Code Composer Studio v6

AM335x Workshop

Texas Instruments

Texas Instruments

Welcome to the AM335x workshop for Code Composer Studio version 6

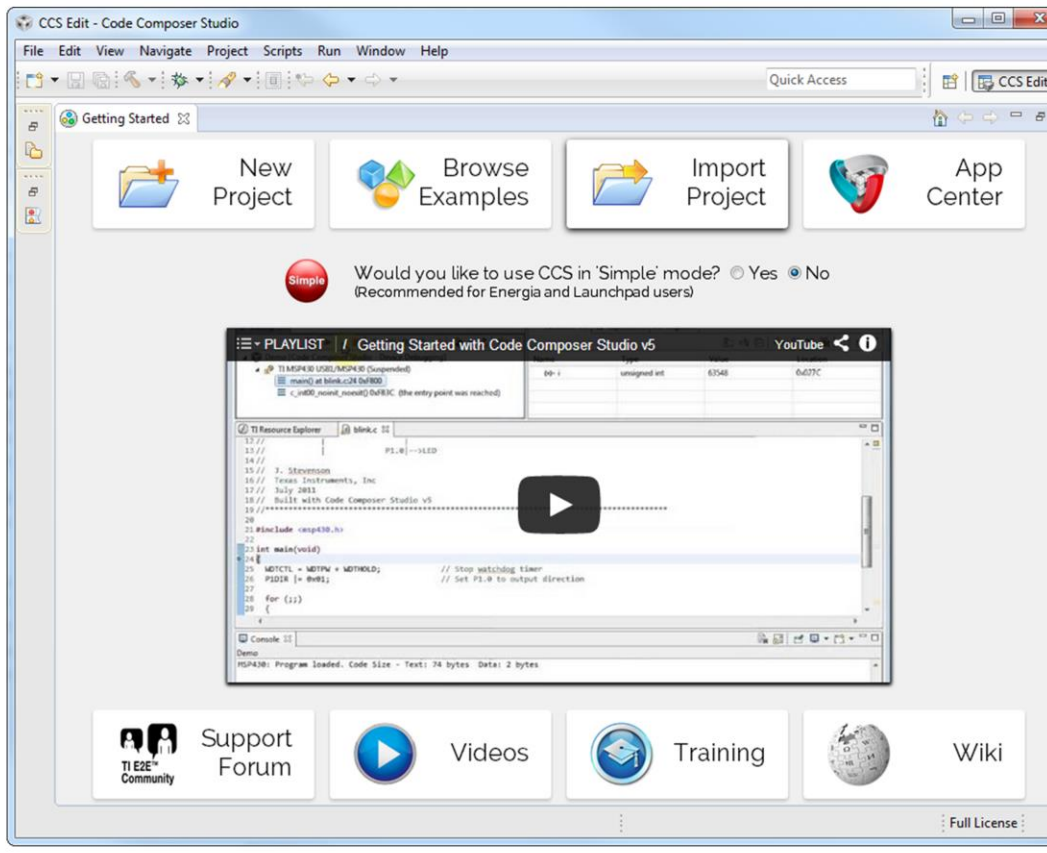
Code Composer Studio

- What is Code Composer Studio?
  - Integrated development environment for TI's embedded processors supporting all Microcontrollers and Processors in a single environment
  - A suite of tools including a debugger, compiler, editor…
- Based on the Eclipse open source software framework
  - Eclipse is widely used by many development environments
  - Offers a rich set of code development tools
  - TI contributes changes directly to the open source community
  - Extended by TI to support device capabilities
- Integrate additional tools & functionality
  - OS application development tools (TI-RTOS, Linux, Android…)
  - Code analysis, source control…

Code Composer Studio is the IDE of choice to develop with TI's microcontrollers and embedded processors. Now in its version 6, it also adds the integration of the low power RF device families (CC2500, CC2600, CC3200) and the eases the integration of embedded Linux development components such as LTTNG viewer, cross compiler integration, QT creator and others.

Getting Started View

- Initial screen visible when Code Composer Studio is launched
- Provides fast access to common tasks
  - Create a new project
  - Use a example
  - Open the App Center…
- Embedded Getting Started Video walks users through the basics of using CCS
- Links to support, videos and training material



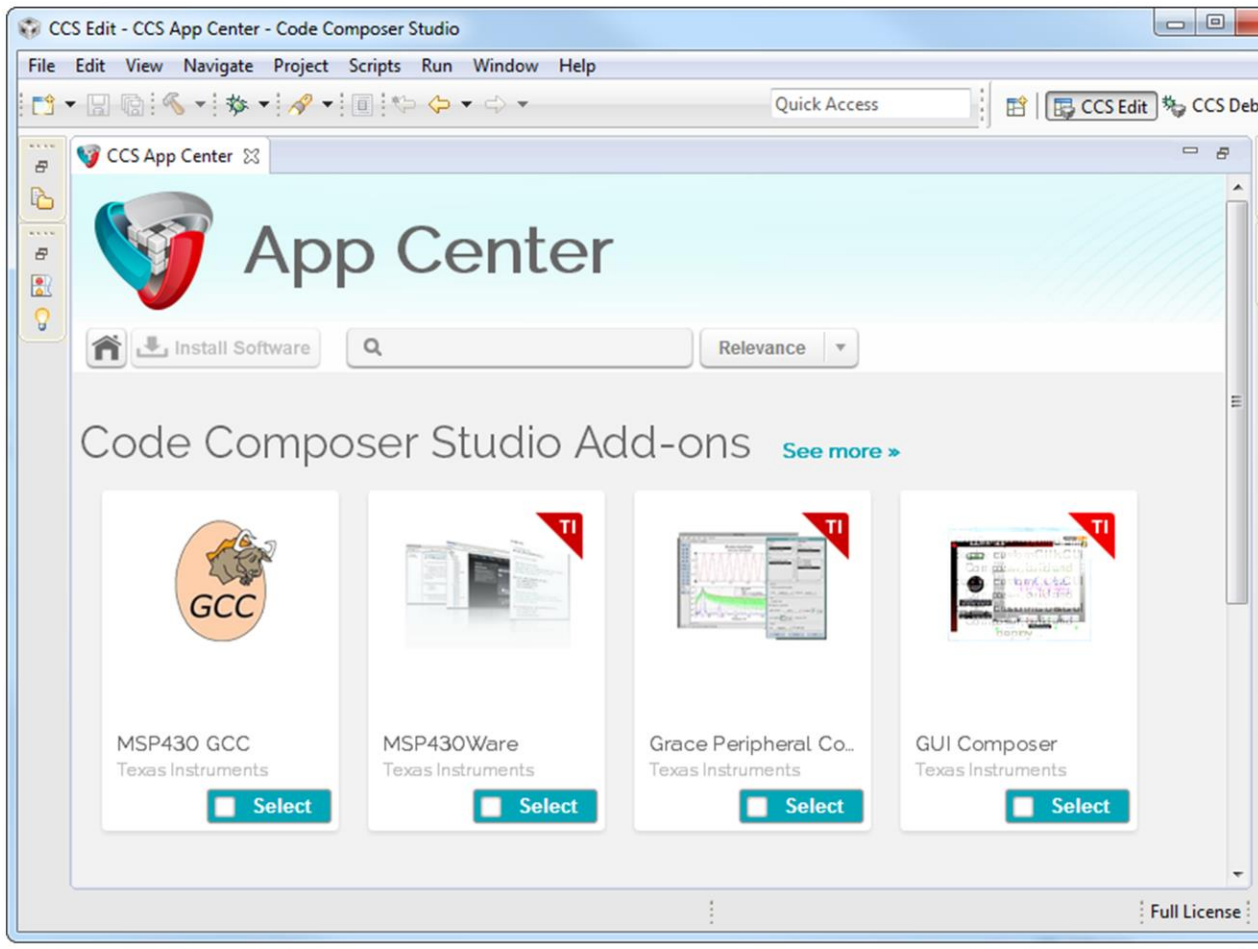
The new Getting Started view is more useful than ever, which means you shouldn't close it as the first thing you do after opening CCSv6.

It contains several buttons for easy access to key functions that allow you to get started with CCS: you can create or import a project, browse installed examples or jump to the App Center to download the remaining components needed for your development (such as MSP430Ware, controlSUITE, TI-RTOS, etc.). There is even a large video that shows how to get started with CCSv6.

Additional links at the bottom are very helpful, as they lead to the e2e forums, as well as the Code Composer Youtube channel, the CCS Training page and the main wiki page.

CCS App Center

- Access to additional products
  - "Wares"
  - SDKs
  - New features
- Keep products up to date from within CCS
- Further reduce CCS download size
- Great way to promote features

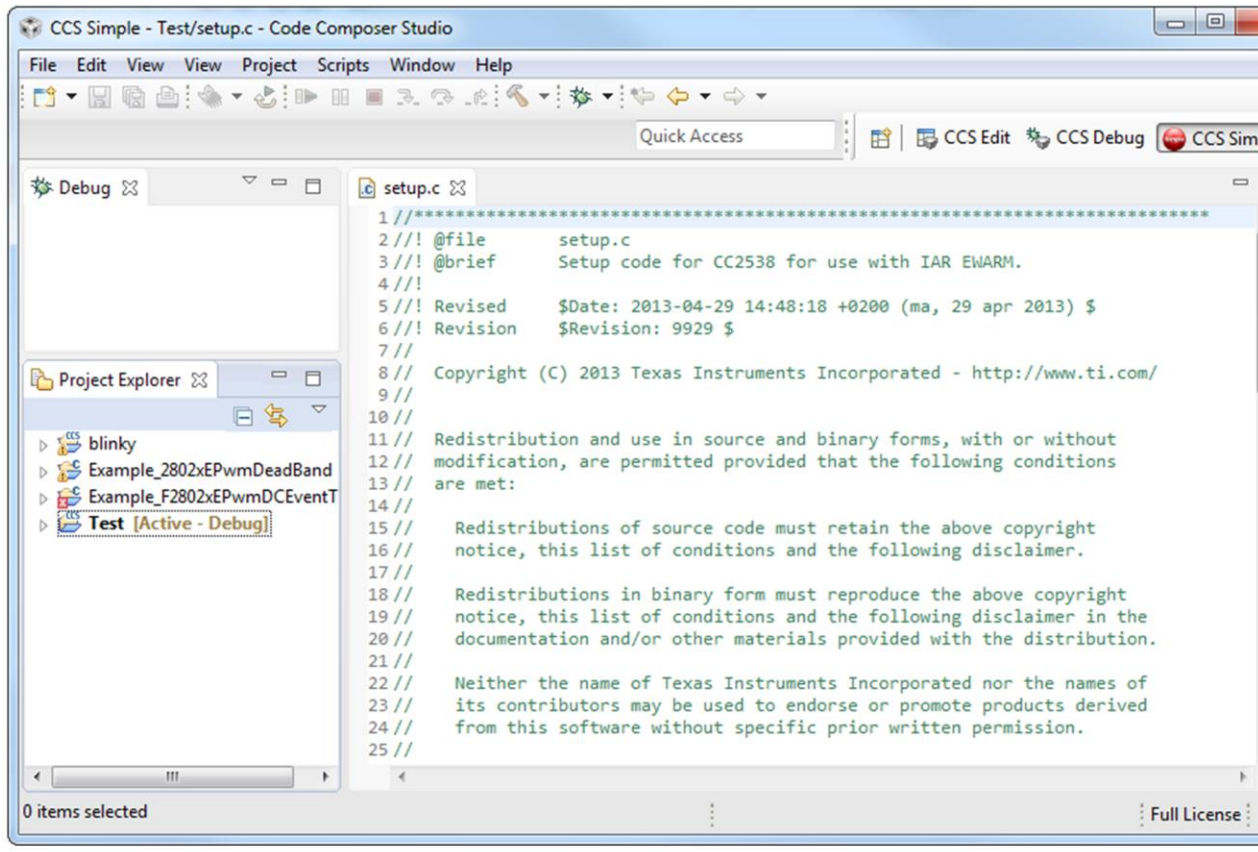


The App Center in CCSv6 is a new way to download and install software packages, new CCS features, update tools such as compilers, SYSBIOS, TI-RTOS, etc.

