one:

```
SysCtlPWMClockSet(SYSCTL_PWMDIV_16);
```

14. The code to enable the GPIOF peripheral is already there. We need to enable the PWM module. Add the following line of code just before the GPIO enable:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM);
```

15. The last line of existing code configures the pin that's connected to the LED as an output. We want to send the output from the PWM to that pin instead. Change that line of code to match the code below:

```
GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0);
```

16. To do some further settings, we need to calculate the PWM Clock based on the system clock. We set the PWM divider to 16, so we need to divide the system clock by 16. Add this line after the last one:

```
ulPWMClock = SysCtlClockGet() / 16;
```

17. Next, we must calculate the number of PWM clock ticks for the desired output frequency. Then subtract one since the counter starts from zero. Add this line after the last:

```
ulLoad = (ulPWMClock / PWM_FREQUENCY) - 1;
```

18. Now we can configure PWM generator 0 as a down counter. It will count down to zero from the value that we load into the period register, and then start again at the load value. Add these two lines after the last:

```
PWMGenConfigure(PWM_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
PWMGenPeriodSet(PWM_BASE, PWM_GEN_0, ulLoad);
```

19. We need to set the pulse width. Loading the pulse width setting with half the ulLoad value will create a 50% duty cycle. Add this line after the last:

```
PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, ulLoad / 2);
```

20. In order for the PWM signal to reach the output pin, it must be enabled. Add this line after the last:

    `PWMOutputState(PWM_BASE, PWM_OUT_0_BIT, true);`

21. Finally, we can turn on the PWM generator. Add this line after the last:

    `PWMGenEnable(PWM_BASE, PWM_GEN_0);`

    Note that there is no code inside the while(1) loop, the PWM generator will be completely autonomous once it has been programmed. Click the Save button to save your work.

    Since we aren't using interrupts, the generic startup code and vector table, **`startup_ccs.c`** will work fine for this lab.

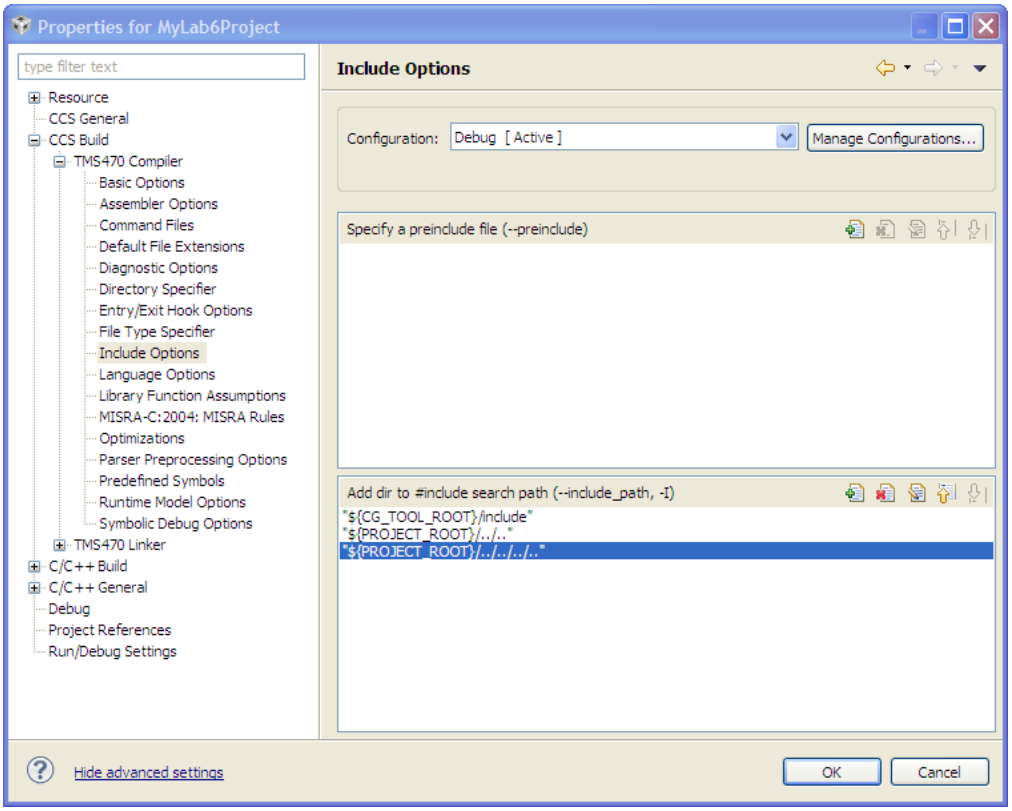
    Your final code should look something like this:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/pwm.h"

#define PWM_FREQUENCY         1

// Driver library error routine
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

int main(void)
{
  volatile unsigned long ulLoad;
  volatile unsigned long ulPWMClock;

  SysCtlClockSet(SYSCTL_SYSDIV_8 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
  SysCtlPWMClockSet(SYSCTL_PWMDIV_16);

  SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

  GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0);

  ulPWMClock = SysCtlClockGet() / 16;
  ulLoad = (ulPWMClock / PWM_FREQUENCY) - 1;
  PWMGenConfigure(PWM_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);

  PWMGenPeriodSet(PWM_BASE, PWM_GEN_0, ulLoad);
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, ulLoad / 2);
  PWMOutputState(PWM_BASE, PWM_OUT_0_BIT, true);

  PWMGenEnable(PWM_BASE, PWM_GEN_0);

  while(1)
    {
    }

}
```

Set the Build Options

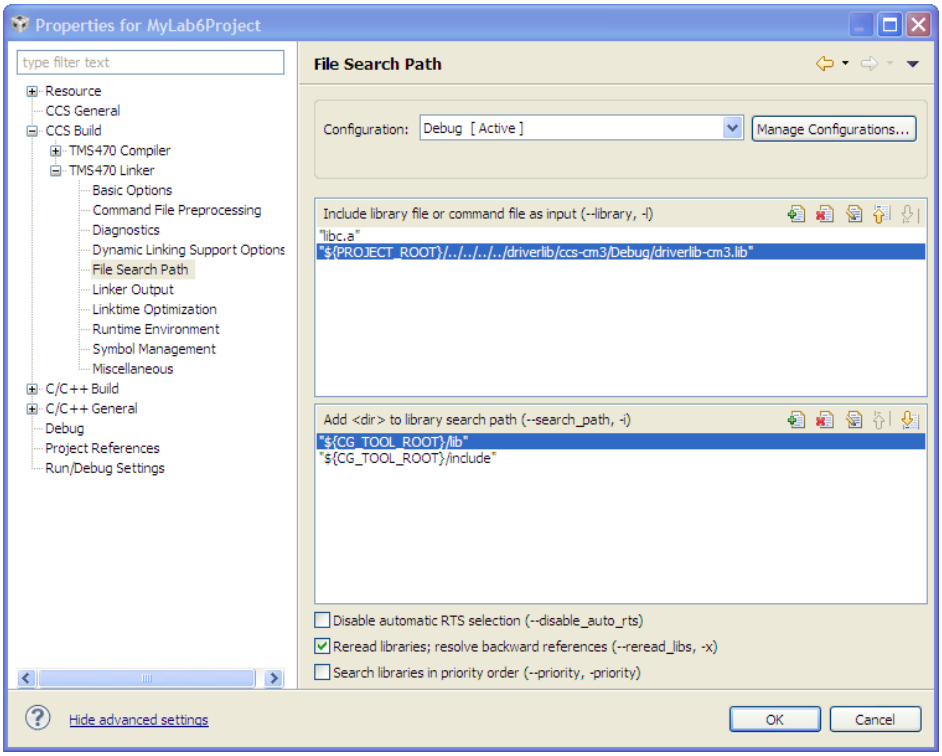
22. Right-click on MyLab6Project in the Project Explorer pane and select Properties. Click the + left of TMS470 Compiler and click on Include Options. In the bottom, include search path pane, click the Add button  and, one at a time, add the following two include search paths.

    ${PROJECT_ROOT}/../..
    ${PROJECT_ROOT}/../../../..



23. Click **File Search Path** under **TMS470 Linker**. Add the following include library file to the top window:

${PROJECT_ROOT}/../../../../driverlib/ccs-cm3/Debug/driverlib-cm3.lib



Click OK to save your changes.

Run the Code

24. Compile and download your application by clicking the Debug button  on the menu bar. If you have any issues, correct them, and then click the Debug button again. After a successful build, the CCS Debug perspective will appear. Remove the expressions in the Expressions window by right-clicking on the first and selecting Remove All.

    Click the Run button  to run the program that was downloaded to the flash memory of your device. The program will generate a 1 Hz PWM signal on Pin 0 of GPIO Port F. This pin is connected to the STATUS LED on the EK-LM3S8962.

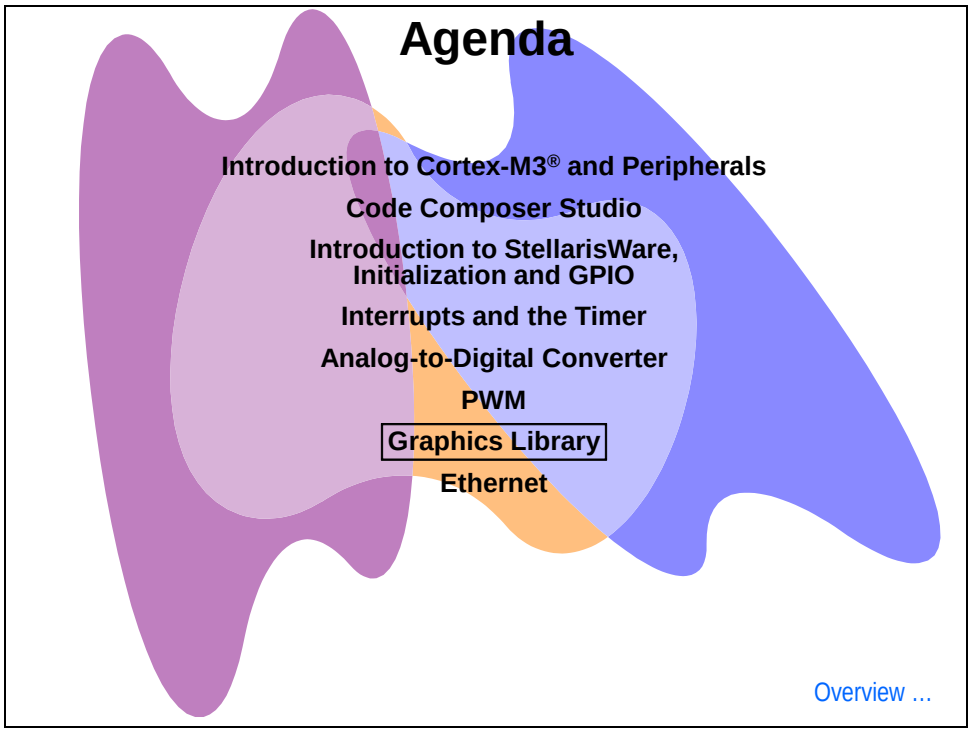
    Feel free to experiment with the value of PWM_FREQUENCY or the calculation of the pulse width. Once you exceed a frequency of about 30Hz, the LED will appear to be illuminated continuously. You can use the pulse width to vary the apparent brightness of the LED. Try a duty cycle of 100% with **`u1Load`** down to 1% with **`u1Load/100`**.

    When you're finished, click the Terminate  button to return to the Editing perspective, close the MyLab6Project project and minimize Code Composer Studio.

You're done.

Introduction

This module will introduce you to the use of the graphic library. Unfortunately the 8962 evaluation board doesn't have the large, color touch-screen that other boards do, so this will be a simple overview of those capabilities. The lab will utilize the graphics library to initialize and draw text and shapes on the OLED display.



Module Topics

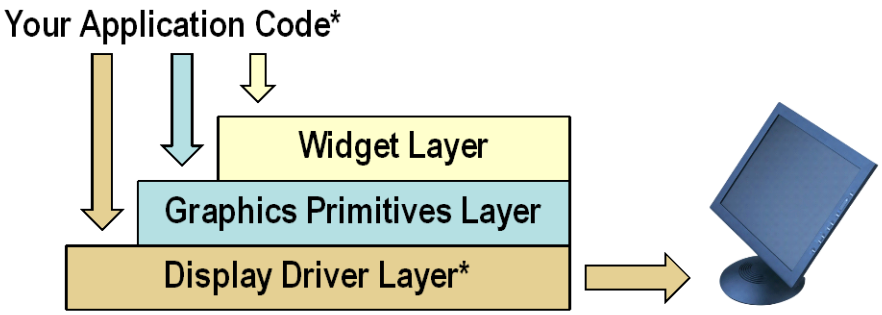
StellarisWare Graphics Library

Graphics Library Overview

The Stellaris Graphics Library provides graphics primitives and widgets sets for creating graphical user interfaces on Stellaris controlled displays.

Note that Stellaris devices do not have an LCD interface. The interface to smart displays is done through serial or EPI ports.

The graphics library consists of three layers to interface your application to the display:



* = user written or modified

Graphics Library Overview

Your application can call any of the layers.

The design of the graphics library is governed by the following goals:

- Components are written entirely in C except where absolutely not possible.
- The graphics library is easy to understand.
- The components are reasonably efficient in terms of memory and processor usage.
- Components are as self-contained as possible.
- Where possible, computations that can be performed at compile time are done there instead of at run time.
- The graphics library can be built with more than one tool chain.

Display Driver…

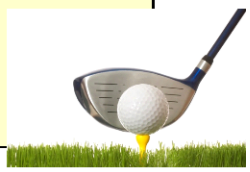
Graphics Library Layers

Display Driver

Low level interface to the display hardware

Provides:

- Routines for display-dependant operations like:
  - Initialization
  - Backlight control
  - Contrast
  - Translation of 24-bit RGB values to screen dependent color map
- Drawing routines for the graphics library like:
  - Flush
  - Line drawing
  - Pixel drawing
  - Rectangle drawing



Graphics Primitives...

