Code Composer Studio v5 Workshop
What is Code Composer Studio?

- **Integrated development environment for TI embedded processors**
  - Includes debugger, compiler, editor, operating system…
  - 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**
  - Going forward CCS 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…
Upgraded user interface for fast, intuitive and easy development

- **Simplified user interface** shows developers what and when features are needed.
- **Resource Explorer** facilitates use of example code.
- **Development tools** for Windows and now Linux operating systems.
- **Eclipse open source framework 3.7** enables customization via latest third-party plug-ins.
- **Video tutorials** explain how to get the most out of features.
Pricing

• **Free**
  - Time limited evaluation tools
  - Tied to development kits
  - When using ultra low cost XDS100

• **Node locked**
  - Full featured
  - Supports all processors
  - $495

• **Floating**
  - Full featured
  - Supports all processors
  - Starting from $795
TI Software Support

• Submitting issues
  – All questions, issues and enhancement requests should be submitted using the e2e Community Forums

• Benefits of forums
  – Connects users directly with the engineers developing & supporting TI products
  – Access an extensive knowledge base on TI products

• Software related forums
  – Development Tools
    • Any issues/questions related to Code Composer Studio (CCS) or TI compilers
  – Embedded Software
    • Linux, Android, WinCE, BIOS and Codecs forums

• Before posting a question check if it is already answered
  – Check the FAQs and topics on the Embedded Processing Mediawiki
  – Search the e2e forums

• Check status of issues
  – Use SDOWP to see what release an issue will be addressed as well as the list of issues fixed in specific releases
GETTING STARTED WITH CCSV5 AND STELLARIS LAUNCHPAD
What is Stellaris LM4F120 Launchpad?

- Low-cost (under $15) evaluation platform for ARM Cortex M4F-based microcontrollers
- Uses LM4F120H5QR microcontroller
- On-board Stellaris In-Circuit debug interface (ICDI)
What is StellarisWare?

- Extensive suite of software designed to simplify and speed up development
- Works with all LM3S and LM4F Stellaris MCUs
- Includes Peripheral Driver library, Graphics Library, USB Library and code examples
- Works with multiple development platforms (CCS, IAR, Code Red, Code Sourcery, ARM/Keil)
- Can be downloaded free at: http://www.ti.com/tool/sw-lm3s
BLINKY EXAMPLE: BASIC PROJECT DEBUGGING
Blink Example: Exercise Summary

• Key Objectives
  – Import and build a simple program to blink the on-board 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
Workspace

- Launching CCS for first time requires you to select a workspace folder
  - Defaults to your user folder
Eclipse Concept: Workspaces

- Main working folder for CCS
- Contains information to manage all the projects defined to it
  - The default location of any new projects created
- User preferences, custom perspectives, cached data for plug-ins, etc all stored in the workspace
- Multiple workspaces can be maintained
  - Only one can be active within each CCS instance
  - The same workspace cannot be shared by multiple running instances of CCS
  - It is not recommended to share workspaces amongst users
Eclipse Concept: Workbench

- *Workbench* refers to the main CCSv5 GUI window

- The Workbench contains all the various views and resources used for development and debug
Eclipse Concept: Projects

- Projects map to directories in the file system

- Files can be added or linked to the project
  - Adding file to project
    • Copies the file into your project folder
  - Linking file to project
    • Makes a reference to the file in your project
    • File stays in its original location

- Projects are either open or closed
  - Closed Projects:
    • Still defined to the workspace, but it cannot be modified by the Workbench
    • The resources of a closed project will not appear in the Workbench, but the resources still reside on the local file system
    • Closed projects require less memory and are not scanned during routine activity

- Projects that are not part of the workspace must be imported into the active workspace before they can be opened
  - Both CCSv4 and v5 projects can be imported into the workspace
Eclipse Concept: Views

- Views are windows within the main Workbench window that provide visual representation of some specific information
  - Most views can be accessed in the menu View
  - Can be identified by the organization in tabs
View: Project Explorer

• Displays all projects defined in the active workspace

• The view is mostly a representation of the file system of the project folder
  – Linked files are marked with a special link graphic in the icon

• Use filters to hide various file types to reduce clutter in the view
  – Default is to filter CCS generated project files (*.*)
Eclipse Concept: Focus

- Focus refers to the highlighted portion of the workbench
  - Can be an editor, a view, a project, etc.

- This concept is important since several operations inside Eclipse are tied to the element in focus
  - Project build errors, console, menu and toolbar options, etc.

'blink' project is in 'Focus' since it has been selected. So pressing the Debug button will build the project and start the debugger for the 'blink' project. Note how the project in focus is also highlighted in bold and with the word 'Active' next to it for easy identification.
View: Console

- **Multiple contexts**
  - Can display build messages or debug messages (including CIO) depending on which console is selected
  - Automatically switches contexts when a new message occurs
    - Can use the “Pin” option to prevent this

- **You can open multiple console windows**
  - CIO output in one and build messages in the other
Target Configuration Files - Basics

• Target Configuration files are xml files that define the connection and device (have a file extension *.ccxml)

• The Target Configurations view is used to manage and work with target configuration files

• Target Configuration Editor is used to create/modify target configuration files

• Basic tab is intended to be used by majority of end users

• Advanced tab is intended to be used for adjusting default properties, initialization scripts or creating target configurations for new boards/devices
View: Target Configurations

- Target configurations easily deleted, copied and opened for inspection (XML files)