By means of a straightforward interface you can have access to several Add ons (which are installed into CCS), Standalone software and information links to various technologies. It can also be configured to alert you of updates to the installed components.

“Simple” Mode

- Enabling “Simple” mode reduces the environment to just essential functionality
- Makes it easier for new users, especially LaunchPad type customers to start using the CCS environment

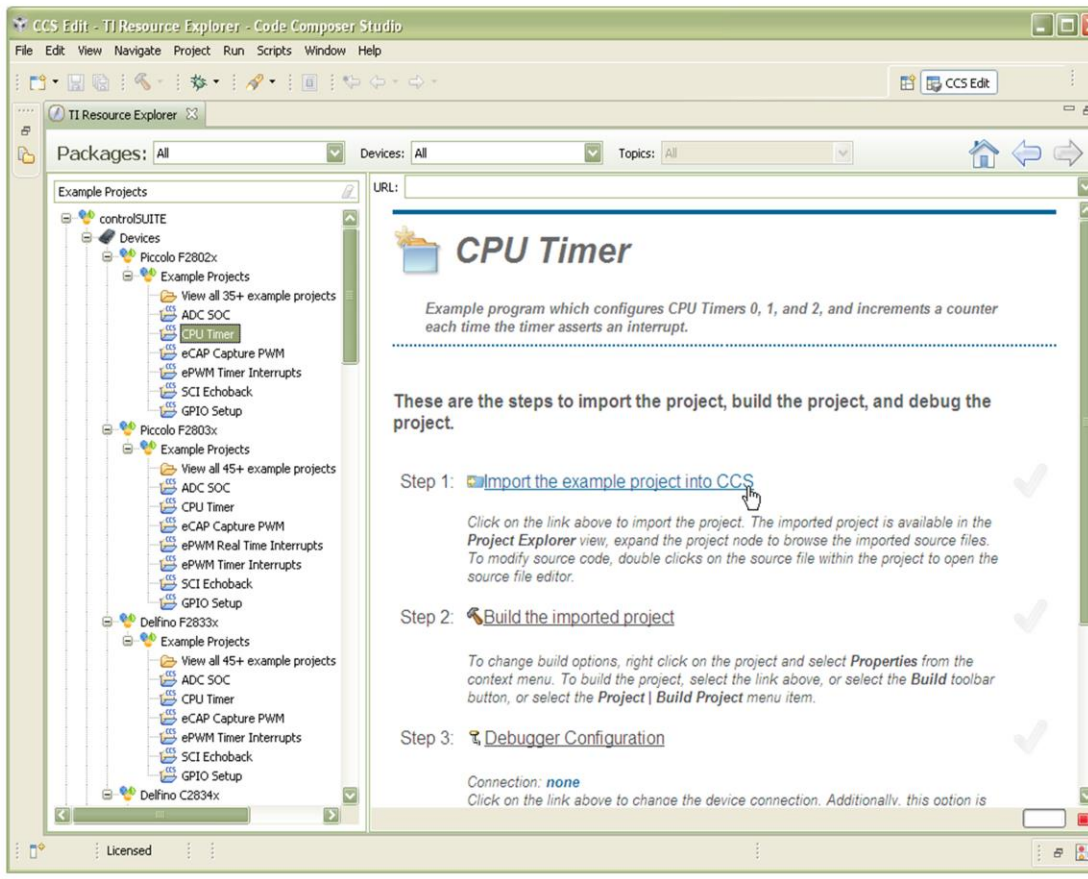


CCSv6 has an additional perspective called “Simple” (not to be confused with dumb ☺). This perspective arranges the views in a more familiar way used by traditional IDEs by aggregating the debugger and the code editor in a single screen, as well as reducing the options in the context and top menus for a more straightforward experience.

This is a good way to start getting acquainted with CCSv6 without jumping to a complete Eclipse environment.

Resource Explorer

- Easily access a broad selection of packages such as controlSUITE, MSP430ware, TivaWare, TI-RTOS…
- Guides you step by step through using examples
- Provides links to documentation, videos and other collateral



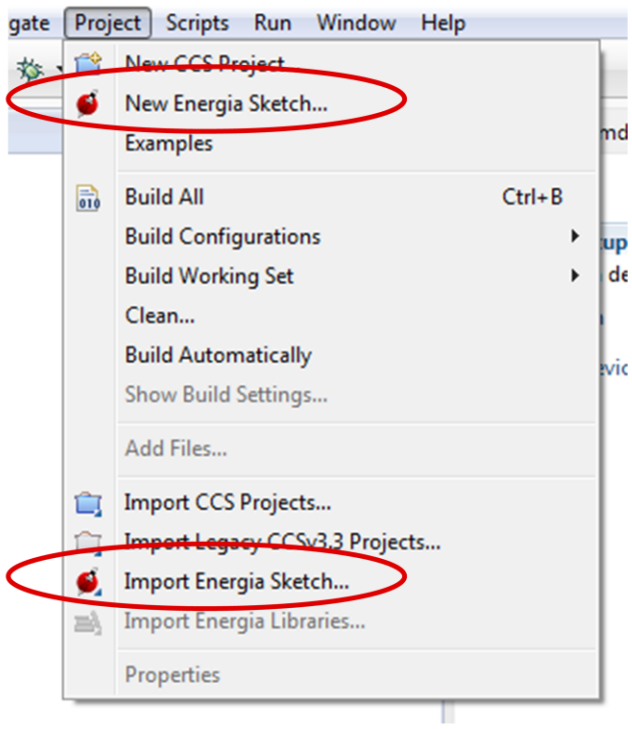
Texas Instruments

Usually the best way to start software development is with an example project. The Resource Explorer helps import example projects of a multitude of software packages: the so-called wares: MSP430Ware, Stellarisware, Tivaware, controlSUITE, etc. It also aggregates documents, schematics, and more (depends on the package).

It arranges the different software packages in a graphical tree for easy access to the resources, and allows importing, building and debugging in three mouse clicks (additional clicks to browse to the desired example are required, though).

Energia Support in CCSv6

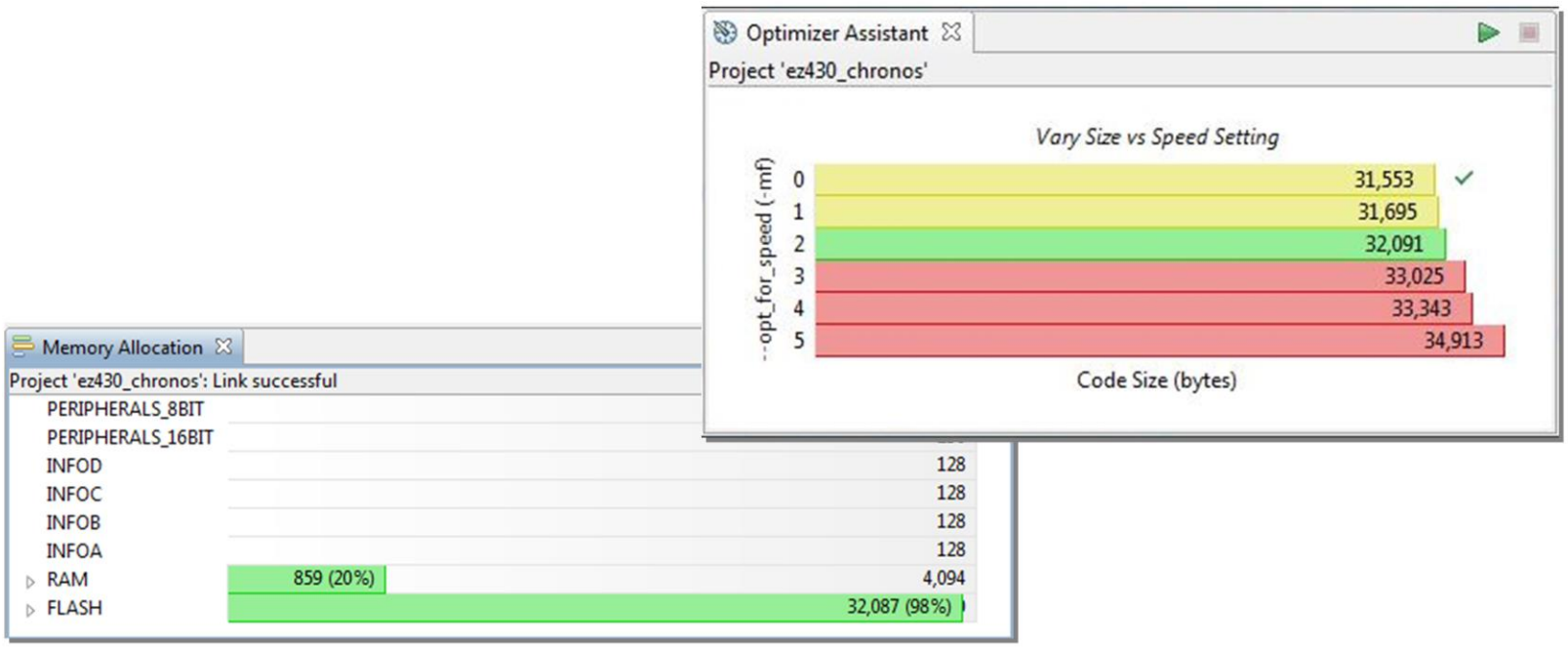
- Creating and Import existing Energia sketches
  - Leverage existing work
  - You can build/edit/debug Energia projects inside CCS
- Provide more debugging capabilities
  - JTAG debug
  - Register display
  - See the generated code
- Keep it simple
  - CCS will most likely be customer's second debugger ever used and their first full software development environment
  - Recommended to use "Simple" Mode



For the Arduino aficionados, Energia is the same environment ported to MSP430 and Tiva microcontrollers. However, any user of Arduino/Energia can quickly run into limitations about the ability to customize and debug the target, therefore CCSv6 allows importing and/or creating sketches (as projects are called in Arduino/Energia) for a more professional development.

Optimizer Assistant

- Provides advice on how to get the best performance for available code size.
- Graphical display of memory usage.