Graphics Primitives

Low level drawing operations

- Drawing support for:
  - Lines, circles, text and bitmap images
- Support for off-screen buffering
- Foreground and background drawing contexts
- Color is represented as a 24-bit RGB value (8-bits per color)
  - ~150 pre-defined colors are provided
- 134 pre-defined fonts based on the Computer Modern typeface

Widget Framework...

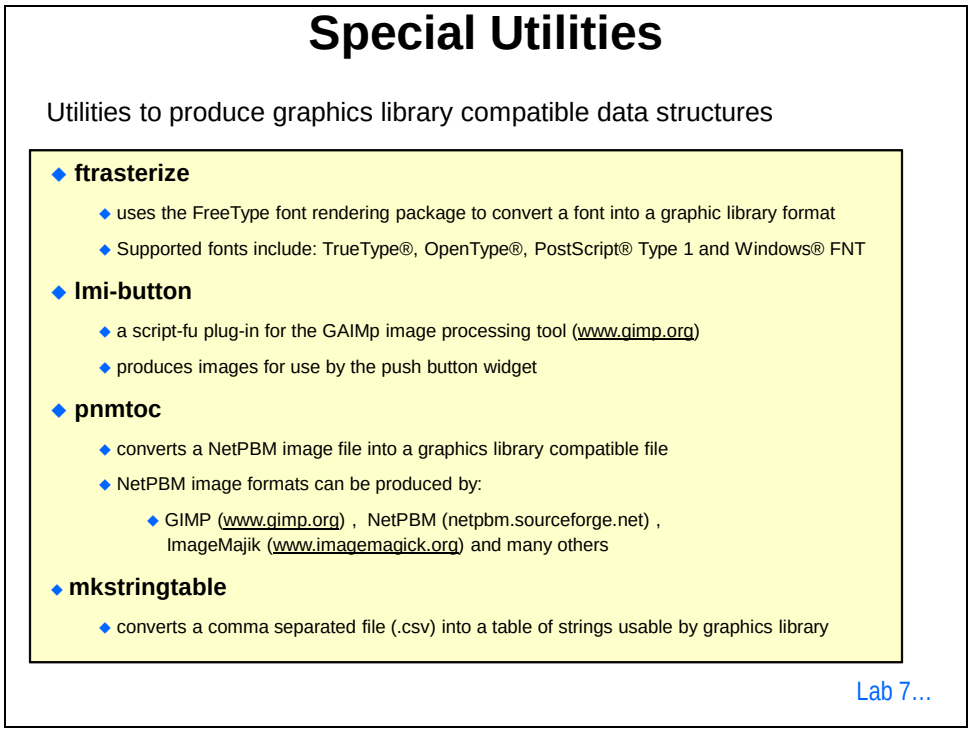
Widget Framework

Ties an on-screen element to user input

- Canvas – a simple drawing surface with no user interaction
- Checkbox – select/unselect
- Container – a visual element to group on-screen widgets
- Push Button – an on-screen button that can be pressed to perform an action
- Radio Button – selections that form a group; like low, medium and high
- Slider – vertical or horizontal to select a value from a predefined range
- ListBox – selection from a list of options

Special utilities…

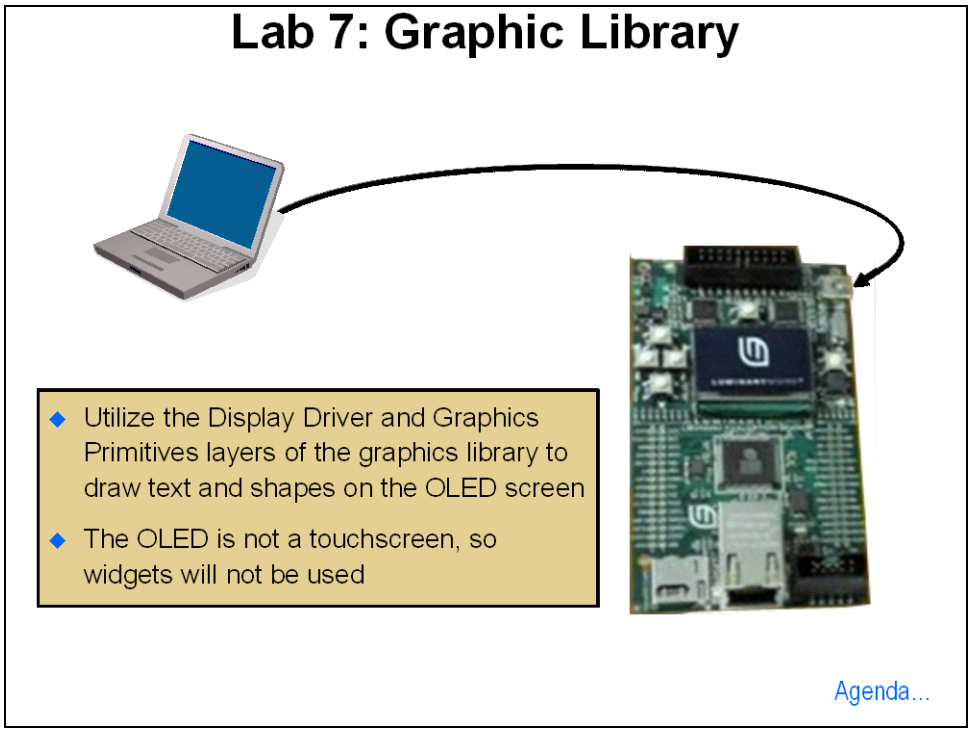
Special Utilities



Lab 7: Graphics Library

Objective

In this lab we'll use the lower two layers of the graphics library to place shapes, text and an image on the OLED display of the 8962 evaluation board.



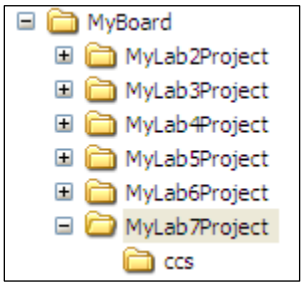
Procedure

Create New Project Folders

1. You know the drill … we need to create some folders to hold Lab 7.

   Open Windows Explorer and navigate to `C:\StellarisWare\boards\MyBoard`. Right-click in the open space of the right-hand pane and select New → Folder. Name the new folder `MyLab7Project` and press the Enter key.

   Double click on `MyLab7Project` to enter the folder and then right-click in the wide open right-hand pane. Select New → Folder. Name the new folder `ccs` and press the Enter key.



Add Source Files to Folder

2. There is an existing driver for the OLED display in StellarisWare, but it was written for another board. We'll copy the .c and .h files into our folder and then modify them.

   Navigate to `C:\StellarisWare\boards\rdk-bdc\bdc-ui` and copy `rit128x96x4.c` and `rit128x96x4.h`. Paste them into the `C:\StellarisWare\boards\MyBoard\MyLab7Project\ccs` folder.

   Copy the `startup_ccs.c` file from the `MyLab6Project` folder into the `C:\StellarisWare\boards\MyBoard\MyLab7Project\ccs` folder.

   Close Windows Explorer.

Image Conversion

3. The first task that we're going to have the lab software do is to display an image. So we need to covert an image into a format that the graphics library can understand. If you have not done so already, download GIMP from www.gimp.org and install it on your PC. Steps 4 and 5 go through the process of clipping the TI logo below and displaying it on the 8962 evaluation board's OLED display. If you prefer to use an existing image or photograph, or one taken from your smartphone camera now, simply adapt the steps below.

4. Press PrtScn on your keyboard. This will copy the screen to your clipboard.

   Open GIMP and click Edit → Paste. In the toolbox window, click the rectangular selection tool, and select tightly around the TI logo as shown above. Zoom in if that is easier for you. Click Image → Crop to selection. Click Image → Scale Image and make sure that the image size width/height are less than 128, 96. Then click Colors → Invert Colors to invert TI logo black and white colors. If you are using a photograph, you can skip this.

   Convert the image to indexed mode by clicking Image → Mode → Indexed. Select Generate optimum palette and select 16 which is the gray scale resolution of the OLED. Click Convert.

   Save the file by clicking File → Save As. Name the image `pic`, change the save folder to `C:\StellarisWare\tools\bin` and select PNM image as the file type using the selection right above the Help button. Click Save. When prompted, select Raw as the data formatting and click Save. Close GIMP.

5. Now that we have a source image file in PNM format, we can convert it to something that the graphics library can handle. We will use the `pnmtoc` (PNM to C array) conversion utility to do the translation.

   Open a command prompt by clicking Start → Run. Type `cmd` in the window and click Open.

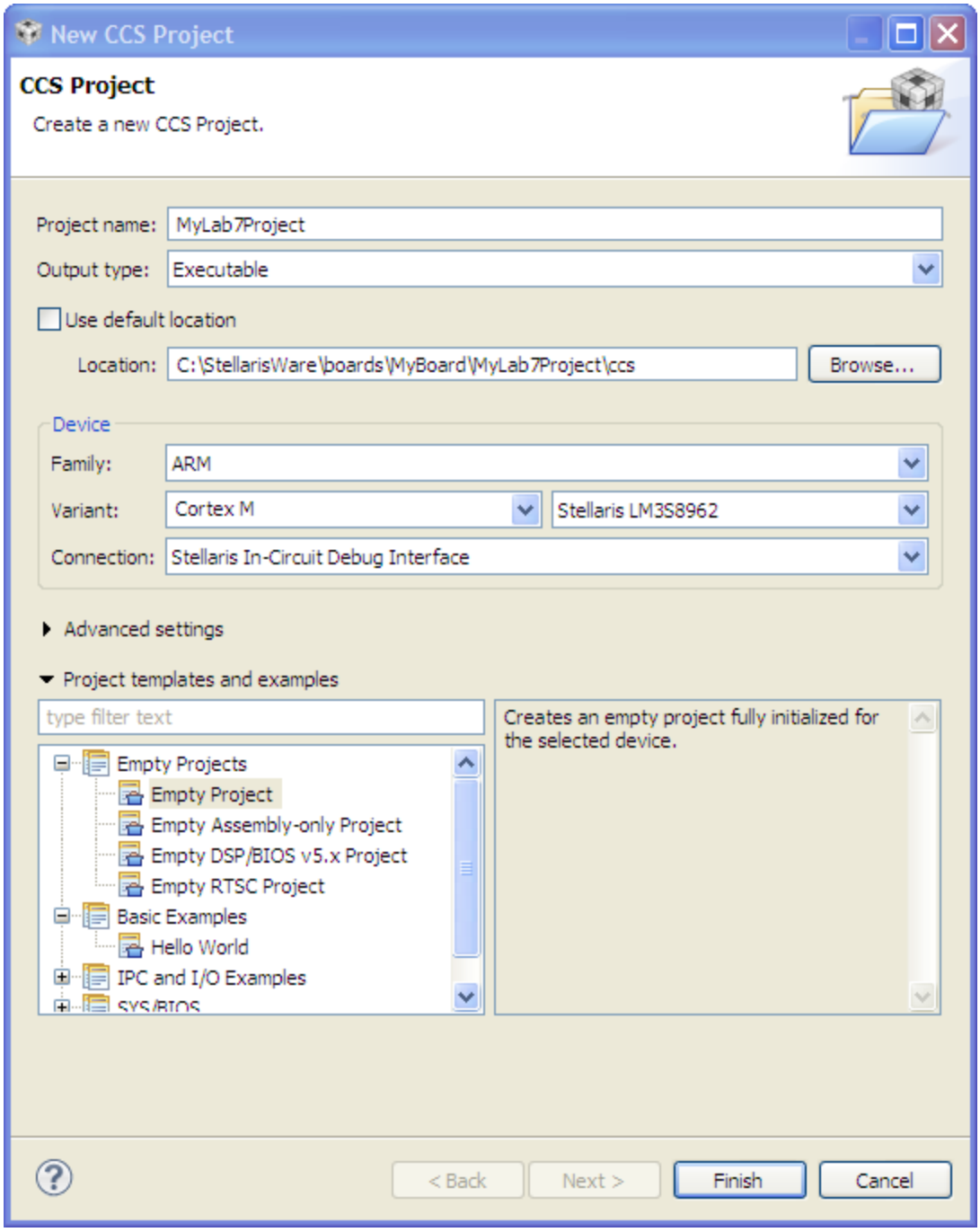
   The `pnmtoc` utility is in `C:\StellarisWare\tools\bin`. So type `cd C:\StellarisWare\tools\bin` in the command window, then hit Enter to change the folder to that location.

   Finally, perform the conversion by typing `pnmtoc –c pic.pnm > pic.c` in the command window and hit Enter. If the process works correctly the cursor will simply drop to a new line.

