Workshop Setup

The software/hardware used in this workshop are:

- CCSv5.3.0.00090

- StellarisWare version 9453 (Sep 2012)
  http://www.ti.com/tool/sw-lm3s

- Stellaris Launchpad Evaluation Kit (EK-LM4F120XL) with USB cable connected from Debug USB Port to PC

Notes:
1. If you are using different versions of software, some lab steps may require slight modifications and/or screenshots may look a bit different
2. If the labs are not being done in a class environment where the software is already installed, install the above listed software first
LAB conventions

• Lab steps are numbered for easier reference
  1. ...
  2. ...

• Explanations, notes, warnings are written in blue
Blinky Example: Exercise Summary

• **Key Objectives**
  – Import and build a simple program to blink LED
  – Start a debug session and load/flash the program to the Launchpad
  – Run the program to blink LED

• **Tools and Concepts Covered**
  – Workspaces
  – Welcome screen / Resource Explorer
  – Project concepts
  – Basics of working with views
  – Debug launch
  – Debug control
  – Profiling Clock Cycles
  – Local History
  – Build Properties
  – Updating compiler tools
  – Changing compiler versions
Import ‘blinky’ Project

1. Import the blinky project into the CCS Workspace by going to menu Project → Import Existing CCS Eclipse Project

2. In the box Select search-directory, type or browse to the StellarisWare directory: C:\StellarisWare\boards\ek-lm4f120xl

3. Wait until the tool finishes discovering the available projects.

4. Select the project “blinky”

5. Click Finish
Build ‘blinky’ Project

1. In the *Project Explorer* view, select the blinky project (it should appear [Active – Debug]) to make it active.

2. Click on the Build icon in the toolbar.

3. The *Console* view will appear at the bottom with build messages (information, warnings, errors) as the project builds.

4. The *Problems* view will also appear at the bottom to highlight any possible build errors or warnings.

5. If the build is successful, the *Problems* view will contain no errors (warnings may still be seen).

In this case, there may be a warning that this project was originally created using a version of the compiler that is currently not installed.
Debug ‘blinky’ Project

1. Click on the “green bug” button – make sure the project is selected first!

   When you hit the green bug button, several actions are done automatically
   – Prompt to save modified source files
   – Build the project (incrementally)
   – Start the debugger (CCS will switch to the CCS Debug perspective)
   – Connect CCS to the target
   – Load the program on the target
   – Run to main()
Debug ‘blinky’ Project

Switched to CCS Debug perspective

Program counter at main()
1. Press the *Run* button in *Debug* view to run the program

LED on the Launchpad should now be blinking
Debugging: Using Watchpoints

A Watchpoint is a type of breakpoint that monitors activity on a memory address.

In this step we will set a watchpoint to halt the CPU every time the LED toggles.

1. Press the *halt/suspend* button to halt the running program
   - The code should stop somewhere in the *for()* loop

2. Open the *Breakpoints* view
   - View → Breakpoints

3. Create a new *Hardware Watchpoint*
   - *Location*: 0x400253FC (*GPIO_PORTF_DATA* register)
   - *Memory*: Write

4. Click OK
Debugging: Using Watchpoints

1. Run the target again.

2. Execution will halt each time LED is toggled, (ie) whenever GPIO_PORTF_DATA register is written to
More Debugging

Investigate other debugging views (Open via View menu)

- Memory Browser
- Registers
- Disassembly (see next slide)

Set breakpoints

- Double click on a source line to set/clear
- See list of breakpoints with the Breakpoints view

Use the buttons in the Debug view to:

- Restart the program
- Source stepping
- Assembly stepping
View: Disassembly

1. Open the disassembly view by going to View->Disassembly
   You can see the current location of the PC (small blue arrow) and any breakpoints (small blue circles)

2. Toggle the Show Source button. Note the toggling of interleaved source with the disassembly
Debugging: Measuring Clock Cycles

The profile clock (used on most other TI devices to count cycles) is disabled on Stellaris Cortex M3/M4F due to hardware limitations, but Count Event can be configured to count cycles.