When you are building your system, you constantly look for that sweet spot where everything just barely fits in memory, it runs very fast, and the battery lasts almost forever. Everyone knows the easy way to do that is build with --the_perfect_compiler_option. Oh wait, there is no such option!

In real life, the way to find the sweet spot is to experiment with the compiler options. Try different options, try different settings for those options, and so on. A common response is that is too hard, or error prone, or not much fun, or something. There has to be something better.

Well, there is something better, and it's called Optimizer Assistant.

ULP (Ultra Low Power) Advisor

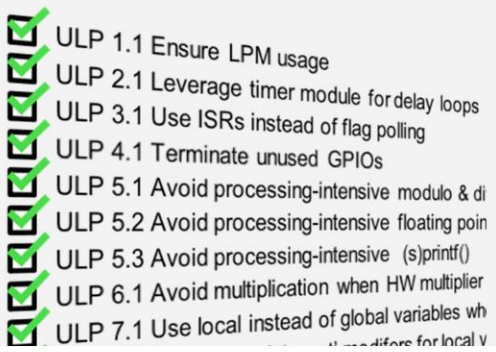
Turning MCU developers into Ultra-Low-Power experts

ULP Advisor analyzes all C code line by line.

Can benefit any application
Checks all code within a project at build time
Enabled by default
Parses code line-by-line

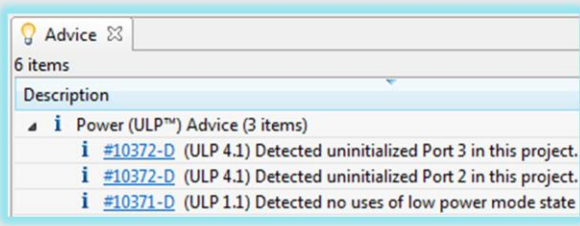
Checks against a thorough Ultra-Low-Power checklist.

- List of 15 Ultra-Low-Power best practices
- Compilation of ULP tips & tricks from the well-known to the more obscure
- Combines decades of Ultra-Low-power development experience



Highlights areas of improvement within code.

- Identify key areas for improvement
- Listed in the “Advice” view
- Includes a link to more information



www.ti.com/ulpadvisor

Texas Instruments

The mysteries and intricacies of developing code for low-power designs is a vast and heavily device-dependent field. To help with that, the ULP (Ultra-Low Power) Advisor helps you write your code in a way that fully utilizes the ultra-low power features of MSP430 microcontrollers.

At build time, ULP Advisor checks your code against a thorough checklist and provides notifications and remarks to highlight areas of it that can be further optimized for lower power. Within these notifications, developers will get a description of the ULP rule in violation, and will also get a link to this ULP Advisor wiki for further information, links to relevant documentation, code examples and forum posts!

Code Composer Studio Licenses

- Free options:
  - 90 day evaluation
  - MSP430: 16KB Code Size Limited with the optimizing TI compiler
  - MSP430: unlimited code size with GCC
  - When using XDS100 JTAG emulators
  - Tied to development kits with onboard emulation (not for MSP430)
- Purchase professional tools:
  - Starting at $495
  - Node Locked License (tied to a PC)
  - Floating Licenses available (shared licenses)
  - www.ti.com/ccstudio

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

Similarly to all previous versions, to use CCSv6 you need a license (no, it doesn't mean you have to invest money to use CCSv6 – wait and see)

Following the trend started with CCSv5, the licensing became even simpler in CCSv6:

- In addition to XDS100, simulators and kits with onboard emulation, the free license now adds both the MSP430 code size limited and the MSP430 GCC unlimited on the same install. Previously a secondary install was required. This is the license you want to trial run CCS and it does not require an active internet connection to use it – simply start running.
- The full license allows using advanced emulators, removes the limit on the MSP430 compiler and can be either tied to the PC or can be installed in the network for multiple PC accesses. This is the license you need if you decide that additional performance is necessary (depending on the device family) or if your MSP430 code increased past the limit of the free license.

MSP430 JTAG Emulators

MSP-FET430UIF

TEXAS INSTRUMENTS
MSP430

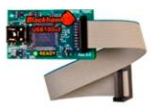
- USB Interface
- Compatible with CCS, IAR and other debuggers
- $115

TEXAS INSTRUMENTS

For MSP430 the MSP Flash Emulation tool or FET is the standard solution.

XDS Debug Probes

XDS100v2 – Entry Level




- Entry level JTAG emulator
- USB interface
- 3 models based on JTAG headers (14pin TI, 20pin TI, 20/10pin ARM)
- $79

XDS200 – Mid Range

- Excellent balance of performance and cost
- USB interface (Ethernet version available)
- 20pin TI, 14pin TI, 20pin ARM and 10pin ARM connectors
- $295

XDS560v2 – High Performance


- USB or USB + Ethernet interfaces
- Includes multiple JTAG adapters (14pin TI, 20pin TI, 20pin ARM, 60pin MIPI, some include 60pin TI)
- System Trace
- $995 - $1495

Pro Trace

- Trace Receiver & XDS560v2 JTAG emulator
- USB + Ethernet interfaces
- MIPI60 and 60pin TI adapters
- DSP, ARM & System Trace to pins
- $3495

Texas Instruments

TI offers a wide variety of JTAG emulators to suit different needs and budgets. We carry entry level xds100 class products all the way up to high end trace receivers like the XDS Pro Trace and everything in-between.

XDS100 is a low cost entry level solution. It is a good product to start development with but typically you will want to upgrade to a higher end product later on. The XDS100 is available in 3 different models, each supporting a different set of JTAG connectors. XDS100 products are available for the low cost of 79 dollars and have the added bonus of working with a free license for Code Composer Studio.

XDS200 is a new class of JTAG emulator. It is a mid range product that fits between the entry level XDS100 and the high performance XDS560. It supports a USB interface, although Ethernet versions are available from some of our partners. The XDS200 comes with a selection of JTAG connectors. This product has a cost of 295 dollars and is an excellent choice for microcontroller devices.

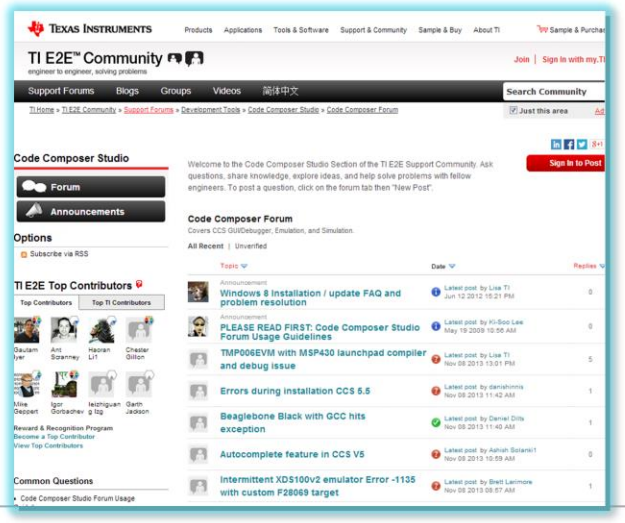
XDS560v2 emulators are our high performance JTAG solution. We offer 2 models, a bus powered USB model and one that offers both USB and Ethernet connections. Note that the USB & Ethernet model includes an extra adapter to support the 60pin TI JTAG header. Both of these models incorporate a System Trace receiver with 128 MB of storage. System trace is an advanced feature available on high end processors that enables you to detect system bottlenecks by tracking bus and peripheral activity, benchmarking data throughput and even the ability to add custom instrumentation. These products start at 995 dollars and are recommended for those working with higher end processors that have large amounts of memory or for those who have a need for System Trace.

The XDS PRO TRACE is our top of the line trace receiver. While all of our XDS products will support reading trace data that is stored in the on-chip trace buffer (ETB), the XDS PRO TRACE is the only one that can capture processor trace that is exported from the device to pins, enabling you to capture huge amounts of trace data. This product is recommended for customers who need to capture processor trace, this is most common on our multicore C6000 devices. It also includes a System Trace receiver. The XDS PRO TRACE is available for 3495 dollars.

Extensive Support

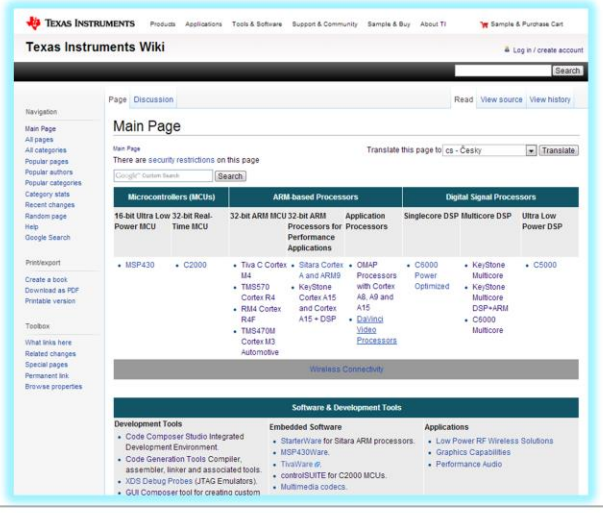
E2E Community

- Videos, Blogs, Forums
- Global customer support
- Search for answers
- http://e2e.ti.com