- Launch a debug session quick: right-click on target configuration and select Launch Selected Configuration

- **Debug** will use target configuration that is identified with [Default] tag in Target Configurations View
  - Right click on a file and select Set as Default to make it the default
  - **Debug Active Project** will also use the Default if there is no target configuration file in the project
Eclipse Concept: Perspectives

• Defines the initial set and layout of views in the Workbench window

• Each perspective provides a set of functionality aimed at accomplishing a specific type of task (*CCS Edit* for project development, *CCS Debug* for debugging, etc)

• Can create custom perspectives
View: Debug

- Debug view displays:
  - Target configuration or project
  - Call stack

- Buttons to run, halt, terminate (debug session), source and asm stepping, reset CPU, restart program
View: Breakpoints

- View all available breakpoints
- Can group breakpoints by CPU (multicore device)
- Specify various actions when the breakpoint is triggered
  - Refresh All Windows or update a specific view
  - Control Profiling (set profile halt/resume points)
  - File I/O (Probe Points in CCS 3.3)
  - Run a GEL expression
  - Set a Watchpoint
  - Control CPU trace (on selected ARM & DSP devices)
  - Use the built-in HW counter events (Cortex devices)
View: Disassembly

- View disassembled code, optionally interleaved with C source
- Toggling the *Show Source* button toggles interleaved C source with the disassembly
- Type symbol names (such as main) in the address field to view code at that address
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
LAB 1: BLINKY EXAMPLE

30 MINUTES

Open CCS and select the default workspace
You can close the TI Resource Explorer View (it will not be used)
Refer to Lab handouts for instructions
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
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
SHARING PROJECTS
Sharing Projects

- Sharing “Simple” projects (all source/header files are contained in the project folder)
- Sharing “Linked file” projects and all source (project uses linked files)
- Sharing “Linked file” projects only (no source)
  - Only the project folder is shared (person receiving project already has all source)
  - Effort involves making the project “portable”
    - Eliminate absolute paths in the project
  - This is the most common use case
Sharing Projects – Simple Projects

- **USE CASE**: Wish to share (give) a project folder and all needed source files to build the project. All source files are inside the project folder.

- **Easy to share projects with no linked files**:
  - The entire project folder can be distributed to another “as-is”
  - The user who receives the project can import it into their workspace going to menu **Project → Import Existing CCE/CCS Project** and selecting the copied folder
  - Works well for simple projects that only reference files inside the project folder
Sharing Projects – “Linked file” Projects

• **USE CASE(S):**
  – Wish to share (give) a project folder only. The person receiving the project file already has a copy of the source files
  – Wish to check the project folder/files into source control

• Most use cases involve sharing **JUST** the projects instead of bundling all the source files
  – People will have their own local copies of the source files

• **Need to make the project portable to make sure the project is easily shared**

• **Portable projects avoid any absolute paths**

• Ideal portable projects should be usable without modifying any of the project files
  – This is ideal for projects maintained in source control
Sharing Projects – “Linked file” Projects

• Portable projects should avoid absolute paths

• Creating portable projects involves use of two kinds of variables:
  – Linked Resource Path variable
  – Build variable

• Both variables can be set to point to a directory

• Source files/resources can then be linked relative to the Linked Resource Path Variable

• Build Variables can be used when setting the project’s compiler and linker options (such as include paths to header files, paths to libraries etc)
Create a New Project

• A shared project is created the same way as a regular project

• Launch the New CCS Project Wizard
  – Go to menu File → New → CCS Project

• Fill in the fields as shown in the right

• Click Finish when done

• Generated project will appear in the Project Explorer view

• Remove the file lm4f120h5qr.cmd that was automatically added to project
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

• Open the workspace preferences
  – Menu Window → Preferences

• Go to the Linked Resources preferences
  – Type ‘Linked’ in the filter field to make it easier to find

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

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

• Go to the **Build Variables** preferences
  – Type ‘Variables’ in the filter field to make it easier to find

• Build Variables allow you to use variables in the project properties
  – Linked Resource variables are only used for linked files

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

• Click **OK**
Link Source Files to Project

• Source Files/Resources can be linked relative to a Linked Resource Path Variable

• As an example, here we will link source files relative to the Linked Resource Path Variable previously created

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

• Select source files and drag and drop them into the “hello” project in the CCS Project Explorer view
Link Source Files to Project

• A dialog will appear asking if you wish to Copy or Link the files:
  – Select *Link to files*
  – Select *Create link locations relative to*:
    • Use the new Linked Resource variable we created (*SW_ROOT*)
  – Hit *OK*

• Files will now appear in the Project Explorer with the ‘link’ icon
Link Files to Project

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

• Paths to include files and libraries can be set relative to Build Variables.

• As a example, here we will add include file search path using the Build Variable.

• Right-click on the project and select Properties.

• In the compiler Include Options, add the following entry to the list of include search paths:
  
  \${SW_ROOT}

• 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.

• NOTE: 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.
**Project vs Workspace Level Variables**

- *Linked Resource Path Variables* and *Build Variables* can be set at the project level or workspace level.

- This current lab set these variables at the workspace level.

- What is the benefit of setting these variables at the workspace level instead of the project level?
  - All projects can reuse the same variable (set it once).
  - Do not need to modify the project!

  - This is important for projects checked into source control and to avoid constant checkouts so the project can be written to!
Questions?
LAB 2: HELLO EXAMPLE
(PORTABLE PROJECT)

30 MINUTES

Refer to Lab Handouts for instructions
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