6. Copy pic.c from `C:\StellarisWare\tools\bin` to your `MyLab7\ccs` folder.

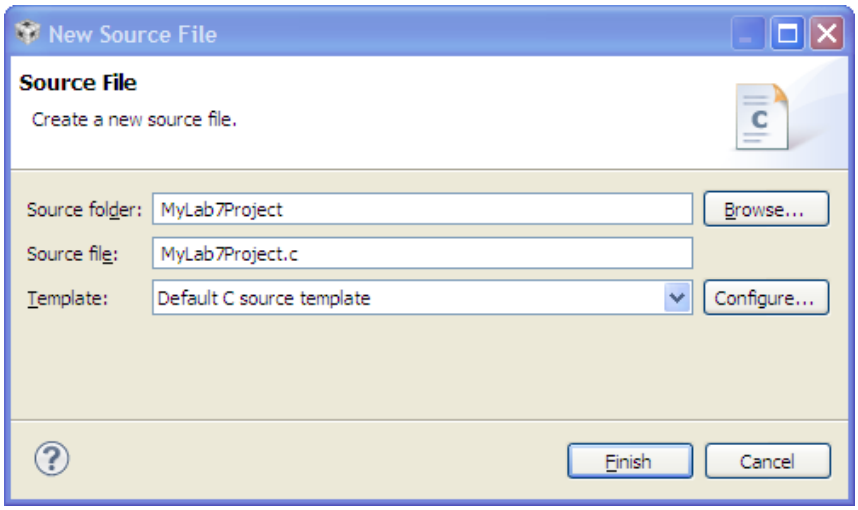
Create `MyLab7Project` Project

7. Maximize Code Composer. On the CCS menu bar select File → New → CCS Project. Make the selections shown below. Make sure to uncheck the "Use default location" checkbox and select the correct path to the "ccs" folder you created. Click Finish.



Add Files to Project

8. Delete main.c from the project. Note that pic.c, rit128x96x4.c and rit128x96x4.h were found in the folder and automatically added to the project.
9. From the CCS menu bar, click File → New Source File. When the New Source File dialog appears, make the selections below to create the main C code file and click Finish.



Modify the Display Driver

10. The lowest level of the graphics library is the display driver. The display driver functions are implemented in `rit128x96x4.c`. Open the file and look for the functions needed to make the display operate; initialization, communication with the display controller, buffer control, etc. This display driver file was written for another board that used a slightly different pin-out, so we need to make a couple of minor changes. In `rit128x96x4.c`, find the following definitions:

```
#define SYSCTL_PERIPH_GPIO_OLEDDC   SYSCTL_PERIPH_GPIOC
#define GPIO_OLEDDC_BASE            GPIO_PORTC_BASE
#define GPIO_OLEDDC_PIN             GPIO_PIN_7
#define GPIO_OLEDEN_PIN             GPIO_PIN_6
```

Change these lines to read as follows. The differences are highlighted in red:

```
#define SYSCTL_PERIPH_GPIO_OLEDDC   SYSCTL_PERIPH_GPIOA
#define GPIO_OLEDDC_BASE            GPIO_PORTA_BASE
#define GPIO_OLEDDC_PIN             GPIO_PIN_6
#define GPIO_OLEDEN_PIN             GPIO_PIN_7
```

Save your changes and close the editor pane for this file.

Modify pic.c

11. Open **pic.c** and add the following include to the very top of the file:

```
#include "grlib/grlib.h"
```

Your **pic.c** file should look something like this (your data will vary greatly):

```
#include "grlib/grlib.h"

const unsigned char g_pucImage[] =
{
    IMAGE_FMT_4BPP_COMP,
    86, 0,
    77, 0,

    15,
    0x00, 0x01, 0x00,
    0x18, 0x1a, 0x19,
    0x28, 0x2a, 0x28,
    0x38, 0x3a, 0x38,
    0x44, 0x46, 0x44,
    0x54, 0x57, 0x55,
    0x62, 0x65, 0x63,
    0x72, 0x75, 0x73,
    0x81, 0x84, 0x82,
    0x93, 0x96, 0x94,
    0xa2, 0xa5, 0xa3,
    0xb3, 0xb6, 0xb4,
    0xc4, 0xc7, 0xc5,
    0xd7, 0xda, 0xd8,
    0xe8, 0xeb, 0xe9,
    0xf4, 0xf8, 0xf5,

    0xff, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0xff, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0xff, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0xfc, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x03, 0x77,
    0x23, 0x77, 0x77, 0xe9, 0x77, 0x78, 0x70, 0x07, 0x07, 0xc1, 0x77, 0x2c,
    0x04, 0xde, 0xee, 0xee, 0xee, 0xe9, 0x3c, 0xee, 0xa1, 0x07, 0x07, 0x77,
    0x2c, 0x03, 0xcf, 0x00, 0xee, 0xee, 0xee, 0xef, 0xee, 0xef, 0xfe, 0xa0,
    0xf0, 0x07, 0x07, 0x77, 0x2c, 0x03, 0xcf, 0xee, 0xee, 0x4f, 0xee, 0xe9,
    0xee, 0xa0, 0x07, 0x07, 0x77, 0x2c, 0x04, 0x03, 0xcf, 0xee, 0xee, 0xee,
    0xe9, 0xee, 0x90, 0xf0, 0x07, 0x07, 0x77, 0x2c, 0x03, 0xcf, 0xee, 0xee,
    0x4f, 0xee, 0xe9, 0xee, 0x90, 0x07, 0x07, 0x77, 0x2c, 0x04, 0x03, 0xcf,

    many, many more lines of this data …

    0x77, 0x2c, 0x19, 0xfe, 0xee, 0xef, 0x03, 0xee, 0xee, 0xee, 0xee, 0xfb,
    0x20, 0x07, 0x07, 0xc1, 0x77, 0x2c, 0x05, 0xdf, 0xee, 0xee, 0xee, 0xe9,
    0x78, 0xf9, 0x07, 0x07, 0x77, 0x2d, 0x01, 0x8d, 0xee, 0x2f, 0xee, 0xee,
    0xe9, 0xf7, 0x07, 0x07, 0x77, 0x2e, 0x00, 0x39, 0xef, 0xee, 0xee, 0xee,
    0xee, 0xee, 0xf7, 0xf0, 0x07, 0x07, 0x77, 0x2e, 0x06, 0xdf, 0xee, 0xee,
    0x0f, 0xee, 0xee, 0xee, 0xf6, 0x07, 0x07, 0x77, 0x2f, 0x01, 0x7d, 0xfe,
    0xee, 0xee, 0xee, 0xee, 0xf7, 0x07, 0xe0, 0x07, 0x77, 0x2f, 0x17, 0xdf,
    0xee, 0xee, 0xee, 0x3c, 0xee, 0xf7, 0x07, 0x07, 0x77, 0x2f, 0x01, 0x7d,
    0x03, 0xee, 0xee, 0xee, 0xee, 0xf9, 0x10, 0x07, 0x07, 0xc0, 0x77, 0x2f,
    0x05, 0xad, 0xee, 0xfe, 0xee, 0xfc, 0x78, 0x20, 0x07, 0x07, 0x77, 0x2f,
    0x00, 0x27, 0x9d, 0x0f, 0xed, 0xee, 0xec, 0x40, 0x07, 0x07, 0x77, 0x2f,
    0x01, 0x00, 0x00, 0x28, 0x9a, 0xcc, 0xa9, 0x30, 0x07, 0xff, 0x07, 0x77,
    0x2f, 0x07, 0x07, 0x07, 0x07, 0x07, 0xc0, 0x07, 0x07,
};
```

Save your changes and close the **pic.c** editor pane.

MyLab7Project .c Includes

12. Add (or copy/paste) the following lines to the top of `MyLab7Project.c` :

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "grlib/grlib.h"
#include "rit128x96x4.h"
```

`grlib.h` : graphics library definitions

`rit128x96x4.h` : API function prototypes for display driver file

Pointer to Image Array

13. Line a line for spacing, then add the following global declaration of the image array after the lines above:

```
extern const unsigned char g_pucImage[];
```

Driver Library Error Routine

14. The following code will be called if the driver library encounters an error.

    Leave a blank line for spacing and enter these line of codes after the lines above:

```
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif
```

Main()

15. The main() routine will be next. Leave a blank line for spacing and enter these line of codes after the lines above:

```
int main(void)
{
}
```

Variables

16. The variables defined below are used for initializing the `Context` and `Rect` structures in this example. `Context` is a definition of the screen such as the clipping region, default color and font. `Rect` is a simple structure for drawing rectangles. Insert these two lines as the first inside main():

```
tContext sContext;
tRectangle sRect;
```

Initialization

17. Set the clocking to run at 50 MHz using the PLL. System clock must be at least 7MHz (see the next step for more detail). Leave a line for spacing, then insert this line after the last ones:

```
SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
```

18. Using the display driver initialization, set the bit rate for the SSI interface to 3.5MHz or 3500000. The clock for the SSI will be the same as system clock and the system clock must be greater than or equal to twice the bit rate in master mode (7MHz vs 3.5MHz). If this were one of the slave modes then the system clock would have to be 12 times the bit rate. Insert this line after the last:

```
RIT128x96x4Init(3500000);
```

19. Initialize the graphics context. This function initializes a drawing context, preparing it for use. The provided display driver will be used for all subsequent graphics operations, and the default clipping region will be set to the extent of the OLED screen. Insert this line after the last:

```
GrContextInit(&sContext, &g_sRIT128x96x4Display);
```

20. Clear the entire screen. The following code will create a rectangle that covers the entire screen, sets the foreground color to black, and fills the rectangle by passing the structure `sRect` by reference. The top left corner of the OLED Display, in between the 'RESET' button and the '^' (up arrow) represents the point (0,0) and the bottom right corner near the 'STATUS' LED represents the point (127,95). Leave a line for spacing, then insert these lines after the last ones:

```
sRect.sXMin = 0;
sRect.sYMin = 0;
sRect.sXMax = 127;
sRect.sYMax = 95;
GrContextForegroundSet(&sContext, ClrBlack);
GrRectFill(&sContext, &sRect);
```

Displaying the Image

21. Display the image by passing the global image variable g_pucImage generated from pnmtoc.exe into GrImageDraw(...) and approximately center the image on the screen by locating the top-left corner at (20,3) …we'll adjust this later if needed. Leave a line for spacing, then insert this line after the last:

```
GrImageDraw(&sContext, g_pucImage, 20, 3);
```

22. The function call below flushes any cached drawing operations. For display drivers that draw into a local frame buffer before writing to the actual display, calling this function will cause the display to be updated to match the contents of the local frame buffer. Leave a line for spacing, then insert this line after the last:

```
GrFlush(&sContext);
```

23. We want to leave the image on the screen long enough to see it, so add this 9 second delay. Leave a line for spacing, then insert this line after the last:

```
SysCtlDelay(SysCtlClockGet() * 3);
```

24. Before we go any further, we'd like to take the code we have for a test run. With that in mind we're going to add the final code pieces now, and insert later lab code in front of this.

    Since an OLED display is prone to burn-in, it would be best to clear the display right before the code ends. This performs the same function as step 24 and also flushes the cache. Leave several lines for spacing and add this code below the last:

```
sRect.sXMin = 0;
sRect.sYMin = 0;
sRect.sXMax = 127;
sRect.sYMax = 95;
GrContextForegroundSet(&sContext, ClrBlack);
GrRectFill(&sContext, &sRect);
GrFlush(&sContext);
```

25. Add a while loop to the end of the code to stop execution. Leave a line for spacing, then insert this line after the last:

```
while(1)
{
}
```

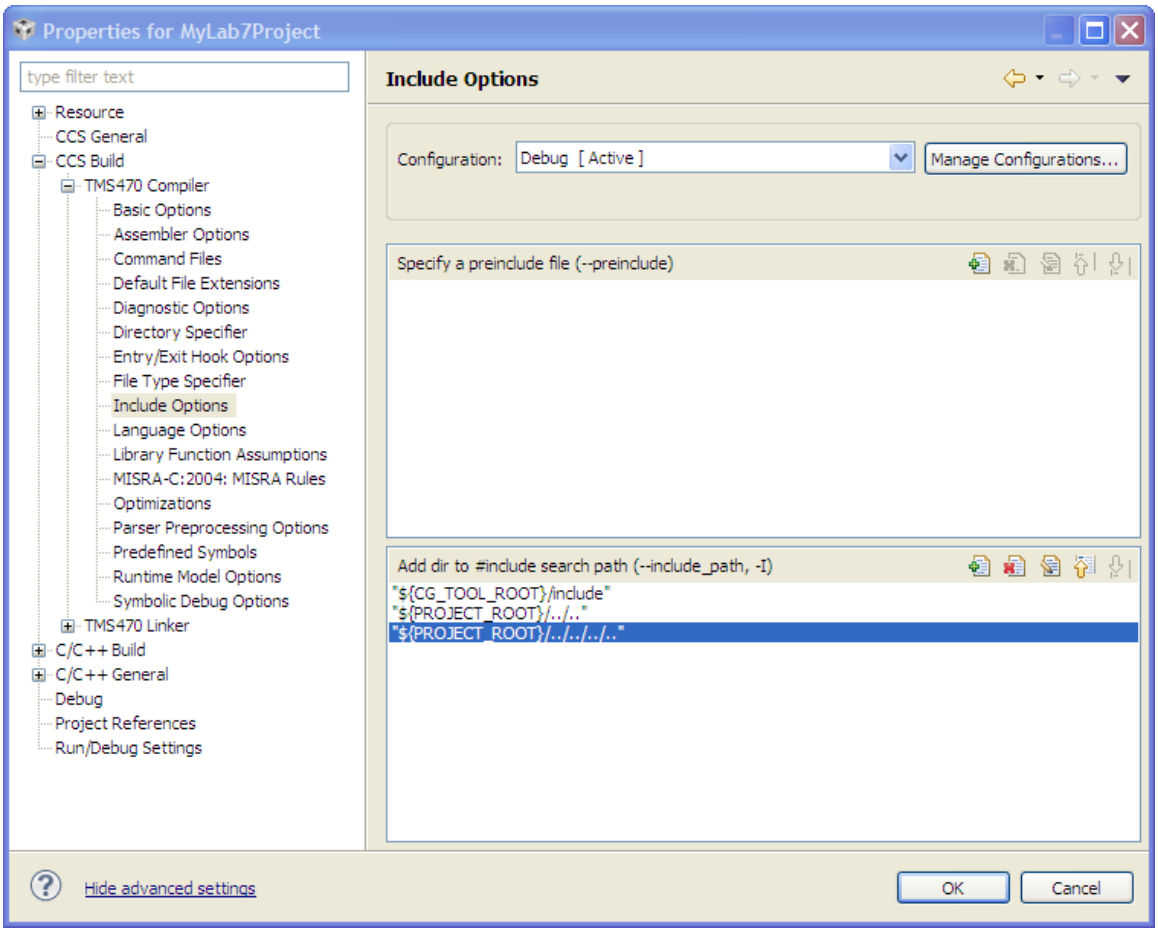
Your code should look like this:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "grlib/grlib.h"
#include "rit128x96x4.h"

extern const unsigned char g_pucImage[];

#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

int main(void)
{
  tContext sContext;
  tRectangle sRect;

  SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
  RIT128x96x4Init(3500000);
  GrContextInit(&sContext, &g_sRIT128x96x4Display);

  // Clear the screen
  sRect.sXMin = 0;
  sRect.sYMin = 0;
  sRect.sXMax = 127;
  sRect.sYMax = 95;
  GrContextForegroundSet(&sContext, ClrBlack);
  GrRectFill(&sContext, &sRect);

  // Draw image on OLED
  GrImageDraw(&sContext, g_pucImage, 20, 3);

  GrFlush(&sContext);

  // Delay 9 seconds
  SysCtlDelay(SysCtlClockGet() * 3);

  // Later lab steps go between here

  // and here

  // Clear the screen
  sRect.sXMin = 0;
  sRect.sYMin = 0;
  sRect.sXMax = 127;
  sRect.sYMax = 95;
  GrContextForegroundSet(&sContext, ClrBlack);
  GrRectFill(&sContext, &sRect);
  GrFlush(&sContext);

  // Loop forever
  while(1)
  {
  }
}
```