Wiki

- Technical articles
- Training
- Design ideas
- http://processors.wiki.ti.com

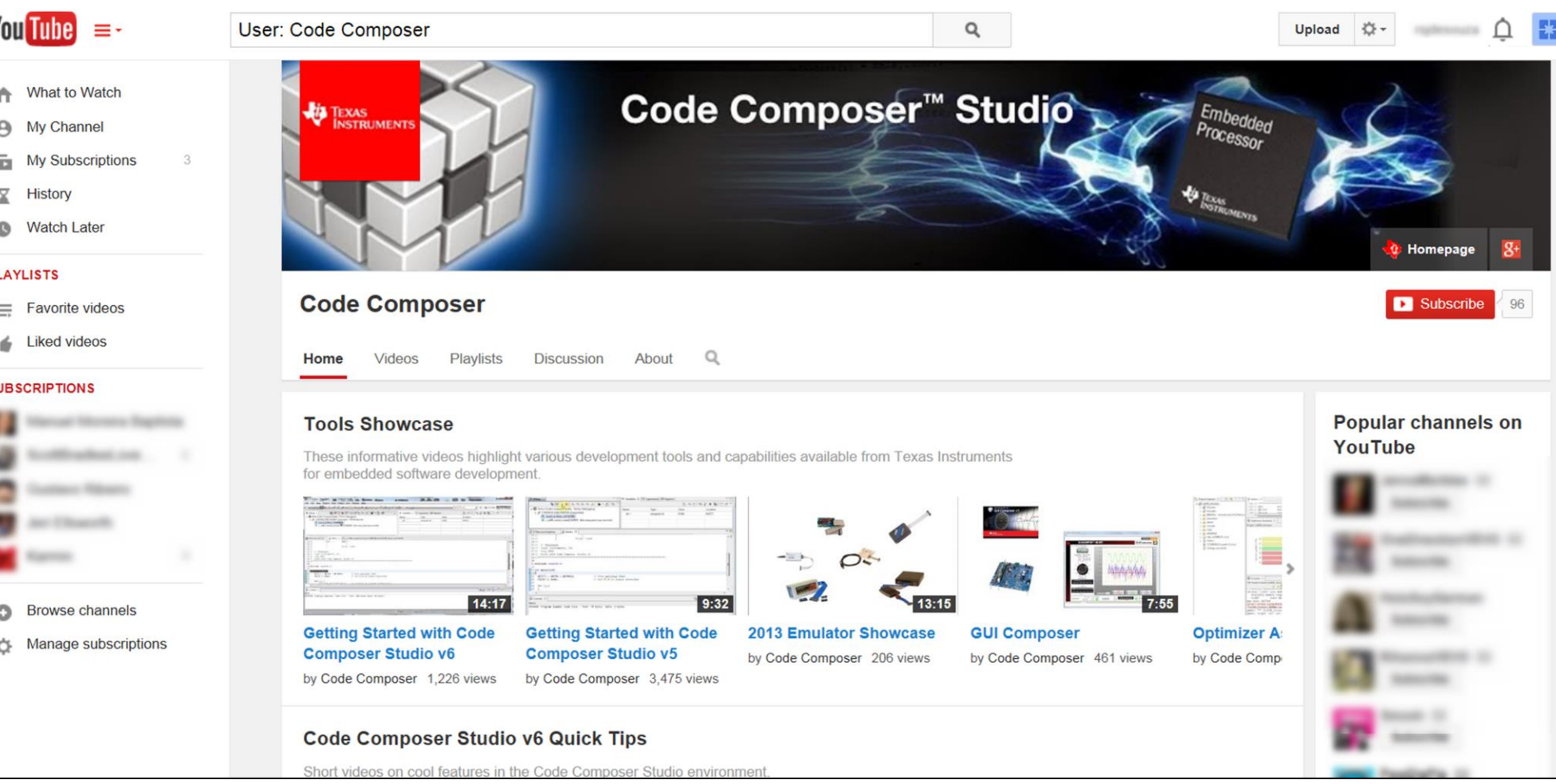


Here are the two main avenues for support everything related to TI, including CCSv6.

The e2e community is a forum that, over the years, has become a very resourceful place to search for answers and tips on how to use TI products. Its main purpose is to enable direct interaction between developers or contact TI development and applications teams.

The wiki is one of the main documentation places to find the most up-to-date information about TI's software and development tools, including CCSv6 and its various components.

YouTube Channel



New for 2014 is the Code Composer Studio youtube channel. This channel contains several quick tips and Getting Started Guides, as well as webinars that showcase new features of the tools.

To find it, simply go to youtube and search for “User: Code Composer” or simply click on the video on the Getting Started view or find links throughout the processors wiki and e2e

Getting Started with CCSv6 and AM3358 BeagleBone

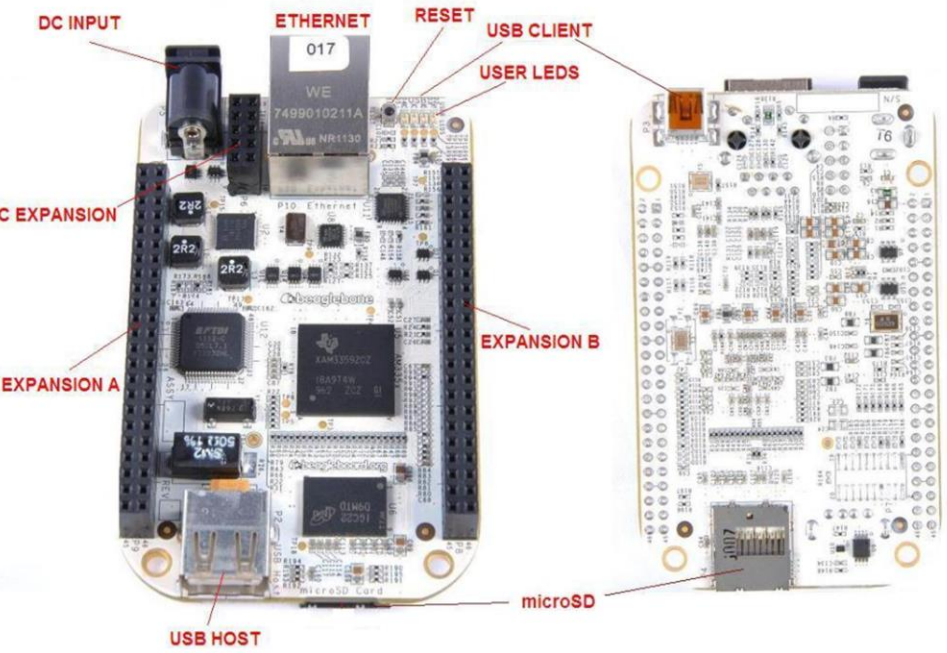
Let’s start working with CCSv6

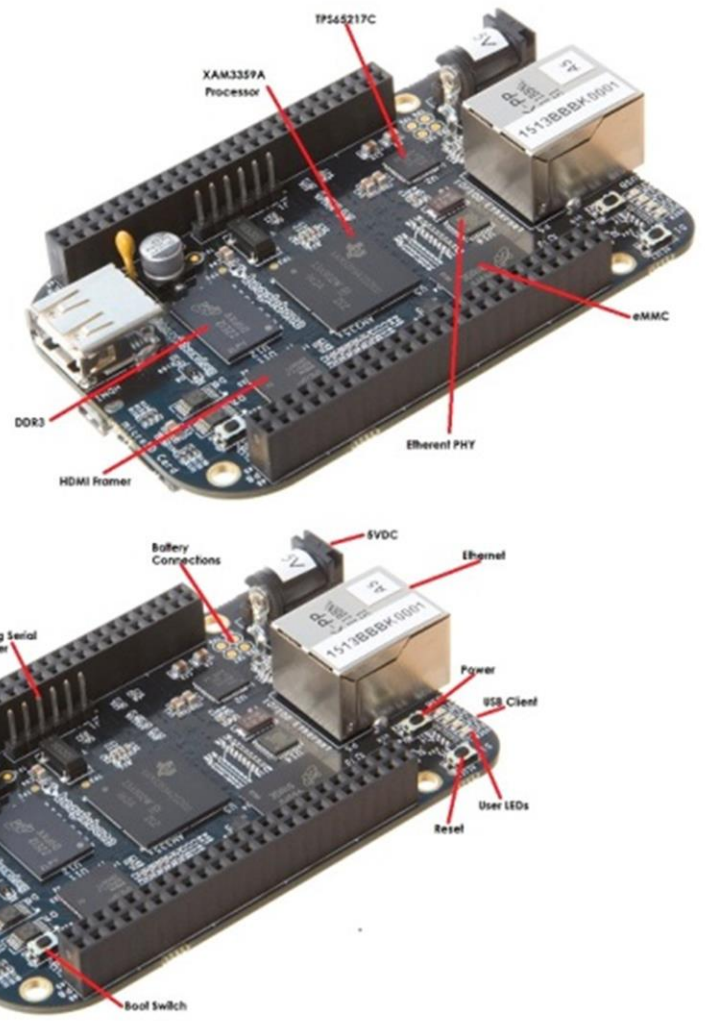
What is AM3358 Beaglebone?

- AM3358 Cortex A8 microprocessor (720MHz on White, 1GHz on Black)
- Quick and easy evaluation of all of the advanced capabilities of the community board for a low price: $89 for White, $45 for Black
- Multiple graphical development platforms for easy development (Cloud9, GateOne, CCS)
- Fits in an Altoids box!
- On-board emulation, access to several I/O pins (ADC, PWM, UART, I^2C, SPI, etc.)
- Detailed example software and documentation
- Complete hardware schematics and board files
- The White has a built-in XDS100 emulator for quick baremetal development and low-level debugging

The BeagleBone is the most popular and low-cost development board that sports a Cortex A8 processor, and the convenient built-in JTAG emulator on the White version allows a seamless operation with CCSv6. This makes it a perfect candidate to be used in this workshop.

AM3358 BeagleBone



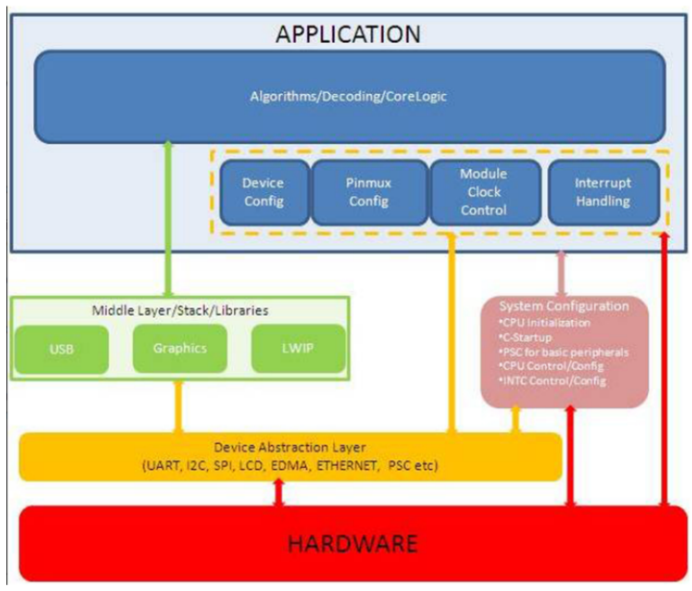


Given its low cost, the BeagleBone boards pack a lot of features and room for expansion throughout its connectors. The Black version is a more recent version that has a higher processor speed (1GHz versus 720MHz for the White) double the RAM (512MB versus 256MB) and a built-in 2GB eMMC card (the White has no non-volatile memory). As pluses for the White are the serial and JTAG ports over USB (the Black requires a special cable to connect to the serial port, an external JTAG emulator and a JTAG connector needs to be soldered). Despite the differences, both boards can be used for this workshop provided the additional hardware for the Black is available.

What is Starterware?

- A quick and easy way to start development without a High level Operating System (Linux, WinCE, etc.)
- Contains example code for several peripherals (GPIO, UART, Ethernet, RTC, etc.)
- Supports multiple development platforms (CCS, IAR, GCC)
- Supports several processors:
  - C6748, OMAPL138, AM1808, AM335x
- Can be downloaded free at:

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



Starterware is a collection of example code and libraries suitable for baremetal development (i.e., without an Operating System such as Linux or TI-RTOS) and therefore is a perfect way to start development with CCSv6. We will use the AM335x version for this workshop.

GPIO LED BLINK EXAMPLE: BASIC PROJECT DEBUGGING

Now let’s move on to the first project

GPIO LED Blink Example: Exercise Summary

- Key Objectives
  - Create and build a simple program to blink USR2 LED (D4)
  - Start a debug session and load/flash the program on the BeagleBone
  - Run the program to blink USR2 LED
- Tools and Concepts Covered
  - Workspaces
  - Welcome screen / Resource Explorer
  - Project concepts
  - Basics of working with views
  - Debug launch
  - Debug control
  - Profile Clock
  - Local History
  - Build Properties
  - Changing compiler versions

The first project is a simple blink LED type of project (the “Hello World” of hardware development kits), which will allow exploring several important concepts of general Eclipse and CCSv6.