1. In the Breakpoints view, create a Count Event

2. Select Event to Count to be Clock Cycles
Debugging: Measuring Clock Cycles

The Count Event works by counting cycles between breakpoints, so we will create two hardware breakpoints and disable the previously created watchpoint.

1. Enable breakpoint at line lines 73 and 85 of blinky.c by double-clicking on the left editor margin.

2. Disable the previously created watchpoint (click in the checkbox to uncheck it).

3. Click the Run button.
   - Clock should now show ~2000007 cycles.
Increase LED Blink Rate

The blink rate of LED can be increased by changing the delay value in the loop

1. Modify lines 78 and 90 of `blinky.c`
   - From: `for(ulLoop = 0; ulLoop < 200000; ulLoop++)`
   - To: `for(ulLoop = 0; ulLoop < 400000; ulLoop++)`

2. Save `blinky.c`

3. Disable the two breakpoints set before at lines 73 and 85! Go to the Breakpoints view and uncheck the checkboxes
**View: Local History**

**CCS keeps a local history of source changes**
1. Switch to the *CCS Edit* perspective
2. Right-click on a file in the editor and select *Team → Show Local History*

You can compare your current source file against any previous version or replace it with any previous version
1. Double-click on a revision to open it in the editor
2. Right-click on a revision and select *Compare Current with Local* to compare that revision to the current version

**CCS also keeps project history**
1. Recover files deleted from the project
2. Right-click on the project and select *Recover from Local History* in the context menu
Rebuild Project

1. Make sure you are in CCS Edit perspective and project blinky is active

2. Rebuild the project
   - Menu Project → Build Project

   CCS will automatically detect that the program currently being debugged has changed/rebuilt and ask if it should reload the file

3. Select Yes and CCS will reload/reflash the program and run to main()
Blink LED

1. Switch to the *CCS Debug* perspective

2. Press the *Run* button 🇺🇸 to run the program
   - The LED on the Launchpad should now be blinking at a slower rate
   - If it is not blinking, remember to *disable* (not delete) the breakpoints set in the previous session! 😊
Debugging: Measuring Clock Cycles

1. Pause execution of the program.
2. In Breakpoint view, right-click on Count Event and go to Breakpoint Properties.
3. Set Reset Count on Run to “true” to reset the cycle count upon each run.
4. Click OK.
5. Enable the two breakpoints at lines 73 and 85 added earlier.
6. Click the Run button to halt at one of the lines that modify GPIO_PORTF_DATA_R.
7. Click Run again.
   - Clock should now show ~ 4000012 or 4400011 cycles (about 2x the values seen earlier).
Terminate the Debug Session

1. In the Breakpoint view, **delete** (not disable) the watchpoint, count event and breakpoints as they will not be used in the next lab – simply highlight them and press the <Delete> key.

2. In Debug View, click on the Terminate button.

3. This will terminate the debugger and return you to the **CCS Edit** perspective.
**Project Properties**

1. Make sure you are in the *CCS Edit* perspective
2. Right click on the blinky project and select *Properties*
3. General Settings and Compiler and Linker options can be set from here

![Image of project properties settings in CCS](image.png)

- **Device and high level settings**
- **Compiler Options**
- **Linker Options**
- **Version of compiler tools used to create original project**
- **Version of compiler tools used by CCS for current build**
Update Compiler Tools

Internet connection is required to install updates using CCS update manager
Note: this compiler update will already be installed for hands-on workshops in class environment

1. Go to Help->Check for Updates

2. In the list of Available Updates, Deselect All and then select the latest version of ARM Compiler Tools, in this case 5.0.3