Set the Build Options

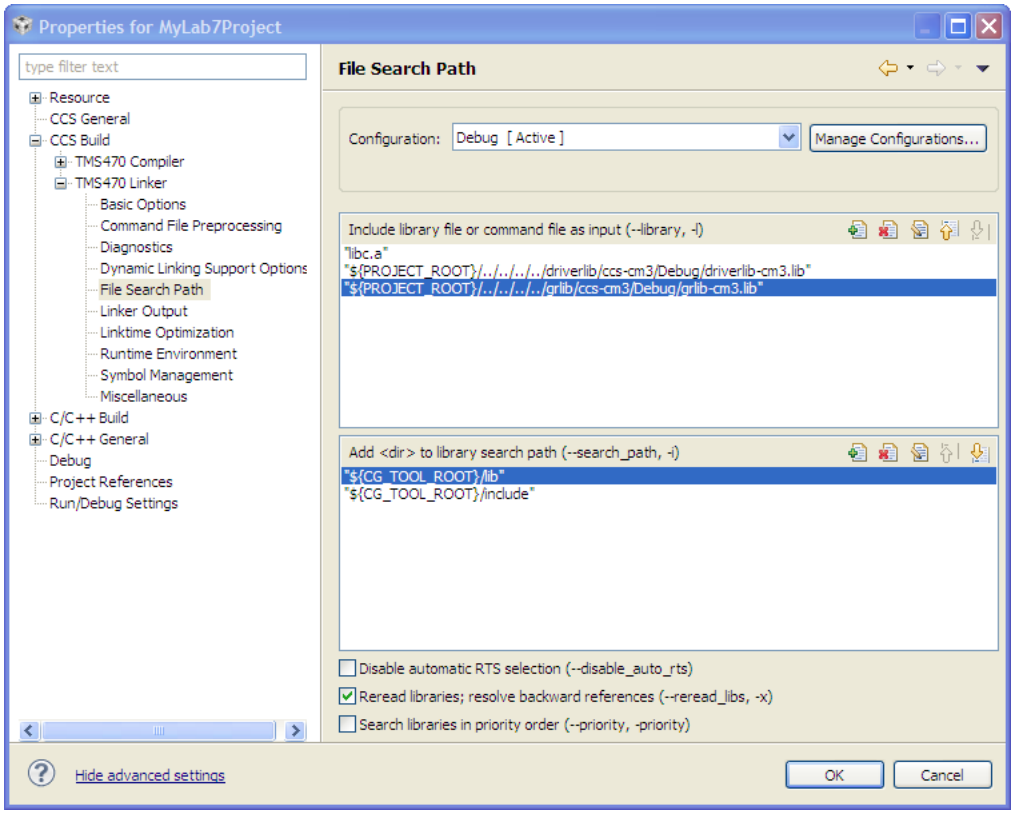
26. Right-click on MyLab7Project in the Project Explorer pane and select Properties. Click the + left of TMS470 Compiler and click on Include Options. In the bottom, include search path pane, click the Add button and, one at a time, add the following two include search paths.

${PROJECT_ROOT}/../..
${PROJECT_ROOT}/../../../..



27. Click **File Search Path** under **TMS470 Linker**. Add the following include library files to the top window:

${PROJECT_ROOT}/../../../../driverlib/ccs-cm3/Debug/driverlib-cm3.lib
${PROJECT_ROOT}/../../../../grlib/ccs-cm3/Debug/grlib-cm3.lib



28. Click **Basic Options** under **TMS470 Linker**. Change the C system stack size from 256 to **512**. Click **OK** to save your changes.

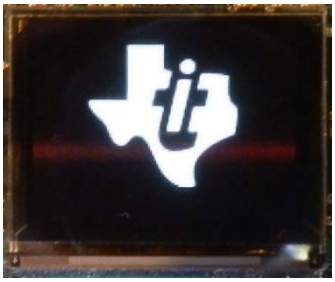
Run the Code

29. Compile and download your application by clicking the Debug button on the menu bar. If you have any issues, correct them, and then click the Debug button again. After a successful build, the CCS Debug perspective will appear.

    Click the Run button to run the program that was downloaded to the flash memory of your device. If your coding efforts were successful, you should see your image appear on the OLED display for about 9 seconds, then disappear.

    If you want to adjust the position if the image on the screen, go back to the code we entered in step 24 and tweak the numbers.

    When you're finished, click the Terminate button to return to the Edit perspective.



Display Text On-Screen

30. Refer back to the code on page 7-16. In MyLab7Project.c in the area marked:

```
// Later lab steps go between here

// and here
```

insert the following code to clear the screen and flush the buffer:

```
sRect.sXMin = 0;
sRect.sYMin = 0;
sRect.sXMax = 127;
sRect.sYMax = 95;
GrContextForegroundSet(&sContext, ClrBlack);
GrRectFill(&sContext, &sRect);
GrFlush(&sContext);
```

31. Next we'll display the introduction text. Display text starting at (x,y) coordinates (11,7) and (27,23) with background of each character turned off, or set to 0. The third (-1) parameter allows a portion of the string to be examined without having to insert a NULL character at the stopping point (which would not be possible if the string was located in flash).

`GrRectDraw(...):` Put a border around the screen.

`GrContextForegroundSet(...):` Set the foreground for the text to be white.

`GrContextFontSet(...):` Set the font to be a max width of 6 pixels with a max height of 8 pixels.

`GrFlush(...):` And refresh the screen by matching the contents of the local frame buffer.

Add the following lines after the previous ones:

```
sRect.sXMin = 1;
sRect.sYMin = 1;
sRect.sXMax = 126;
sRect.sYMax = 94;
GrContextForegroundSet(&sContext, ClrWhite);
GrContextFontSet(&sContext, &g_sFontFixed6x8);
GrStringDraw(&sContext, "Texas Instruments", -1, 11, 7, 0);
GrStringDraw(&sContext, "Graphics Lab", -1, 27, 23, 0);
GrRectDraw(&sContext, &sRect);
GrFlush(&sContext);
```

32. Add a three second delay so that we can appreciate our work.

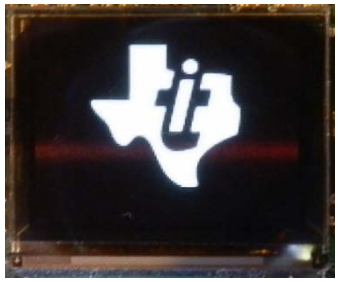
```
SysCtlDelay(SysCtlClockGet() * 1);
```

Your added code should look like this:

```
// Later lab steps go between here

// Clear the screen and flush the buffer
sRect.sXMin = 0;
sRect.sYMin = 0;
sRect.sXMax = 127;
sRect.sYMax = 95;
GrContextForegroundSet(&sContext, ClrBlack);
GrRectFill(&sContext, &sRect);
GrFlush(&sContext);

// Write text to the OLED
sRect.sXMin = 1;
sRect.sYMin = 1;
sRect.sXMax = 126;
sRect.sYMax = 94;
GrContextForegroundSet(&sContext, ClrWhite);
GrContextFontSet(&sContext, &g_sFontFixed6x8);
GrStringDraw(&sContext, "Texas Instruments", -1, 11, 7, 0);
GrStringDraw(&sContext, "Graphics Lab", -1, 27, 23, 0);
GrRectDraw(&sContext, &sRect);
GrFlush(&sContext);

// 3 second delay
SysCtlDelay(SysCtlClockGet() * 1);

// and here
```

Build, Load and Test

33. Build, load and run your code. If your changes are correct, you should see the image again for 9 seconds, followed by the on-screen text in a box for 3 seconds. Then the display will blank out. Return to the edit perspective when you're done.



Other Drawing

34. Let's add a circle. Make the foreground white and center the circle at (35,70) with a radius of 10. Add a line for spacing and add these lines after the one in step 32:

```
GrContextForegroundSet(&sContext, ClrWhite);
GrCircleDraw(&sContext, 35, 70, 10);
```

35. Draw another rectangle starting with the top left corner at (60,60) and finishing at the bottom right corner nearest the 'STATUS' LED on the board at (110,80). Note that since we have not refreshed the screen yet with `GrFlush(...)` we did not need to call `GrContextForegroundSet(...)`. However, this is entered here for the sake of copy/paste for portability of code. Add a line for spacing and add the following lines after the last ones:

```
sRect.sXMin = 60;
sRect.sYMin = 60;
sRect.sXMax = 110;
sRect.sYMax = 80;
GrContextForegroundSet(&sContext, ClrWhite);
GrRectDraw(&sContext, &sRect);
```

36. Draw a pixel at the center of the circle. Add a line for spacing and add this one after the ones above:

```
GrPixelDraw(&sContext, 35, 70);
```

37. Draw a horizontal line from (60,70) to (70,70). Add a line for spacing and add this one after the one above:

```
GrLineDrawH(&sContext, 60, 70, 70);
```

38. Draw a vertical line from (80,60) to (80,80). Add a line for spacing and add this one after the one above:

```
GrLineDrawV(&sContext, 80, 60, 80);
```

39. Draw a line from (90,60) to (100,80) and update the screen to show everything that has been drawn. Add a line for spacing and add the following lines after the last ones:

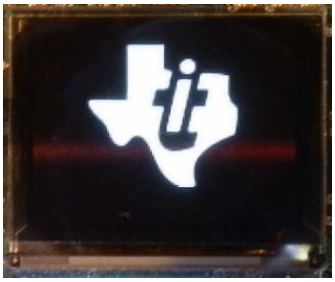
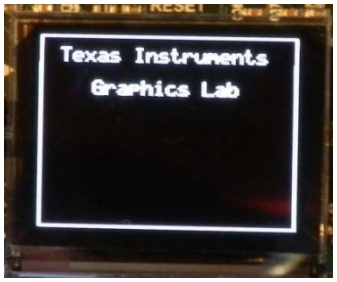
```
GrLineDraw(&sContext, 90, 60, 100, 80);
GrFlush(&sContext);
```

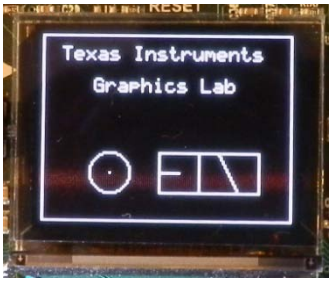
40. Add a short delay to appreciate your work. Add a line for spacing and add the following line after the last ones:

```
SysCtlDelay(SysCtlClockGet() * 1);
```

Build, Load and Test

41. Build, load and run your code to make sure that your changes work. Return to the edit perspective when you are done.



Final Touches

42. Fill the circle and rectangle with `GrCircleFill(...)` and `GrRectFill(...)` by passing the same coordinates and delay for 3 seconds. Add a line for spacing, then add the following lines after the one added in step 40:

```
GrCircleFill(&sContext, 35, 70, 10);
GrRectFill(&sContext, &sRect);
GrFlush(&sContext);

SysCtlDelay(SysCtlClockGet() * 1);
```

43. Display the original text, all the shapes drawn and add 'Completed.' starting at (40,39) and delay for 6 seconds. Add a line for spacing and add the following lines after the last ones:

```
GrContextForegroundSet(&sContext, ClrWhite);
GrContextFontSet(&sContext, &g_sFontFixed6x8);
GrStringDraw(&sContext, "Texas Instruments", -1, 11, 7, 0);
GrStringDraw(&sContext, "Graphics Lab", -1, 27, 23, 0);
GrStringDraw(&sContext, "Completed", -1, 40, 39, 0);
GrFlush(&sContext);

SysCtlDelay(SysCtlClockGet() * 2);
```