The next slides will simply show the fundamental concepts of Eclipse and how they are related to the CCSv6 development environment.

GPIO LED Blink Example: Requirements

- Software
  - Code Composer Studio release 6.0.0.00179
  - Starterware-AM335x release 02.00.01.01

  Note: the lab procedure assumes that both Code Composer Studio and Starterware are installed at their default locations under C:\ti

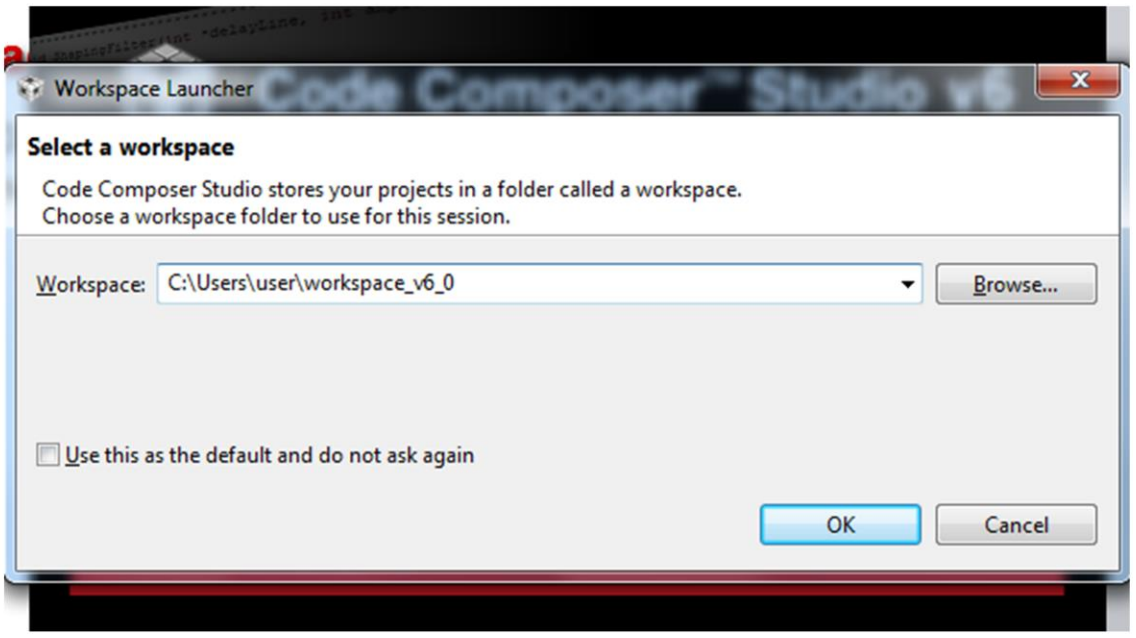
- Hardware
  - BeagleBone White Rev A5 or higher
  - USB A to mini-B cable

  Note: BeagleBone Black can also be used, provided it is fitted with the JTAG connector and an external JTAG emulator is available. For details, check section 5.1 of:
  http://elinux.org/Beagleboard:BeagleBoneBlack

Here are the requirements for the labs. Just a board, CCSv6 and Starterware.

Workspace

- Launch CCS and select a workspace folder
  - Defaults to your user folder



First thing to choose is a workspace. The workspace is the main working folder for CCSv6 and where it stores all the projects, files, links and their configurations. This is the case even if the projects themselves do not physically reside inside the workspace folder.

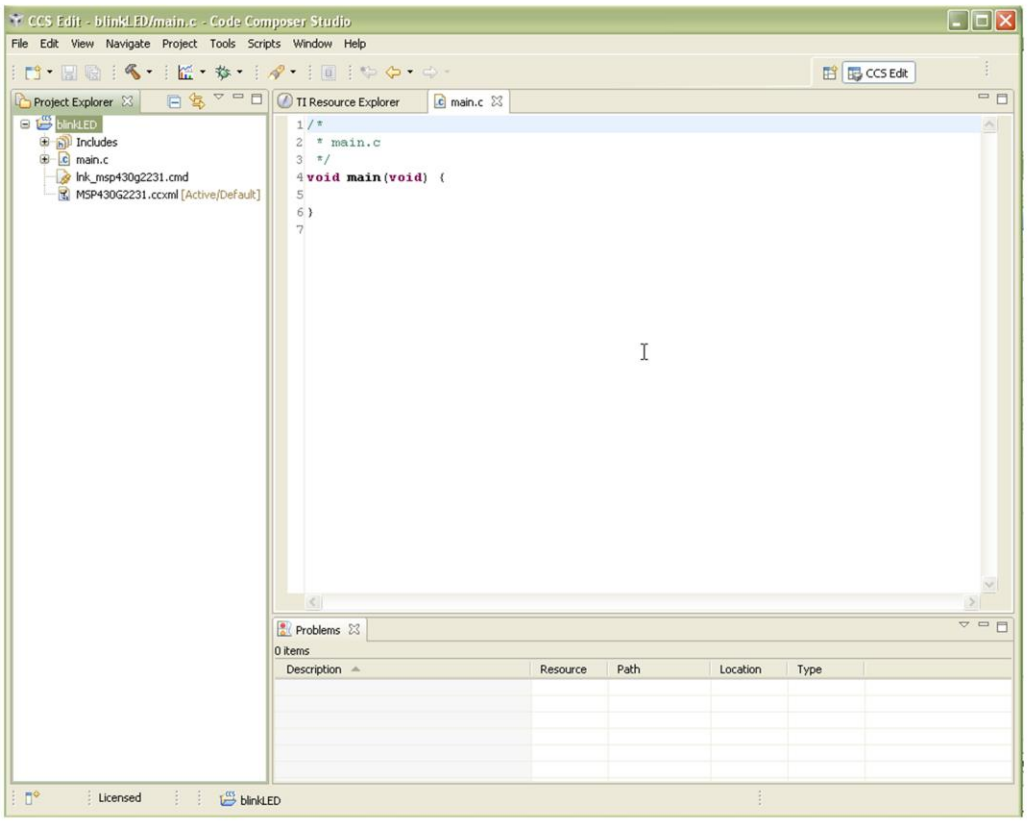
Eclipse Concept: Workspaces

- 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
- Workspaces are not to be confused with CCSv3 workspace files (*.wks)
- 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

The default location of any new projects created in CCS will be within the workspace folder. Once a project has been defined to the workspace, it will be visible in the ‘Project Explorer' view and can be edited and built (compile/assembled/linked). To be able to work on an existing CCSv6 project, it needs to be imported into the workspace.

Eclipse Concept: Workbench

- *Workbench* refers to the main CCSv6 GUI window
  - Equivalent to the *Control Window* in CCS 3.x
- The Workbench contains all the various views and resources used for development and debug



The Workbench refers to the CCSv6 application itself and it contains all the various views and resources used for development and debug. Multiple CCSv6 Workbench windows can be opened (menu 'Window->New Window').

While each Workbench window can differ visually (arrangement of views, toolbars and such), all windows refer to the same workspace and the same running instance of CCS – for example, if a project is opened from one Workbench, that same project will be open in all the Workbench windows.

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
    - CCS 3.x project files used this concept exclusively
- 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
    - This differs from CCS 3.x, where closed projects do not appear at all in the CCS 3.x project view window.
- Projects that are not part of the workspace must be imported into the active workspace before they can be opened
  - Both CCSv4/5/6, CCE projects and legacy CCSv3 projects can be imported into the workspace

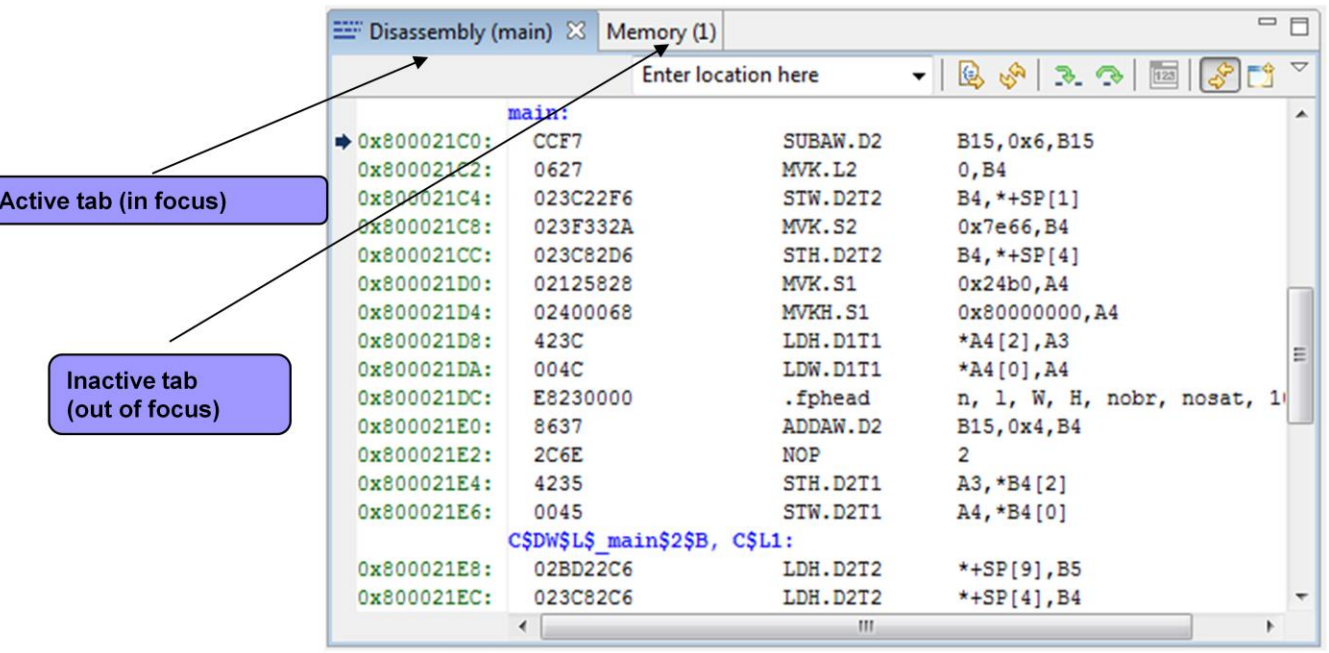
Like the workspace, projects map to directories in the file system and typically contain directories and files. When creating a new project, a directory with the project name is created within the workspace folder. A reference to it will be made in the workspace and the project is now available for use within the Workbench and visible from the 'C/C++ Projects' view. Projects outside of the workspace directory can also be used – that is when the project is “linked” to the workspace.

Projects are either open or closed. When a project is closed, it is 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. Hence closing unnecessary projects can improve performance of CCS. Note that closed projects will still be visible in the 'C/C++ Project' view (with a 'closed' folder icon).

Any projects that have not been defined to the workspace must be imported into CCS.