3. Click Next and again Next

4. Accept terms of license agreement

5. Click Finish

6. Restart CCS when it prompts for restart and go through installation
Change Compiler Version

1. Make sure you are in the CCS Edit perspective

2. Right click on the blinky project and select Properties

3. Notice that Effective compiler version is set to TI v5.0.3
   - CCS will default to use the latest version installed

4. Click on the More… button beside Compiler version TI v4.9.5

5. Notice the two compiler versions are available to select

6. Click Cancel to leave settings as-is
Optional: Change Build Options

Build options are set per build configuration

1. Change your Configuration to *Release*

2. Change the optimization settings
   - Go to the *Build → ARM Compiler → Optimization*
   - Change the optimization level to 4

3. Click OK
Optional: Change Build Configuration

1. **Change the active configuration to Release**
   - Right click on the Project
   - Select *Build Configurations* ➔ *Set Active* ➔ *Release*

2. **Build the project by clicking the build button**
   - In the console view you will see that the *Release* configuration has been built

You can also change the configuration and build it by clicking on the arrow beside the build button and selecting the configuration you want to build
   - Select *Release* and it will build this configuration
   - The active configuration is indicated by the Checkmark
Optional: Load the optimized code

1. After building the code, hit the “green bug” button

2. Press the Run button in Debug view to run the program
   LED on the Launchpad should now be blinking
Blinky Example: Summary

- In this lab we completed the following:
  - Imported and built a simple program to blink the on-board LED
  - Started a debug session and flashed the program to the Launchpad
  - Ran the program to blink LED
  - Used data watchpoints to halt CPU during each LED toggle
  - Measured clock cycles
  - Studied Project Build Properties
  - Updated version of compiler tools
  - Changed compiler version and compiler options in the project and rebuilt/re-ran the code
Hello Example (Portable Project): Exercise Summary

• Key Objectives
  – Create a new portable project based on the “hello” example
  – Create workspace level variables for the project
  – Link files to the project using variables
  – Configure build properties using variables
  – Validate project by building, loading and running the program

• Tools and Concepts Covered
  – Portable Projects
  – Linked resources
  – Linked resource path variables
  – Build variables
Create a New Project

1. Launch the **New CCS Project Wizard**
   - Go to menu *File ➔ New ➔ CCS Project*

2. Fill in the fields as shown in the right
   - **Project name:** *hello*
   - **Device ➔ Family:** *ARM*
   - **Variant:** *Stellaris LM4F120H5QR*
   - **Connection:** *Stellaris In-Circuit Debug Interface*

3. In **Project templates and examples**, select **Empty Project**

4. Click *Finish* when done
   - Generated project will appear in the Project Explorer view

5. Expand the project to view contents

6. Remove the file lm4f120h5qr.cmd from the project (select it and press <Delete> key)
Create a Linked Resource Path Variable

Here we will create the Linked Resource Path Variable which will be used when linking source files (resources) to the project.

1. Open the workspace preferences
   - Menu Window → Preferences

2. Go to the Linked Resources preferences
   - Type ‘Linked’ in the filter field to make it easier to find

3. Use the New button to create a ‘Linked Resource Variable’ (named SW_ROOT) that points to the root location of the StellarisWare directory:
   C:\StellarisWare

4. Click OK
Create a Build Variable

Here we will create the Build Variable which will be used when setting the project’s compiler and linker options

1. Go to the Build Variables preferences
   - Type ‘Variables’ in the filter field to make it easier to find

2. Build Variables allow you to use variables in the project properties
   - Linked Resource variables are only used for linked files

3. Use the Add button to create a ‘Build Variable’ (named SW_ROOT) that points to the root location of the StellarisWare directory

4. Click OK and OK again
Link Source Files to Project

Here we will link source files relative to the Linked Resource Path Variable previously created

1. Open Windows Explorer and browse to:
   C:\StellarisWare\boards\ek-lm4f120xl\hello

2. Select the following files and drag and drop them into the “hello” project in the CCS Project Explorer view
   - hello.c, startup_ccs.c, hello_ccs.cmd
Link Source Files to Project

A dialog will appear asking if you wish to Copy or Link the files:

1. Select *Link to files*

2. Select *Create link locations relative to*:
   - Use the new Linked Resource variable we created (*SW_ROOT*)

3. Hit *OK*

Files will now appear in the Project Explorer with the ‘link’ icon
Follow the same procedure to link source file uartstdio.c

1. Open Windows Explorer and browse to: C:\StellarisWare\utils
2. Select “uartstdio.c” and drag and drop it into the “hello” project in the CCS Project Explorer view
3. In the dialog that appears, select Link to files
4. Select *Create link locations relative to:* 
   - Use the new Linked Resource variable we created (SW_ROOT)
5. Click OK
Link Files to Project

1. Right-click on source file “hello.c” and check the Properties.
   Notice how the Location parameter references the Linked Resource Path Variable.
Modifying Project Properties

Here we will add include file search paths using the Build Variable

1. Right-click on the project and select **Properties**

2. In the Compiler **Include Options**, add the following entry to the list of include search paths:
   
   `${SW_ROOT}`

3. Click **OK**

`'${<BUILD VARIABLE>}'` is the syntax to use a Build Variable in the project properties. Here we are setting an include path to the root of StellarisWare installation

Linked Resource Path Variables are only used when linking source files to a project. They cannot be used for build options. Use Build Variables when modifying build options
Modifying Project Properties

Here we will add paths to libraries using the Build Variable

1. Right-click on the project and select Properties
2. In the Linker File Search Path, click on the icon for --search_path option to add the following entry:
   ${SW_ROOT}/driverlib/ccs-cm4f/Debug
3. In the Linker File Search Path, click on the icon for --library option to add the following entry:
   driverlib-cm4f.lib
4. Click OK
Here we will add compiler options specific to this Stellaris device and project

1. Right-click on the project and select Properties

2. In the Compiler Advanced Options->Predefined Symbols, click on icon for --define option to add the following entries:
   - PART_LM4F120H5QR
   - TARGET_IS.BLIZZARD_RA1

3. In the Compiler Advanced Options->Language Options, enable the checkbox for Enable support for GCC extensions (--gcc)

4. Click OK
Project Properties

1. Go to Resource → Linked Resources to see all the Linked Resource Path Variables that are available to the project
   This will show all variables created at the project level and workspace level

2. Check if the workspace level Linked Resource Path Variable that was created appears in the list
   Variables may be edited here but changes will only be recorded at the project level (stored in the project files)
**Project Properties**

The *Linked Resources* tab will show all the files that have been linked to the project:
- It will sort them by files linked with a variable and files linked with an absolute path.

Links can be modified here with the *Edit…* button.
Links can be converted to use an absolute path with the *Convert…* button.
1. Go to *Build* → tab *Variables* to see all the Build Variables that are available to the project.
   Only project level variables will be listed by default.

2. Enable the *Show system variables* checkbox to see variables set at the workspace and system level.
   See how the workspace level Build Variable that was created appears in the list.
1. Make sure the ‘hello’ project is in focus. Then use the *Debug* button to build and load the code.
Seeing the output on terminal

A terminal program, such as PuTTY is used to view the output on the UART (UART connected to Stellaris Virtual Serial Port)

1. Start up PuTTY

2. Select the Connection type as “Serial”

3. Set the Serial line parameters are shown below and click Open
   - COM port number should be the appropriate one for your PC (Use Stellaris Virtual Serial Port COM# listed in Device Manager)
Seeing the output on terminal

4. Click the Run button in CCS. The PuTTY screen will show the string below:
   - Hello, world!

5. Click the Suspend button to pause the program

6. Click the Terminate button to terminate the debug session

You have now successfully created a “Hello World” portable projects that makes use of Linked Resource Path variables and Build variables!!
Hello Example (Portable Project): Summary

• **In this lab we completed the following:**
  – Created a new portable project
  – Created workspace level variables for the project – Linked Resource path variable and Build variable
  – Linked source files to the project using Linked Resource path variable
  – Configured include paths for compiler using Build variable
  – Validated project by building, loading and running the program