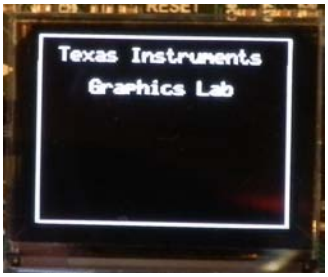
The code that we added starting in step 30 should look like the code on the following page:

```
        // Later lab steps go between here

        // Clear the screen and flush the buffer
        sRect.sXMin = 0;
        sRect.sYMin = 0;
        sRect.sXMax = 127;
        sRect.sYMax = 95;
        GrContextForegroundSet(&sContext, ClrBlack);
        GrRectFill(&sContext, &sRect);
        GrFlush(&sContext);

        // Write text to the OLED
        sRect.sXMin = 1;
        sRect.sYMin = 1;
        sRect.sXMax = 126;
        sRect.sYMax = 94;
        GrContextForegroundSet(&sContext, ClrWhite);
        GrContextFontSet(&sContext, &g_sFontFixed6x8);
        GrStringDraw(&sContext, "Texas Instruments", -1, 11, 7, 0);
        GrStringDraw(&sContext, "Graphics Lab", -1, 27, 23, 0);
        GrRectDraw(&sContext, &sRect);
        GrFlush(&sContext);

        SysCtlDelay(SysCtlClockGet() * 1); // Delay 3 seconds

        // Draw a circle
        GrContextForegroundSet(&sContext, ClrWhite);
        GrCircleDraw(&sContext, 35, 70, 10);

        // Draw a Rectangle
        sRect.sXMin = 60;
        sRect.sYMin = 60;
        sRect.sXMax = 110;
        sRect.sYMax = 80;
        GrContextForegroundSet(&sContext, ClrWhite);
        GrRectDraw(&sContext, &sRect);

        // Draw a pixel at the center of the circle
        GrPixelDraw(&sContext, 35, 70);

        // Draw a horizontal line
        GrLineDrawH(&sContext, 60, 70, 70);

        // Draw a vertical line
        GrLineDrawV(&sContext, 80, 60, 80);

        // Draw a line, then display all
        GrLineDraw(&sContext, 90, 60, 100, 80);
        GrFlush(&sContext);

        SysCtlDelay(SysCtlClockGet() * 1); // Delay 3 seconds

        // Fill the circle and rectangle
        GrCircleFill(&sContext, 35, 70, 10);
        GrRectFill(&sContext, &sRect);
        GrFlush(&sContext);

        SysCtlDelay(SysCtlClockGet() * 1); // Delay 3 seconds


        // Display original text and shapes and add text 'Completed.'
        GrContextForegroundSet(&sContext, ClrWhite);
        GrContextFontSet(&sContext, &g_sFontFixed6x8);
        GrStringDraw(&sContext, "Texas Instruments", -1, 11, 7, 0);
        GrStringDraw(&sContext, "Graphics Lab", -1, 27, 23, 0);
        GrStringDraw(&sContext, "Completed", -1, 40, 39, 0);
        GrFlush(&sContext);

        SysCtlDelay(SysCtlClockGet() * 2); // Delay for 6 seconds

        // and here
```

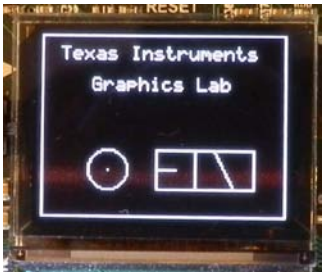
Build, Load and Test

44. Build, load and run your code to make sure that your changes work. Return to the edit perspective when you are done. Close MyLab7Proect and minimize Code Composer Studio. Congratulations! That was a lot of work!

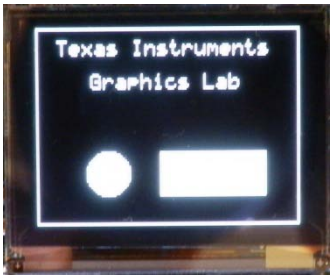
9 seconds



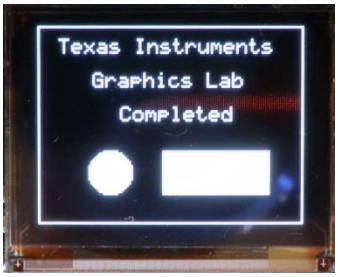
3 seconds



3 seconds



3 seconds



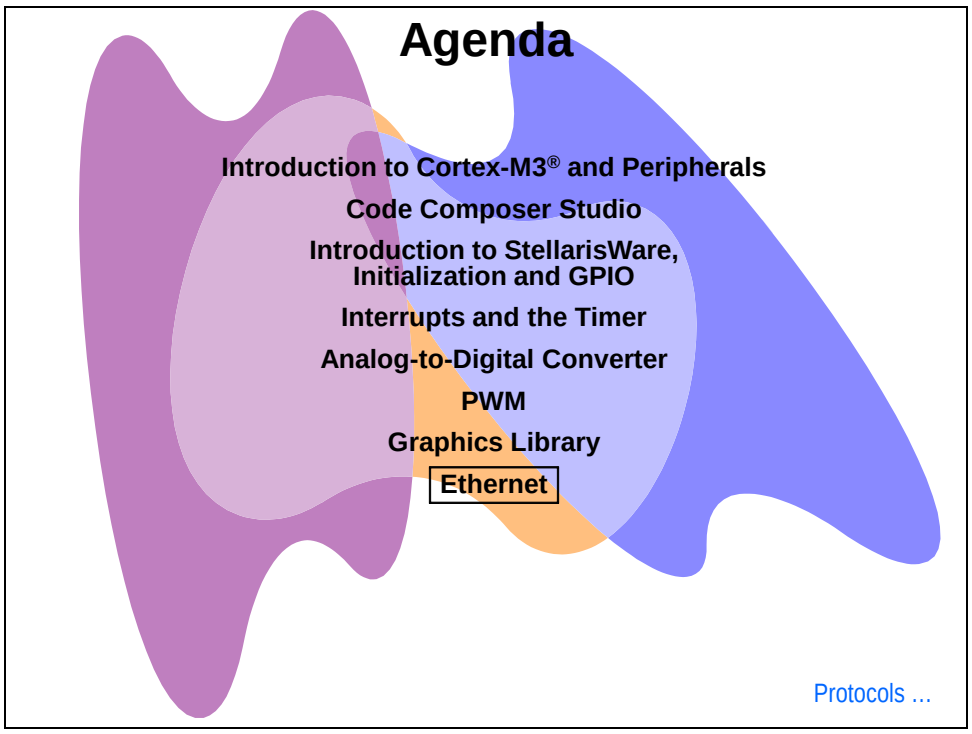
6 seconds, then blank

You're done.

Ethernet Peripheral

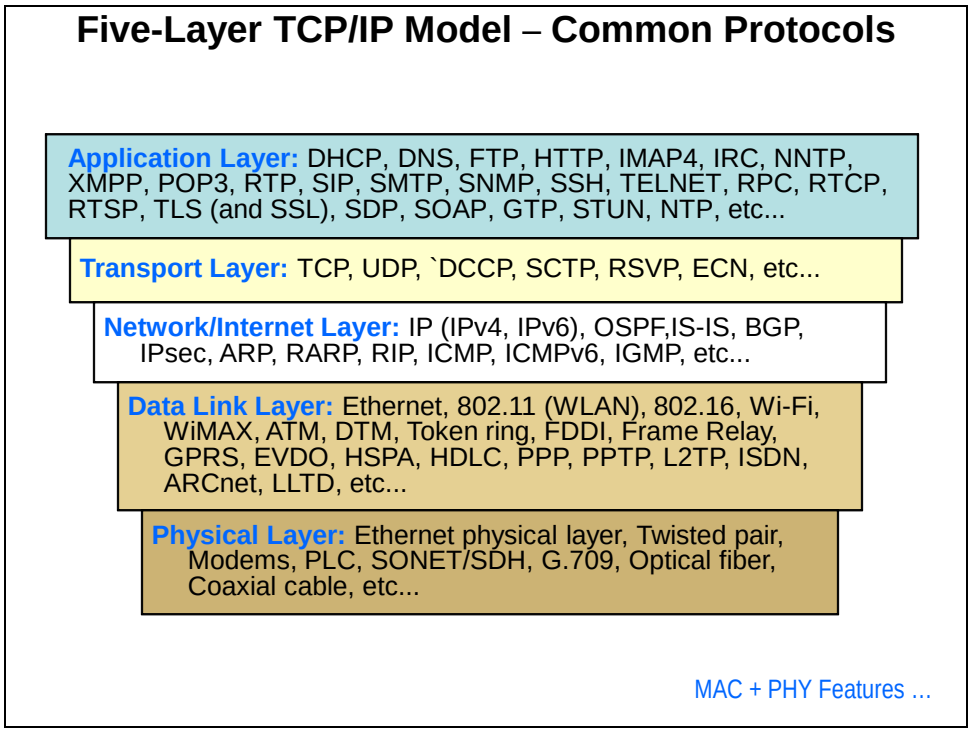
Introduction

This module covers the Ethernet peripheral, library and examples



Module Topics

Common Protocols



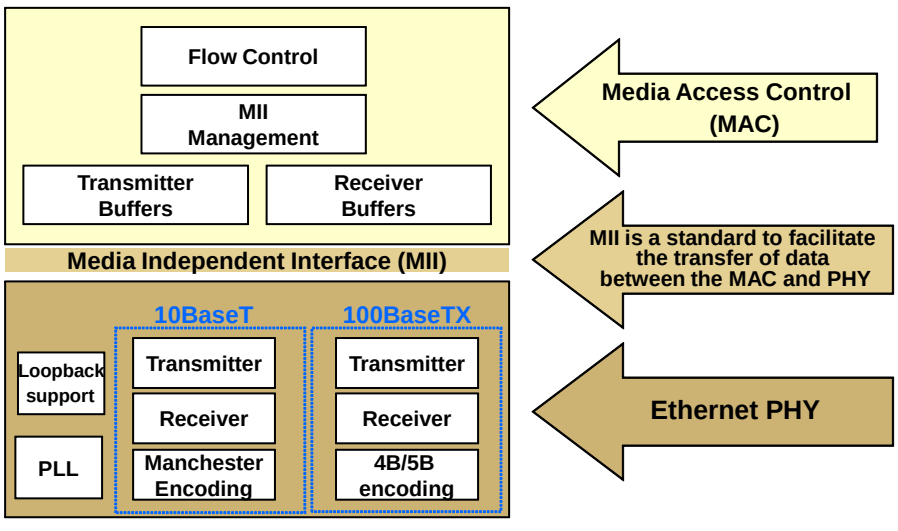
Stellaris MAC and PHY

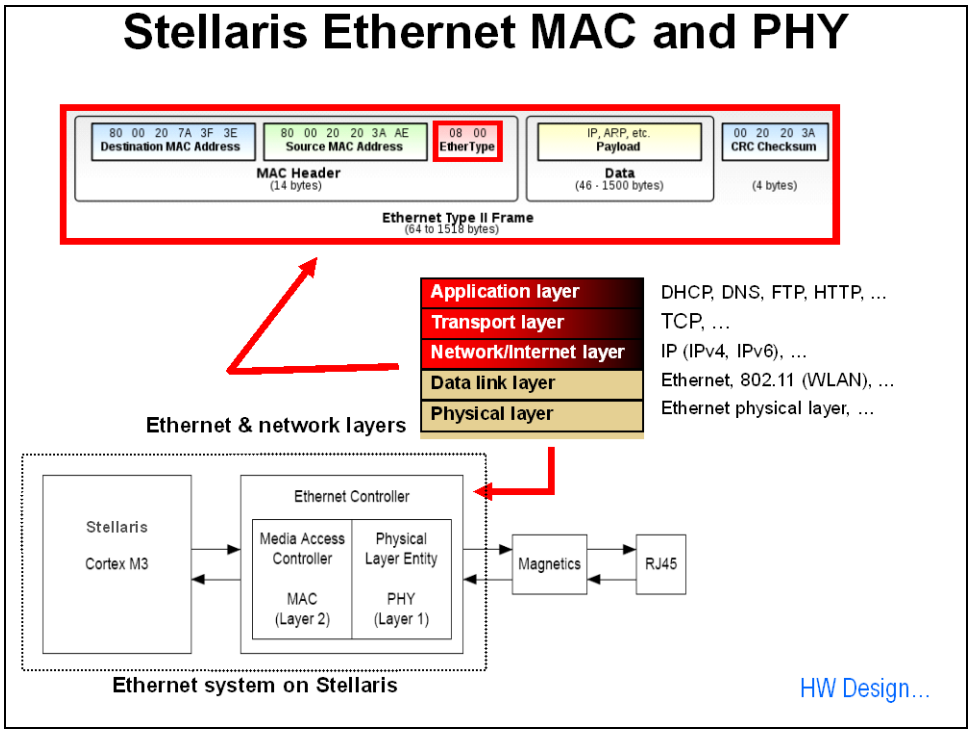
Stellaris Ethernet MAC and PHY Features

- Integrated 10/100 Mbps Transceiver (PHY)
- IEEE 1588 PTP Hardware-Assisted Support
- 10BASE-T and 100BASE-TX/RX IEEE 802.3 Full/Half-Duplex support
- Automatic MDI/MDI-X cross-over correction
- Programmable MAC address
- Promiscuous mode support
  - Ability to receive all packets sent on network – like CAN, versus only those sent to specific MAC address
- 2KB Transmit FIFO / 2KB Receive FIFO

MAC + PHY…

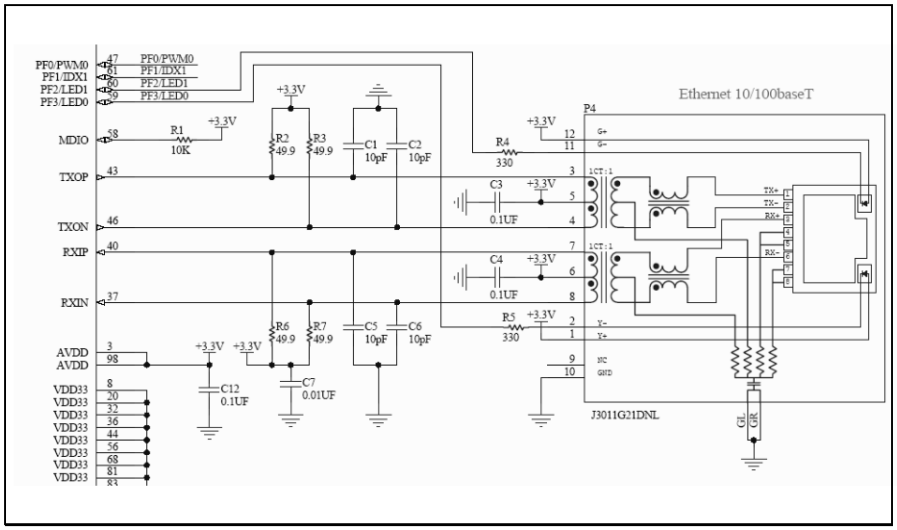
Stellaris Ethernet MAC and PHY




Stellaris Ethernet MAC and PHY
80 00 20 7A 3F 3E
Destination MAC Address
80 00 20 20 3A AE
Source MAC Address
08 00
EtherType
MAC Header
(14 bytes)
IP, ARP, etc.
Payload
Data
(46 - 1500 bytes)
00 20 20 3A
CRC Checksum
(4 bytes)
Ethernet Type II Frame
(64 to 1518 bytes)
Application layer
Transport layer
Network/Internet layer
Data link layer
Physical layer
DHCP, DNS, FTP, HTTP, ...
TCP, ...
IP (IPv4, IPv6), ...
Ethernet, 802.11 (WLAN), ...
Ethernet physical layer, ...
Ethernet & network layers
Stellaris
Cortex M3
Ethernet Controller
Media Access Controller
MAC (Layer 2)
Physical Layer Entity
PHY (Layer 1)
Magnetics
RJ45
Ethernet system on Stellaris
HW Design...