Eclipse Concept: Views

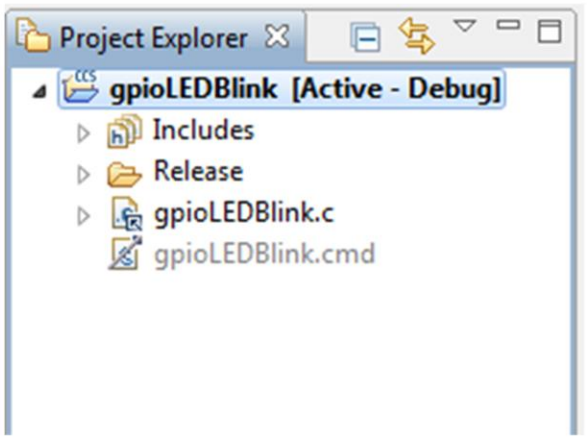
- Views are windows within the main Workbench window that provide visual representation of some specific information.
  - Most views can be accessed in the menu View
  - Can be identified by the organization in tabs



Views are windows within the main Workbench window that provide visual representation of some specific information. The Workbench window mainly consists of the editor and a collection of views. Examples of some views are 'C/C++ Projects', 'Debug', 'Outline', 'Memory', 'Disassembly', etc.

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 (.*)

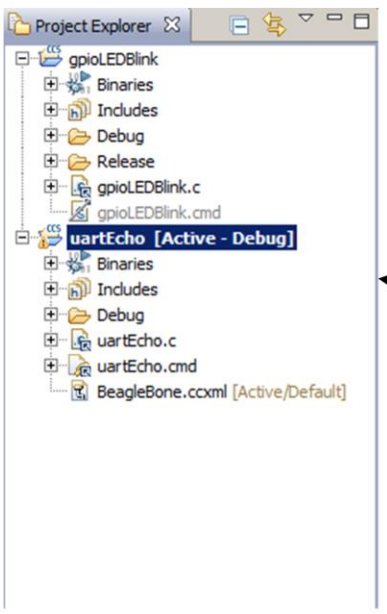


The Project Explorer is the most useful view for development, as it displays all projects in the workspace.

It shows everything in a familiar tree view and acts like a file browser, allowing copy, paste operations to and from the file system; delete and rename, filtering to display only the desired files and so on. It also has right-click context menus, which are very useful to explore the Project Options, Build Configurations and other features.

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.



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

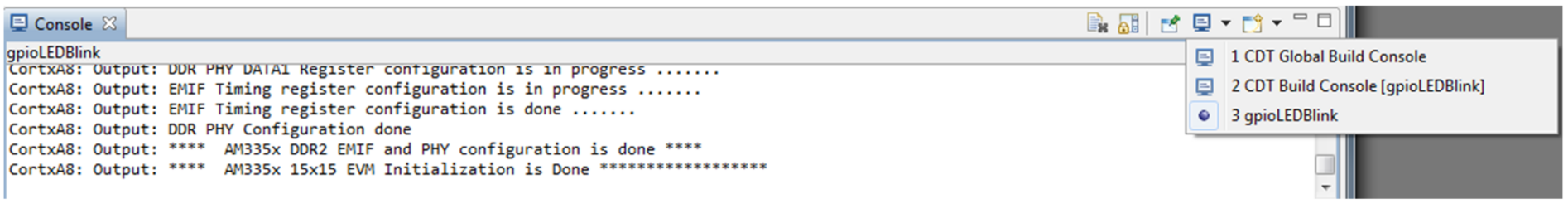
Texas Instruments

Focus is an action – it is basically the Eclipse name for highlight. Despite it is a simple concept, Eclipse relies heavily on this to tailor menus, right-click options, and influences other actions such as debug a project, save a file, etc.

Therefore, whenever you try to get a menu or is doing something that does not seem to be working properly, always make sure the correct view, project, or file is in focus.

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
  - CCS 3.x had separate, dedicated views for build output, CIO output and debugger messages
- You can open multiple console windows
  - CIO output in one and build messages in the other



The Console is a view that aggregates a lot of information in stacked panes selectable by the small blue icon shown above. It displays the build command line, console output from board printfs, scripting output, etc.

Its contents are also shown based on the project or element in focus, as well as which perspective is active.

Target Configuration Files - Basics

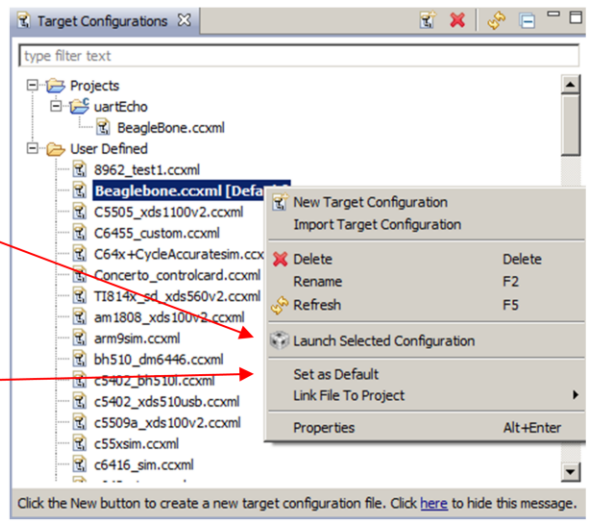
- Target Configuration files are xml files that define the connection and device (have a file extension *.ccxml)
  - Equivalent of the CCS 3.x configuration file (*.ccs extension)
- The *Target Configurations* view is used to manage and work with target configuration files
- *Target Configuration Editor* 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

Any time you need to connect to a target – either a software simulator or a hardware JTAG emulator tied to a development kit or your product board – a target configuration file must be created.

The target configuration files are XML files that define the type of connection (simulator, type of JTAG emulator) and a device (AM335x, C66x, MSP430, etc.). They also carry some basic information that configures the JTAG emulation hardware and can point to scripts that perform hardware initialization.

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

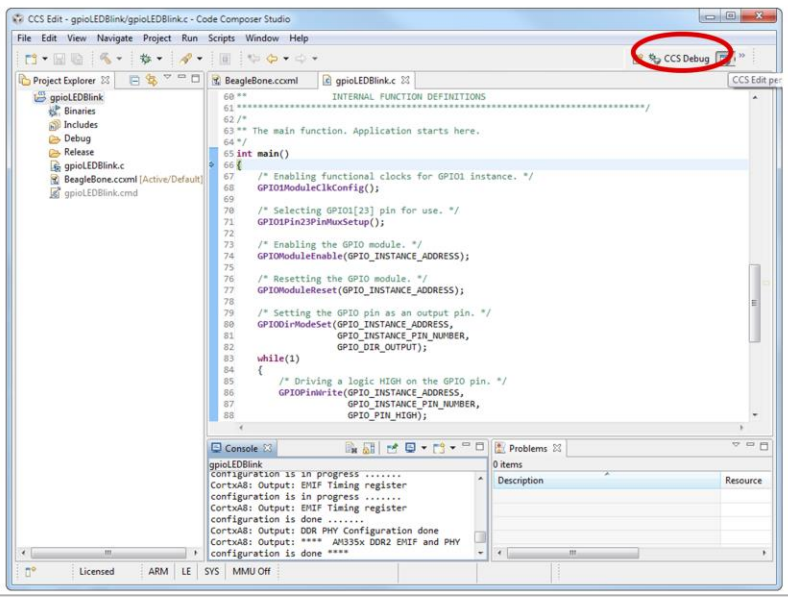


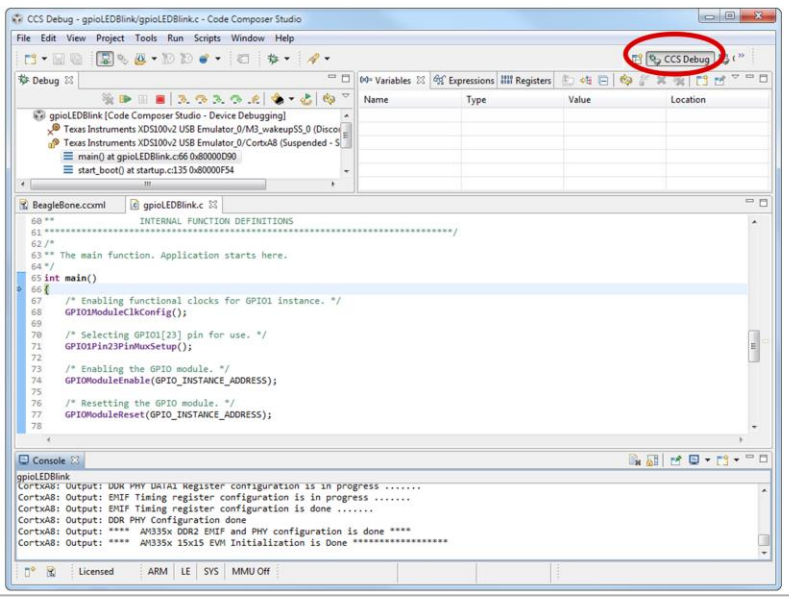
The target configurations view is extremely useful if you use multiple boards or emulators. It is shown in a tree view, similarly to the Project Explorer view, and allows copying, renaming, editing and manually launching targets.

You can also link target configurations to projects, so they can be easily launched by the Debug button.

Eclipse Concept: Perspectives

- Defines the initial set and layout of views in the Workbench window
- Similar in concept to CCSv3 *workspaces* (*.wks) except that multiple perspectives are available from the Workbench window (though only one can remain active at a time)
- 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

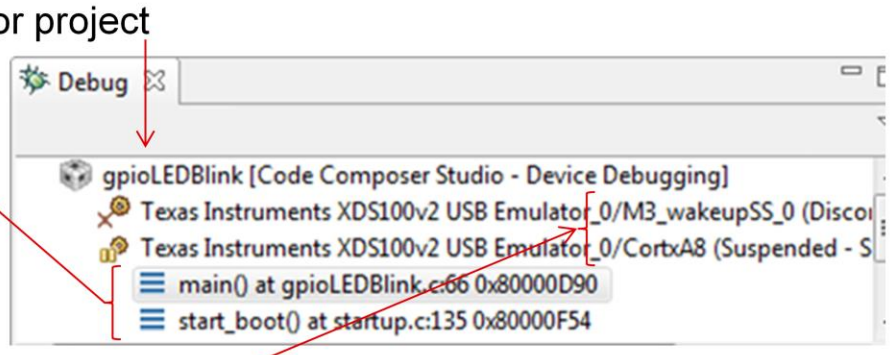




A perspective is the basic arrangement of views and menu options that are available to the user. Each perspective provides a set of functionality aimed at accomplishing a specific type of task. For example, the default 'CCS Edit' perspective displays views most commonly used during code development, such as the Project Explorer' view, 'Problems' view and the Editor. When a debug session is started, CCS will automatically switch to the 'CCS Debug' perspective, which (by default) displays the 'Debug' view, 'Watch' view and 'Local' view. Also in the 'Debug' perspective, menus and toolbars associated with debugging (such as target connect, load program, reset target, etc) are now available.