Hardware Design

Stellaris Ethernet Hardware Design



Note: VCC PHY pins are not labeled in the reference design, but are needed

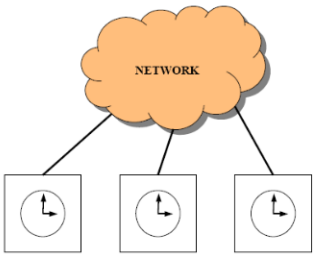
Required crystal not shown

PTP ...

IEEE1588 Precision Time Protocol

Ethernet - IEEE1588 PTP

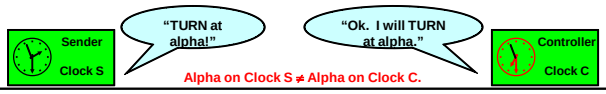
- IEEE 1588 is "Precision Clock Synchronization Protocol for Network and Control Systems" or Precision Time Protocol (PTP)
- IEEE 1588 is a protocol designed to synchronize real-time clocks in the nodes of a distributed system that communicate using a network (Ethernet) at a high degree of accuracy
- Microsecond accuracy is easily achievable using low cost, small footprint implementations such as Stellaris



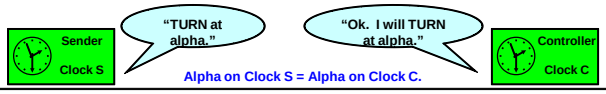
Visualizing IEEE 1588

- **Before IEEE 1588, Ethernet communication in control applications occurred without absolute determinism:**
  - **Assume** *Sender* **sends a control instruction** *Turn* **to** *Controller*
  - **Assume also that** *Clock S* **and** *Clock C* **are not synchronized**
  - **If** *Sender* **asks** *Controller* **to** *Turn* **upon receipt of the instruction, then there is no telling when** *Controller* **will receive** *Turn*.

  - **Even if** *Sender* **asks** *Controller* **to** *Turn* **at a given time** *alpha*, **there is still the problem of unsynchronized clocks.**



  - **But if** *Sender* **asks** *Controller* **to** *Turn* **at a given time** *alpha*, **and the clocks are synchronized to a master, then determinism is achieved**



PTP in Industrial Applications

- Industry synchronization requirements for PTP Applications

Application area	Required synchronization accuracy
Low speed sensors (e.g. pressure, temperature)	Milliseconds
Common electro-mechanical devices (e.g. relays, breakers, solenoids, valves)	Milliseconds
General automation (e.g. materials handling, chemical processing)	Milliseconds
Precise motion control (e.g. high speed packaging, printing, robotics)	A few microseconds
High speed electrical devices (e.g. synchrophasor measurements)	Microseconds
Electronic ranging (e.g. fault detection, triangulation)	Sub microsecond

- PTP and motion control
  - Variable frequency drives require few 10s of microseconds
    - Software generally 5uS
    - 44% of applications are networked, 63% use Ethernet TCP/IPB
  - Servo-controlled systems require 100s of nanoseconds
    - Requires significant hardware assist
    - 36% of applications are networked, 56% use Ethernet TCP/IPC
- Stellaris implementation
  - Open source lwIP + PTPd : within 500nS of master clock, jitter +/- 500nS
  - This represents a greater than ten fold improvement over typical SW-only implementations

Choosing a Stack ...

Ethernet Stacks

Choosing a Stack – Application Examples

- *"Transmission/Reception of information needs to be guaranteed."*
  - TCP over UDP
- *"Workers at each machine workstation to be able to monitor and control the workstation from the workstation's HMI computer."*
  - HTTP
- *"We need the ability to send system update files to each workstation from our main control room."*
  - FTP
- *"We want the system to automatically send an email if certain conditions are exceeded."*
  - ICMP
- *"We need our system to operate behind a firewall so that no one can hack in from the outside."*
  - NAT
- *"We need to be able to add more workstations to the network with ease."*
  - DHCP

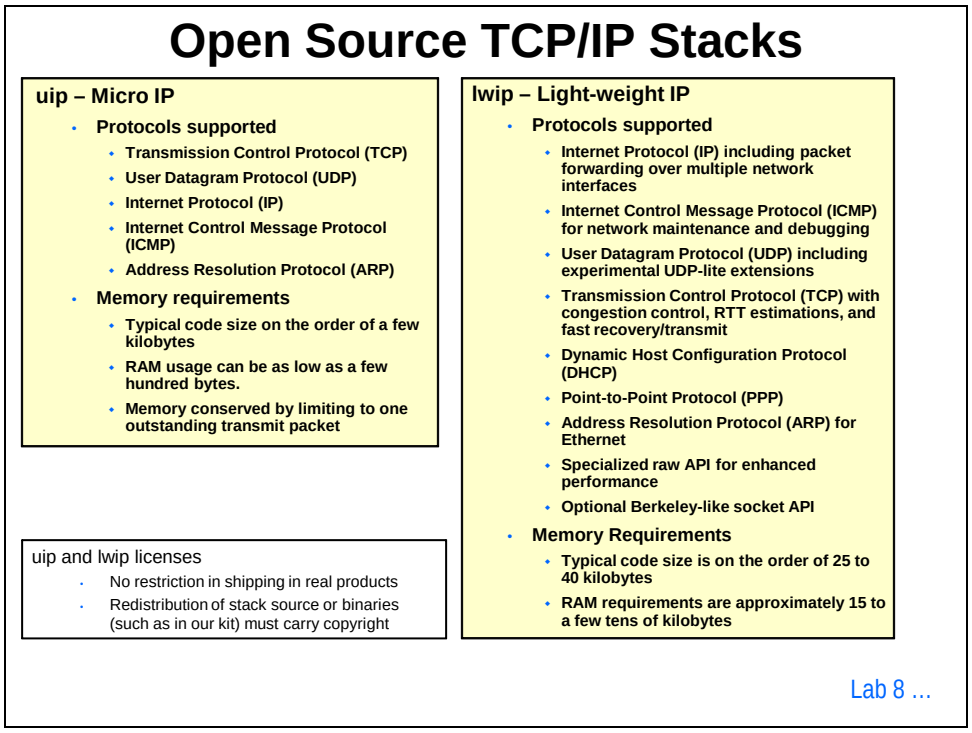
Available Stacks…

Communications Stacks for Stellaris®

TPV	Product	Stack	ARP	AutoIP	BOOTP	BSD	DHCP	DNS	FTP	HTTP	ICMP	IGMP	IKE	IP	IPSec	NAT	POP3	PPP	PTP	RARP	RIP	RTP	SLIP	SMTP	SNMP	SNTP	SSL	TCP	Telnet	TFTP	UDP	802.11
CMX Systems	CMX-MicroNET	TCP/IP	•		•						•	•		•				•					•					•			•	
CMX Systems	CMX Add Ons	Networking SW Options					•	•	•	•							•	•						•	•	•				•		•
Express Logic	NetX	TCP/IP	•								•	•		•						•								•			•	
Express Logic	NetX Add Ons	Networking SW Options		•		•	•	•	•	•						•	•	•						•	•	•			•	•		
Interniche	NicheLITE	TCP/IP	•		•		•	•			•			•														•		•	•	
Interniche	NicheStack	TCP/IP	•		•	•	•	•	•		•	•		•														•	•	•	•	
Interniche	Interniche Add Ons	Networking SW Options					•	•	•	•			•		•	•	•	•			•	•		•	•	•	•		•			
Micriμm	μC/UDP-IP	UDP/IP	•			•					•			•																	•	
Micriμm	μC/TCP-IP	TCP/IP	•			•					•			•														•			•	
Micriμm	Micriμm Add Ons	Networking SW Options					•	•	•	•							•							•		•			•	•		
SEVENSTAX	SEVENSTAX TCP/IP	TCP/IP									•															•		•			•	
SEVENSTAX	SEVENSTAX Add Ons	Networking SW Options	•		•		•	•		•				•				•	•					•								
SEGGER	embOS/IP	TCP/IP	•		•		•	•	•	•	•	•		•				•			•			•			•	•	•		•	
uIP	open source	TCP/IP	•								•		•	•														•			•	
lwIP	open source	TCP/IP	•	•			•	•			•	•		•				•								•		•			•	

Open Source Stacks…

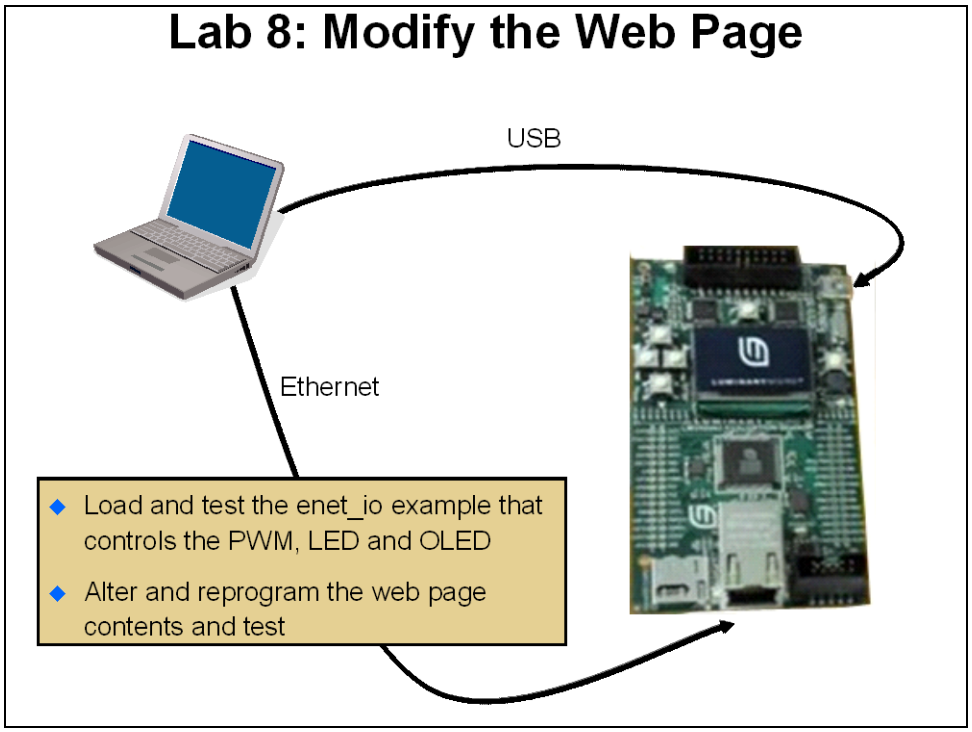
Open Source Ethernet Stacks



Lab8: Ethernet Lab

Description:

We'll use the LM3S8962 evaluation board and connect it to our PC with an Ethernet cable. We'll take a look at the lwip stack and the I/O web server. In the lab, we'll alter the contents of the served web page and reprogram it into the 8962's flash memory.



Procedure

Open Code Composer

1. Maximize Code Composer. Click on **Project**, and then select **Import Existing CCS/CCE Eclipse Project**. When the **Import** dialog appears, browse to **C:\StellarisWare\driverlib\ccs-cm3** and click **OK**. Be sure that the checkbox next to **driverlib-cm3** in the Discovered projects pane is **checked** and that **Copy projects into workspace** is **checked**. Click **Finish**.

2. Click on **Project** again, and then select **Import Existing CCS/CCE Eclipse Project**. When the **Import** dialog appears, browse to **C:\StellarisWare\boards\ek-lm3s8962\enet_io** and click **OK**. Be sure that the checkbox next to **enet_io** in the Project pane is **checked**. Click **Finish**.

3. **Maximize** the IDE window if you haven't already and expand the enet_io project by clicking on the + to the left of the name. Double-click on `enet_io.c` to open it for editing.

   Read the example description starting around line number 50. This code handles both IO control demos.

enet_io.c is made up of several modules:	
ControlCGIHandler()	Called when the web browser requests LED or PWM control
SetTextCGIHandler()	Called when the web browser requests to write test to OLED
SSIHandler()	Called by the HTTP server when it encounters an SSI tag
DisplayIPAddress()	Displays the lwIP type IP address
SysTickIntHandler()	Handles the SysTick interrupt
lwIPHostTimerHandler()	Supports host timer functions
main()	Sets up the clock, ethernet and IO ports, inits the OLED, configures SysTick and enables interrupts

4. Before we change anything, let's make sure that the project works as it is.

   Connect the LM3S8962 board to your PC via the USB cable, as usual. Then, using the Ethernet cable included in the kit, **connect** the LM3S8962 board to the Ethernet port of your PC. If your PCs wireless connection is enabled, you should **disable** it now.

   If you have problems connecting to the board in later steps, you may need to disable your firewall software.

   Click the  **Debug** button now to build and load the project to your board.

5. When the build and load completes, make sure that you are in the **Debug perspective**. **Click** the **Run** button to run the code on your board. You should see **"Web-Based I/O Control"** on the OLED display on the LM3S8962 board.

   Minimize Code Composer Studio.

   After 30 seconds or so, you should see the following on the OLED display (your IP address may be different):

   Web-Based I/O Control
   IP: 169.254.254.109
   MASK: 255.255.0.0
   GW: 0.0.0.0

6. There are two possibilities here: either your PCs Ethernet port and the board have compatible addresses, or they don't. To find out, we need the IP address of your Ethernet port. By the way, if you have Wi-Fi or 3G wireless on your machine, now would be a good time to disable it.

   In Windows XP, click **Start → Control Panel → Network Connections → Local Area Connection** and then click the Support tab to see your IP address.

   In Windows7, click **Start → Control Panel → Network and Internet → Network and Sharing Center → Change adapter settings → Local Area Connection → Details** and look for the IPv4 Address.

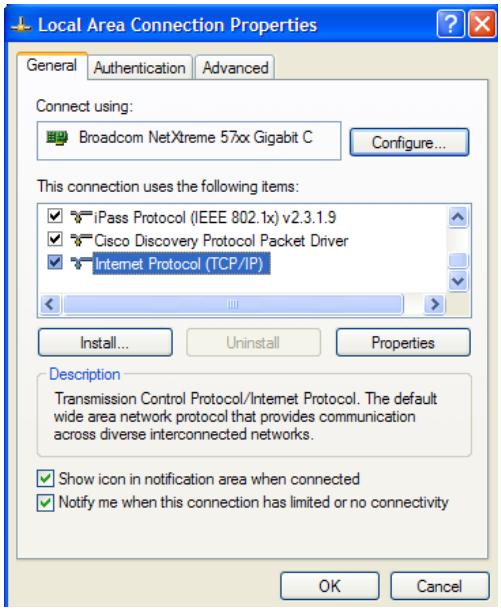
   If the first **three** parts of your IP address are the **same** as the address shown on the LM3S8962 OLED display, you got lucky! The addresses are compatible and you can skip to **step 9**.

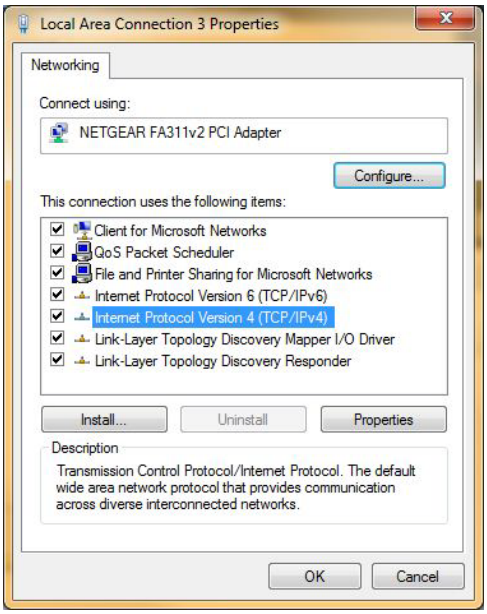
7. The **lwIP** stack has defaulted to a preprogrammed IP address. In order to communicate with that address, we need to give the PC's Ethernet port a compatible address.

   Your screen should still be displaying the Local Area Connections Status (or Details) from step 6.

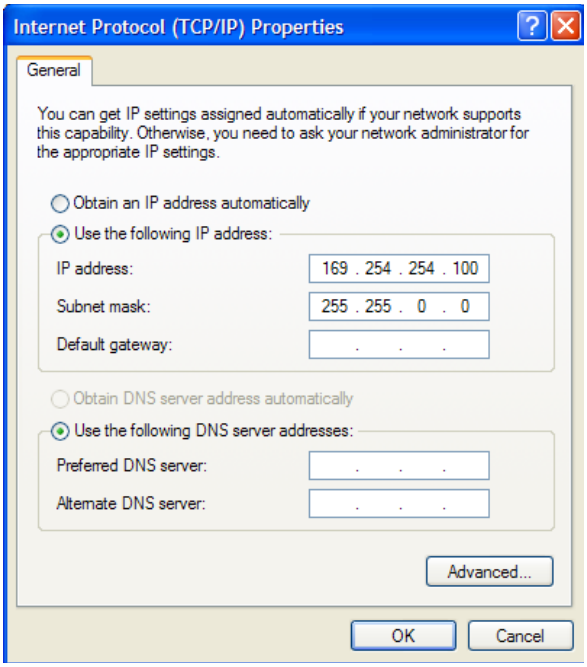
   In Windows XP, click the **General tab**, then **Properties**. Scroll down until you see **Internet Protocol (TCP/IP)** like shown on the left below and click on it.

   In Windows 7, close the Network Connection Details window and click the **Properties** button. Scroll down until you see the Internet Protocol Version 4(TCP/IPv4) like shown on the right below and click on it.





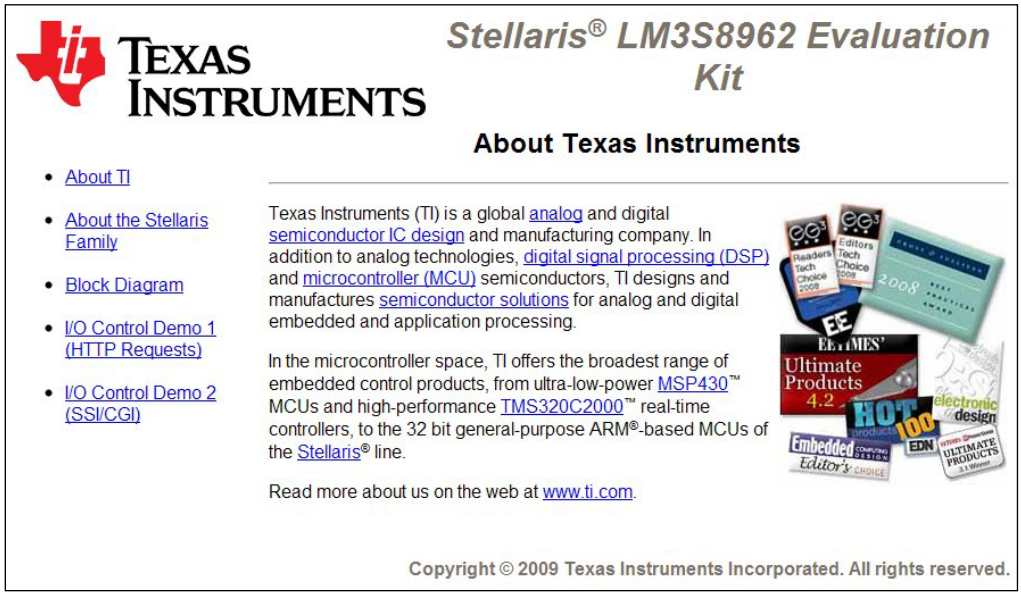
8. Click the **Properties** button. When the **Internet Protocol (TCP/IP) Properties** window appears, make a note of the settings you see in the space provided below, then make the selections shown below (if your boards' IP address was different, simply make sure that the **first three fields are the same, and the fourth is different**):



Note your previous settings here:

Click **OK**. Close all the network status windows.

9. Start your web browser, and enter the address shown on your OLED display and press **Enter**. A web page served from the LM3S8962 should appear.



If you are having issues seeing the web page on your browser, you may have one or more of the following going on:

1) You typed the IP address incorrectly for either your Ethernet port or the board. Remember that the Ethernet port's address and the board address cannot be the same.

2) Your firewall software is getting in the way … disable it for now.

3) You didn't disable your wireless ports and your browser is trying to access the address over that connection instead of the wired Ethernet connection.

4) You may not have the Java Runtime Engine installed. Go to www.java.com and install the JRE.

10. On the web page, **click** the link to **I/O Control Demo 1** and then press the **Toggle LED** button a few times, noting the LED on the LM3S8962 board. You may also note the green activity light on the LM3S8962 Ethernet connector as you press the button. Try the other controls, but try not to annoy everyone with the PWM/speaker output.

    On the web page, **click** the link to **I/O Control Demo 2**. Type the text of your choice (keep it clean) in the **Display this text on the screen:** box and click **Send Text**. The text that you typed should appear on the OLED screen. Try the other controls.

    When you're done fooling around, close your web browser.

11. The embedded web server used in the **enet_io** example uses the open source **lwIP TCP/IP** stack. When you first start the application, the **index.htm** file is displayed in your web browser.

    In this part of the lab, we will modify this web page using notepad as our editor. The easiest way to provide the html files to the application is by putting them on a microSD card. The enet_io application is coded to look for the file system there first. In the workshop, we're not using a microSD card. So instead, we will create a new file system image to embed in the application itself. There is a command line tool in the **\StellarisWare\tools\bin** folder that will generate a header file with an array for each file in the **\fs** folder.

    Using **Windows Explorer**, find **index.htm** file in the **C:\StellarisWare\boards\ek-lm3s8962\enet_io\fs** folder. Right click on **index.htm** and select **Open with**, then **click Notepad** to open the file for editing using Notepad.

12. **Find** the lines near the bottom of the file that look like this:

```
</p>
<p>Read more about us on the web at <a href="http://www.ti.com" target="_top">www.ti.com</a>.
<p></p>
<br>
```

    Add some code so that it looks like this:

```
</p>
<p>Read more about us on the web at <a href="http://www.ti.com" target="_top">www.ti.com</a>.
<p></p>
</p>
<p>WooHoo! YourName was here!
<p></p>
<br>
```

    Save the file and **close** your editor.

13. Convert the HTML files to a Header File

In Code Composer Studio, examine the file **lmi_fs.c** in the **enet_io** project. You will find the command line and options that are needed to run the **makefsfile** utility in the comments towards the top of this file.

Open a DOS command prompt window using **Start → Run**, type **cmd** and click **OK**.

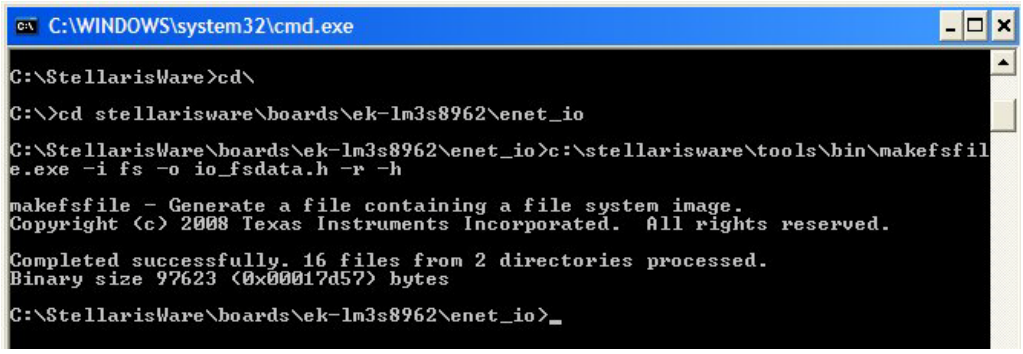
Type **cd** and press **Enter** to return to the root directory **C:**

Then type **cd stellarisware\boards\ek-lm3s8962\enet_io** and press **Enter**.

In the command window, type the command shown below to call the **makefsfile** utility.

C:\StellarisWare\tools\bin\makefsfile.exe -i fs -o io_fsdata.h -r -h

This will create a new **io_fsdata.h** header file, with the changes that you made, which is included by **io_fs.c** . **Close** the command window when you are finished.



13. Rebuild the enet_io Example Application

Maximize Code Composer Studio. We just modified one of the files in the project without the IDE knowing it, so we need to perform a clean build.

Return to the **C/C++ perspective**. Right-click on **enet_io** in the **Project Explorer** window and select **Clean Project**. **Click** the **Debug** button to build/load the project.

Make sure your LM3S8962 evaluation board is connected and assure that CCS is in the **Debug** perspective.

14. Load the Modified Website in your Browser

Make sure your Ethernet cable is connected and **reset** the LM3S8962 evaluation board (**disconnect/reconnect** the USB cable). Wait for the IP address to show up on your OLED, make sure that it is still compatible with your PCs IP address and type it into address bar:

Restore your network settings

Remember your original network settings on your PC? **Restore** those and re-enable your wireless connection (if necessary). **Terminate the Debug session, close** the enet_io project and **exit** CCS. **Close** any other open windows on your desktop.