View: Debug

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



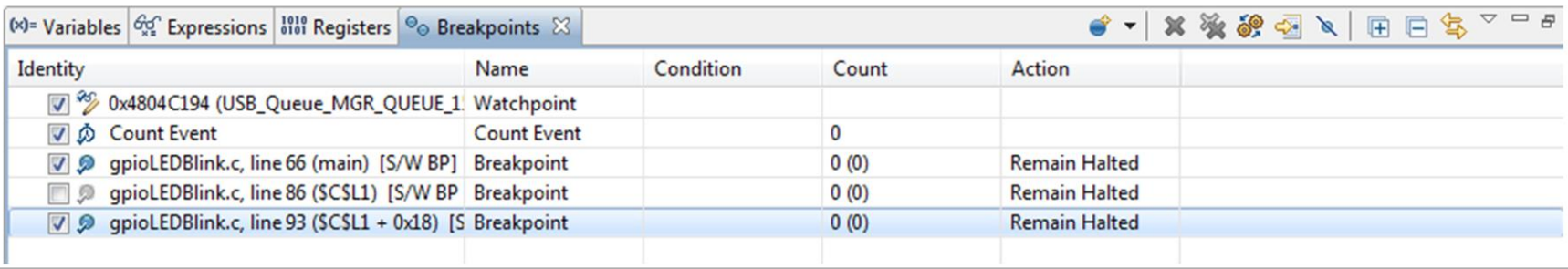
- The cores that comprise the AM335x device (in the example above, one Cortex M3 and one Cortex A8)

For complex processors such as the AM335x family, the debug is a very important view of the Debug Perspective as it shows which core is active, connected, has a program loaded, is running or halted, etc.

One change between CCSv5 and CCSv6 is that this view does not have the run, halt and terminate buttons anymore.

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 (replaces “Animate” in CCS 3.3)
  - 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)

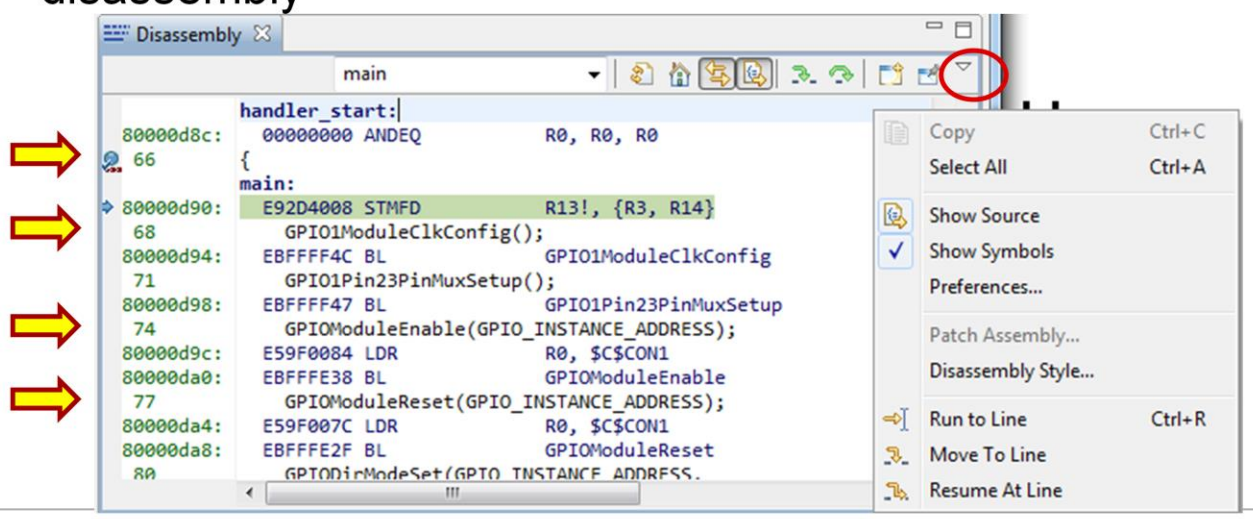


The Breakpoints view consolidates all the breakpoints used in your debug session and displays useful information about them. It also allows disabling, removing and configuring each one in a consolidated view.

This becomes a very useful view when debugging complex systems, especially in advanced processors such as the Cortex A8 (microcontrollers only have a handful of usable hardware breakpoints).

View: Disassembly

- Go to the main() symbol in the Disassembly view by typing it in the address field and hit <ENTER>
- You can see the current location of the PC (small blue arrow) and any breakpoints (small blue circles)
- Toggle the *Show Source* button. Note the toggling of interleaved source with the disassembly

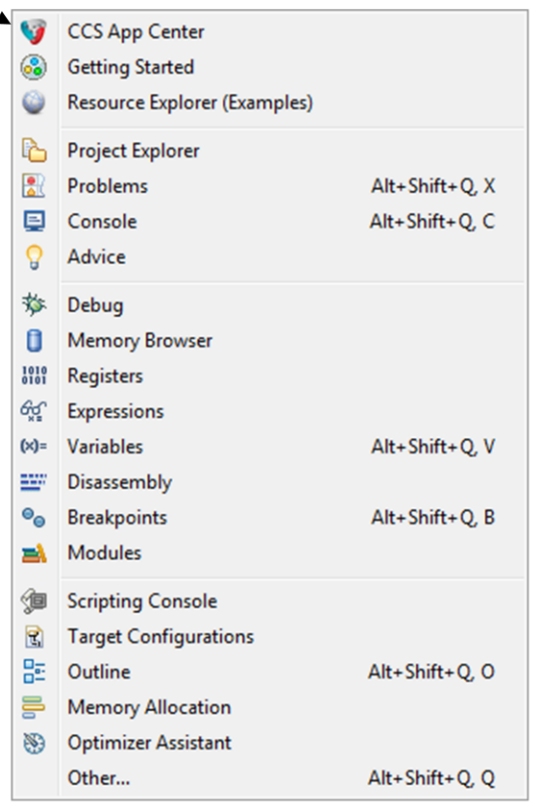


The Disassembly view is extremely important if assembly debugging is needed. Very useful in baremetal development, as sometimes you need to go beyond the C source code to really know what is going on with the code flow.

Despite being mostly assembly oriented, it still performs correlation with the C source code (enabled by default) for easier debugging.

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



The View menu consolidates all the views useable from inside CCSv6 – some applicable to certain perspectives, but others applicable everywhere. The most useful views are listed, and the bottom entry called “other” allows searching for any of the possible views of CCSv6.

LAB 1: GPIOLEDBLINK EXAMPLE

30 MINUTES

Open CCS and select the default workspace
You can close the TI Resource Explorer View (it will not be used)

Now let's start working on the lab. 30 minutes are typically needed to finish everything.

UART ECHO EXAMPLE: PORTABLE PROJECT

Texas Instruments

Now let’s move on to the second project

UART Echo Example: Exercise Summary

- Key Objectives
  - Create a new portable project based on the UART Echo 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

The second project is a bit more complex as it involves Hardware configuration to perform communications via UART. The hardware configuration is conveniently done by the Starterware example code, but the important concept is to create a completely portable project in CCSv6.

UART Echo Example: Requirements

- Software
  - Code Composer Studio release 6.0.0.00188
  - Starterware-AM335x release 02.00.01.01

  Note: the lab procedure assumes that both Code Composer Studio and Starterware are installed at their default locations under C:\ti

- Hardware
  - BeagleBone Rev A5
  - USB A to mini-B cable

  Note: BeagleBone Black can also be used, provided it is fitted with the JTAG connector and an external JTAG emulator is available. Also, you need to have a serial cable adapter. For details, check sections 5.1 and 5.4 of:
  http://elinux.org/Beagleboard:BeagleBoneBlack

Here are the requirements for the labs. Just as in the first project, only a board, CCSv6 and Starterware are needed.

SHARING PROJECTS

The next slides will show the typical steps required to create a portable project in CCSv6.

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**

There are basically three types of projects based on the location of the files: a simple project that contains all source files copied inside its folder; a project that has all source files linked but it requires to be shared with everything in a complex directory structure; at last, a project that has all source files linked but only the project folder is shared and not all the source files (the assumption is that the project is shared among multiple users each of who has access to the source files)

Due to the popularity, only the first and the last case will be analyzed.

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

The simple projects have everything in the same directory, therefore sharing such project is simply a matter of creating a .zip file with everything inside. In the destination system, CCSv6 even imports the .zip file directly, not even needing to unzip it beforehand.

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

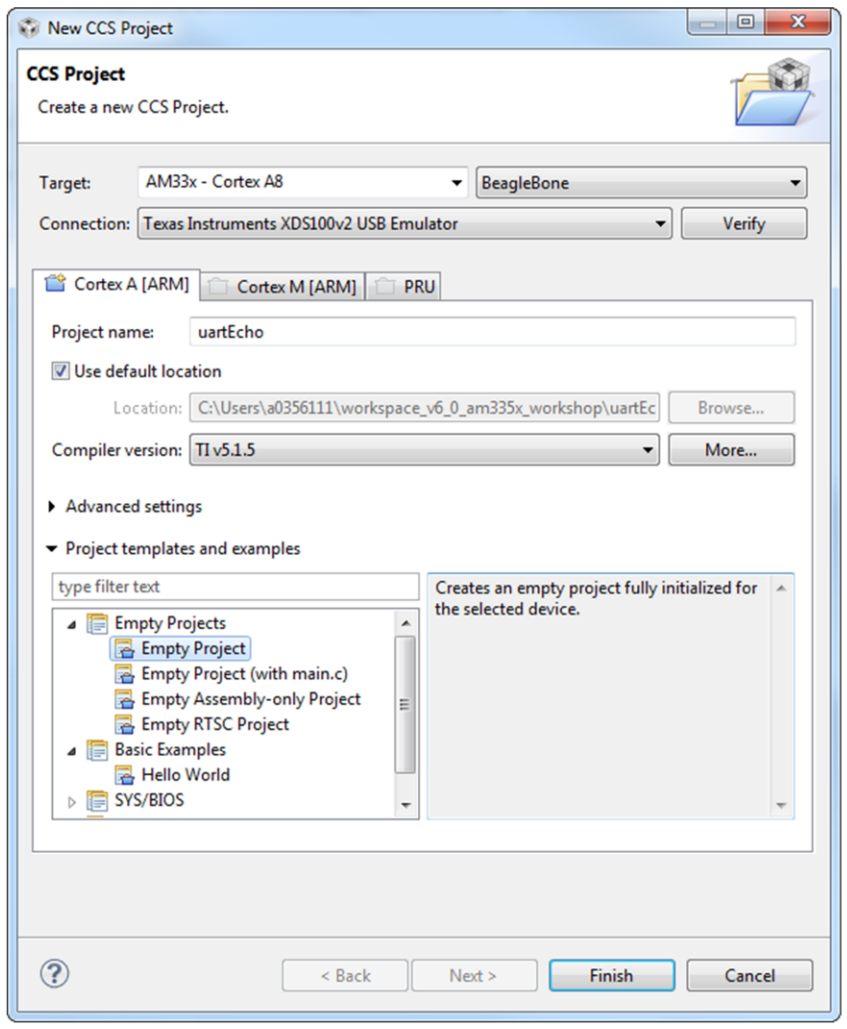
The sharing of a project with linked files (but no source) has some benefits, but the most important is that this method allows modifying the imported project without risking modifying the original source files, as the receiving end will have its own copy of them.