You're done

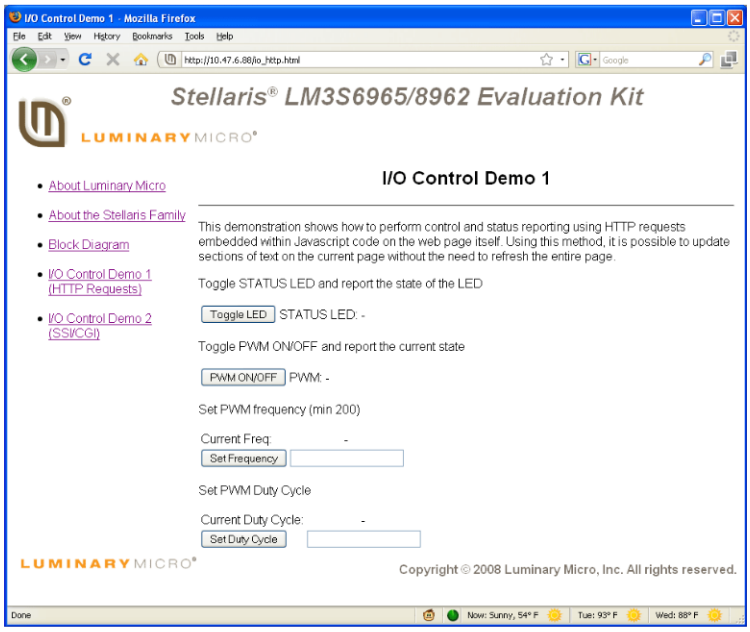
Additional Web Server Information

The following material is for your information and will not be covered during the regular workshop session.

I/O Control Demonstrations

- The enet_io application illustrates two methods of controlling board operations from the web browser:
  - I/O Control Demo 1 shows direct HTTP requests generated via JavaScript code in the web page (io_http.html).
  - I/O Control Demo 2 shows the use of Server Side Includes (SSI) and Common Gateway Interface (CGI) to perform the same operations (io_cgi.shtml)
- All web site files are stored as a file system image (io_fsdata.c) linked into the application image in flash.

I/O Control Demo 1



Demo 1– HTTP Requests

- JavaScript in the web page generates HTTP GET requests for specific filenames.
- lwIP web server passes these filenames to the file system (`lmi_fs.c`) which recognizes them as "special" and performs whichever task the filename is intended to trigger.
  - Toggle LED state (`/cgi-bin/toggle_led`)
  - Set and report PWM frequency (`/pwm_freq?value=`<frequency>)
- JavaScript reads the response from the web server and uses `<div>` tags to insert the relevant text into the displayed page.

Demo 1– HTTP Requests

- **Advantages:**
  - Updates can be made without having to reload the whole web page.
  - Quicker user feedback.
  - Lower network traffic.
- **Disadvantages:**
  - Client browser must support JavaScript.
  - HTML files are more complex to develop and less easy to understand.

Embedded Control

- **Below is an example of the html code used for the Toggle LED button from "io_http.html"**

```
<table>
  <tr>
    <td>
      <input id="toggle" value="Toggle LED"
       onclick="toggle_led()" type="button">
    </td>
    <td>
      STATUS LED:
    </td>
    <td>
      <div id="ledstate" align="center"> - </div>
    </td>
  </tr>
</table>
```

Embedded Control

- **Below is an example of the JavaScript code used for the Toggle LED button from "io_http.html"**

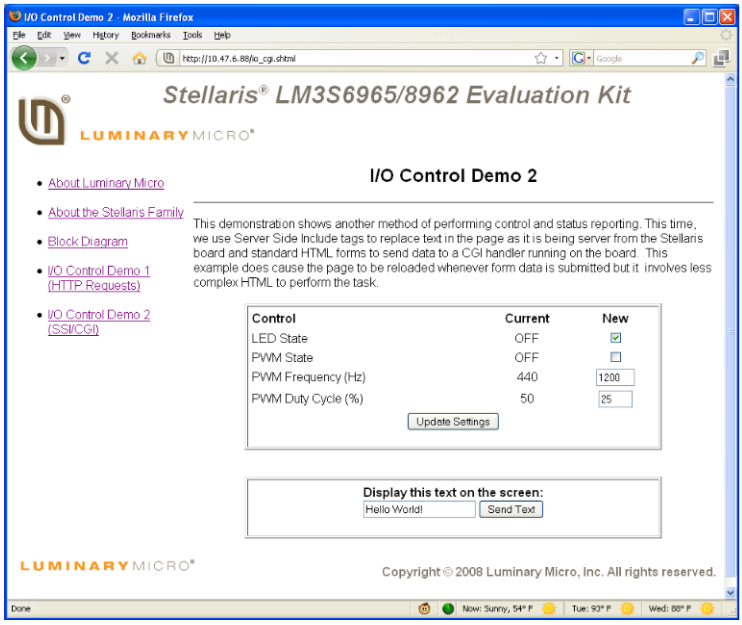
```
function toggle_led()
{
  var req = false;
  var led = false;

  function ledComplete() {
    if(led.readyState == 4) {
      if(led.status == 200) {
        document.getElementById("ledstate").innerHTML = "<div>" + led.responseText +
                                "</div>";
      }
    }
  }
  if(window.XMLHttpRequest){
    req = new XMLHttpRequest();
    led = new XMLHttpRequest();
  }
  else if(window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
    led = new ActiveXObject("Microsoft.XMLHTTP");
  }
  if(req){
    req.open("GET", "/cgi-bin/toggle_led?id" + Math.random(), true);
    req.send(null);
  }
  if(led){
    led.open("GET", "/ledstate?id=" + Math.random(), true);
    led.onreadystatechange = ledComplete; led.send(null);
  }
}
```

Embedded Control

- **Below is an example of the C source code from lmi-fs.c that handles the request for toggling the LED**

```
//
// Process request to toggle STATUS LED
//
if(strncmp(name, "/cgi-bin/toggle_led", 19) == 0)
{
    //
    // Toggle the STATUS LED
    //
    io_set_led(!io_is_led_on());

    //
    // Setup the file structure to return whatever.
    //
    ptFile->data = NULL;
    ptFile->len = 0;
    ptFile->index = 0;
    ptFile->pextension = NULL;

    //
    // Return the file system pointer.
    //
    return(ptFile);
}
```

Demo 2 – SSI/CGI



Demo 2 – SSI/CGI

- HTML pages include “Server Side Include” tags indicating values to be inserted in the page data as it is served to the browser.
- The application registers SSI and CGI handlers with the HTTP server during initialization.
- The HTTP server calls the SSI handler when a tag from the registered list is detected and handler returns the text to insert after the tag.
- The HTTP server calls the registered CGI handler if a URL matching the registered CGI name is requested.
- The HTML contains standard forms to gather user input.

Demo 2 – SSI/CGI

- **Advantages**
  - Client browser need not support JavaScript.
  - HTML is extremely simple and uses only standard forms and some “comment-like” SSI tags (<!--#<tag>-->).
  - Offloads work to the common HTTP server module (URL checking, parameter parsing).
  - File system driver is independent of the application data that it is managing.
- **Disadvantages**
  - Page reload each time a form is submitted.

Demo 2 – SSI/CGI

- **Below is an example of the html code used to send a text string to the browser in "io_cgi.shtml"**

```
<form method="get" action="settxt.cgi" name="settxt">
<table>
  <tr>
    <td>
      <b>Display this text on the screen:<br></b>
      <input maxlength="20" size="20" name="DispText">
      <input name="Display" value="Send Text" type="submit">
    </td>
  </tr>
</table>
</form>
```

Demo 2 – SSI/CGI

- **Below is an example of the html code from "io_cgi.shtml" which uses SSI tags to insert status information.**

```
<tr>
  <td >LED State</td>
  <td>
    <!--#LEDtxt-->
  </td>
  <td>
  <input name="LEDOn" value="1" type="checkbox">
  </td>
</tr>
<tr>
  <td>PWM State</td>
  <td>
    <!--#PWMtxt-->
  </td>
  <td>
    <input name="PWMOn" value="1" type="checkbox">
  </td>
</tr>
```

The HTTP server parses the SSI tags from the page as it is being sent and passes them to the application SSI handler.

The application returns an appropriate block of text which is inserted into the page following the comment tag.

Note: The tag is *not* replaced. The SSI text is inserted after the closing "-->" This is different from typical SSI implementations but allows operation without additional buffering.

"`<!--#LEDtext-->`" in the source is expanded to, for example "`<!--#LEDtext-->OFF`"

Registering SSI Handlers

```
// SSI tag indices for each entry in the g_pcSSITags array.
#define SSI_INDEX_PWMDUTY   0
#define SSI_INDEX_PWMSTATE  1

// This array holds all the strings that are to be recognized as SSI tag
// names by the HTTPD server.  The server will call SSIHandler to request a
// replacement string whenever the pattern <!--#tagname--> (where tagname
// appears in the following array) is found in ".ssi", ".shtml" or ".shtm"
// files that it serves.
static const char *g_pcConfigSSITags[] =
{
    "PWMDuty",       // SSI_INDEX_PWMDUTY
    "PWMtxt"         // SSI_INDEX_PWMSTATE
};

// The number of individual SSI tags that the HTTPD server can expect to
// find in our configuration pages.
#define NUM_CONFIG_SSI_TAGS     (sizeof(g_pcConfigSSITags) / sizeof (char *))

// Prototype for the main handler used to process server-side-includes for the
// application's web-based configuration screens.
static int SSIHandler(int iIndex, char *pcInsert, int iInsertLen);
```

After initializing the HTTPD server, register the handler by calling…

```
// Register our SSI tags and handler with the HTTP server.
http_set_ssi_handler(SSIHandler, g_pcConfigSSITags, NUM_CONFIG_SSI_TAGS);
```

The SSI Handler Function

```
// This function is called by the HTTP server whenever it encounters an SSI
// tag in a web page.  The iIndex parameter provides the index of the tag in
// the g_pcConfigSSITags array. This function writes the substitution text
// into the pcInsert array, writing no more than iInsertLen characters.

static int SSIHandler(int iIndex, char *pcInsert, int iInsertLen)
{
    unsigned long ulVal;

    // Which SSI tag have we been passed?
    switch(iIndex)
    {
    case SSI_INDEX_PWMSTATE:
// Write the PWM state (ON or OFF) into the supplied insert buffer.
        io_get_pwmstate(pcInsert, iInsertLen);
        break;

    case SSI_INDEX_PWMDUTY:
      // Get the current PWM duty cycle.
      ulVal = io_get_pwmdutycycle();

  // Write it as an ASCII string into the supplied insert buffer.
      usnprintf(pcInsert, iInsertLen, "%d", ulVal);
      break;
    }

  // Tell the server how many characters our insert string contains.
    return(strlen(pcInsert));
}
```

Registering CGI Handlers

```
// Prototypes for the various CGI handler functions.
static char *ControlCGIHandler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]);
static char *SetTextCGIHandler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]);

// CGI URI indices for each entry in the g_psConfigCGIURIs array.
#define CGI_INDEX_CONTROL       0
#define CGI_INDEX_TEXT          1

// This array is passed to the HTTPD server to inform it of special URIs
// that are treated as common gateway interface (CGI) scripts.  Each URI name
// is defined along with a pointer to the function which is to be called to
// process it.
static const tCGI g_psConfigCGIURIs[] =
{
    { "/iocontrol.cgi", ControlCGIHandler },      // CGI_INDEX_CONTROL
    { "/settxt.cgi",    SetTextCGIHandler }       // CGI_INDEX_TEXT
};

// The number of individual CGI URIs that are configured for this system.
#define NUM_CONFIG_CGI_URIS     (sizeof(g_psConfigCGIURIs) / sizeof(tCGI))
```

After initializing the HTTPD server, register the handlers by calling…

```
                        // Register our CGI handlers with the HTTP server.
                    http_set_cgi_handlers(g_psConfigGGIURIs, NUM_CONFIG_CGI_TAGS);
```

A CGI Handler Function

```
// This CGI handler is called whenever the web browser requests URI /iocontrol.cgi.
static char *ControlCGIHandler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[])
{
    tBoolean bParamError;
    long lPWMState,lPWMFrequency;

    // We have not encountered any parameter errors yet.
    bParamError = false;

    // Get each of the expected parameters.
    lPWMState     = FindCGIParameter("PWMOn", pcParam, iNumParams);
    lPWMFrequency = GetCGIParam("PWMFrequency", pcParam, pcValue, iNumParams,
                                &bParamError);

    // Was there any error reported by the parameter parser?
    if(bParamError || (lPWMFrequency < 200) || (lPWMFrequency > 20000))
    {
        // Return the URI of the parameter error page.
        return("/perror.html");
    }

    // We got all the parameters and the values were within the expected ranges
    // so go ahead and make the changes.
    io_pwm_freq((unsigned long)lPWMFrequency);
    io_set_pwm((lPWMState == -1) ? false : true);

    // Send back the default response page.
    return("/io_cgi.shtml");
}
```

Appendix

This page contains some additional hints and tips.

When CCS asked me for my workspace, I accidentally checked the "Use this as the default and do not ask again" checkbox. How can I unchecked that box?

You can always switch workspaces in CCS using File → Switch Workspace. But you can return the startup behavior to normal like this: On the CCS tool bar, click Window → Preferences → the + next to General → the + next to Startup and Shutdown → Workspaces. Then check the Prompt for workspace on startup checkbox.

Sometimes when I rename a file, like to main.c, Windows thinks it's a text file. What's up?

Windows can hide extensions for known file types. Then when you rename a file, it becomes main.c.txt without you knowing it. The fix is simple…

In Windows XP:

Click on Start → Control Panel → Folder Options → View and uncheck "hide extensions for known file types" and click OK.

In Windows 7:

Click on Start, In the "Search programs and files" box, type "folder options". Press Enter. Click the View tab. Uncheck "hide extensions for known file types" and click OK.

In the Debug perspective in Code Composer, my Run, Halt, Terminate buttons have disappeared.

The buttons are actually part of the Debug window, not the Debug perspective. In the Debug perspective, click View → Debug.