To illustrate, imagine that a product requires an SDK (such as Starterware), which contains a few libraries, header and source files and is installed by each user. The product also requires a few libraries, header and source files from the company, which are also installed on a different place on the user’s system. By creating a true portable project, this collection of files can be easily referenced and customized for each user’s system. Also, if the project itself is version controlled, the process of checking out modifications to it guarantees that another user is able to get the exact same project.

The next slides will briefly cover what is needed to create a portable project.

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 generated <AM335x.cmd> from the project



Nothing special is required to create a new portable project at this time. Simply start from the standard Project wizard in CCS with no modifications.

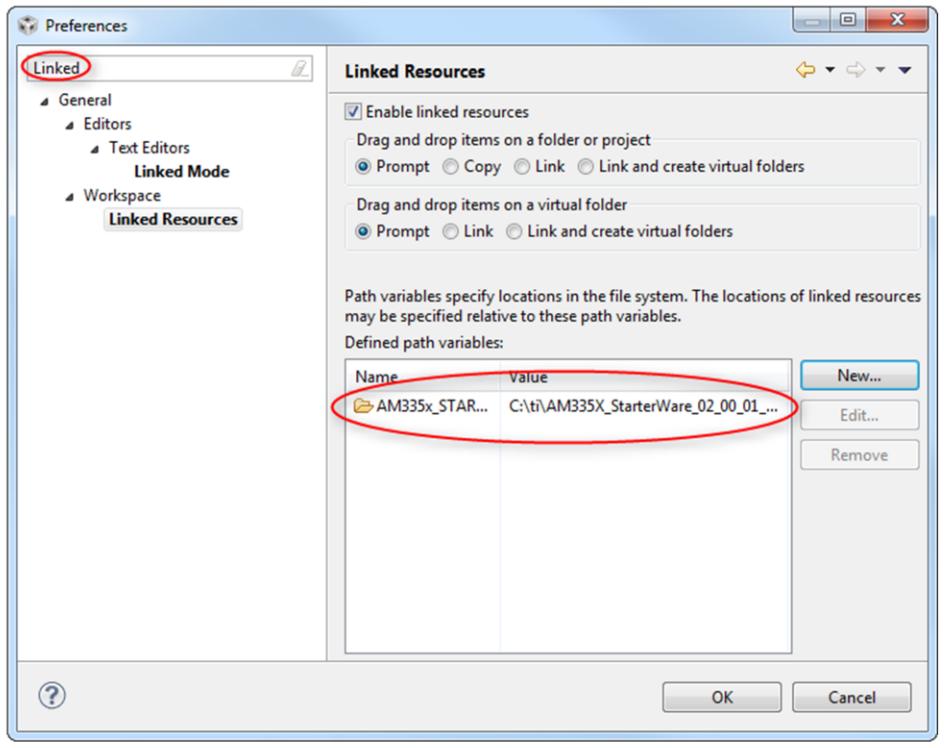
Create a Linked Resource Path Variable

- **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' (AM335x_STARTERWARE_ROOT) that points to the root location of the AM335x StarterWare directory

C:\ti\AM335X_StarterWare_02_00_01_01

- A Build variable of the same name (used in the compiler and linker options) is also automatically created.



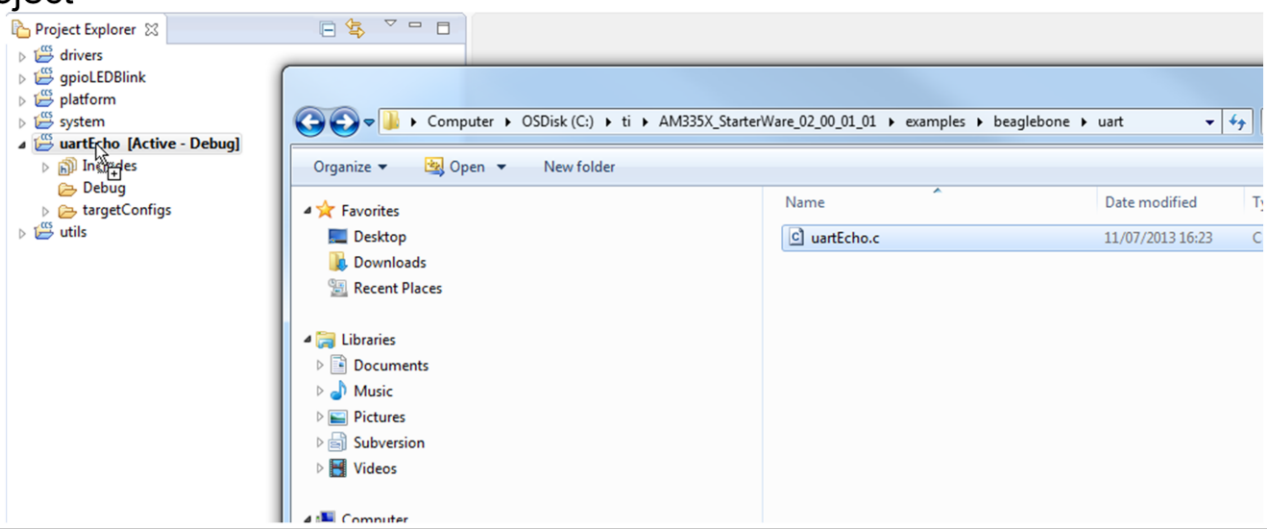
Once the project is created, the Linked Resource Path variable can be set in the workspace. This variable will guarantee that all files can be referenced to the Starterware installation directory.

In CCSv6 a Build variable is also automatically created under the hood. Build variables are used when specifying paths in the compiler and linker options, and they used to have to be set separately in CCSv5.

Link Source Files to Project

- Here we will link the source file relative to the Linked Resource Path Variable previously created
- Open Windows Explorer and browse to:
  C:\ti\AM335X_StarterWare_02_00_01_01\examples\beaglebone\uart

  – Drag and drop the following file into the new project in the CCS Project Explorer view
    – <uartEcho.c>

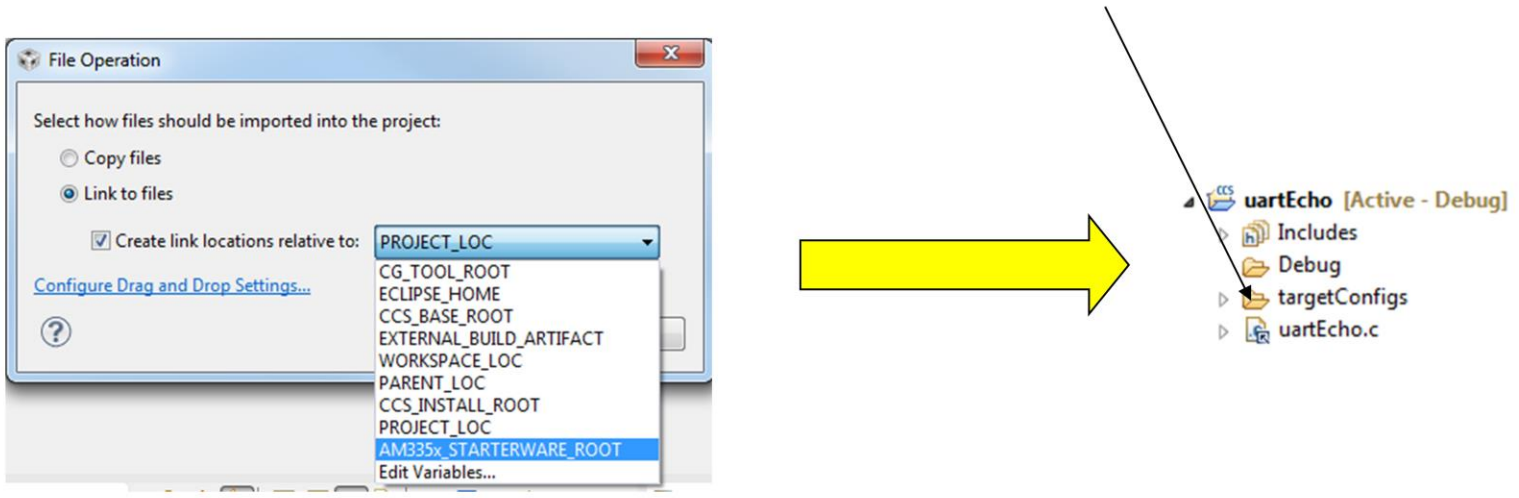


Texas Instruments

Linking files to a project is as easy as drag-and-drop, due to the Project Explorer's ability to act as a file explorer.

Link Source Files to Project

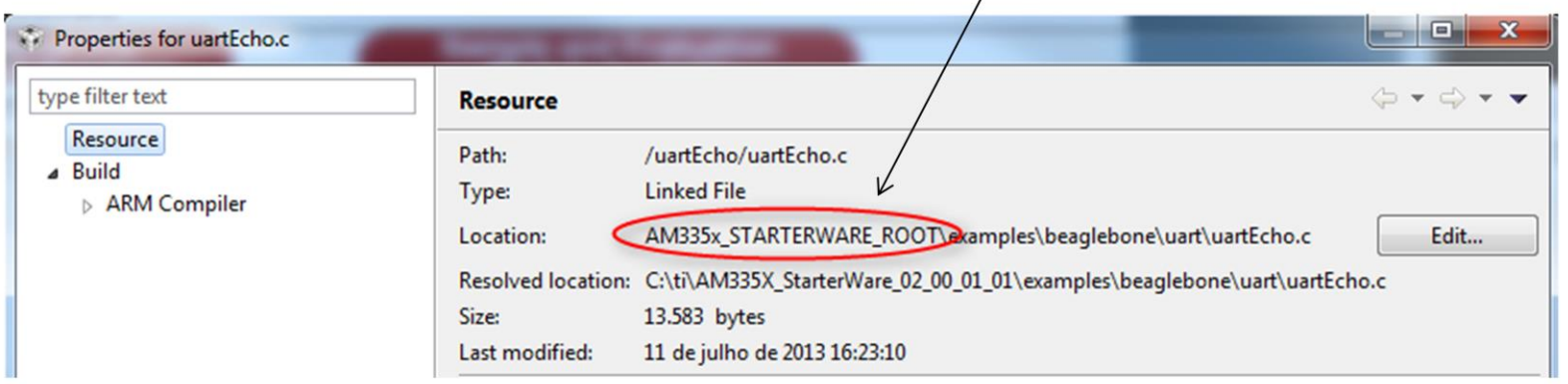
- A dialog will appear asking if you wish to Copy or Link the files:
  - Select *Link to files*
  - Select *Create link locations relative to:*
    - Use the new Linked Resource variable we created (**AM335x_STARTERWARE_ROOT**)
  - Hit *OK*
- Files will now appear in the Project Explorer with the ‘link’ icon



Once dropping the file on the Project Explorer view, CCSv6 asks what you want to do – either create a local copy of the original file, or create a link to the file and leave it in the original location.

Link Files to Project

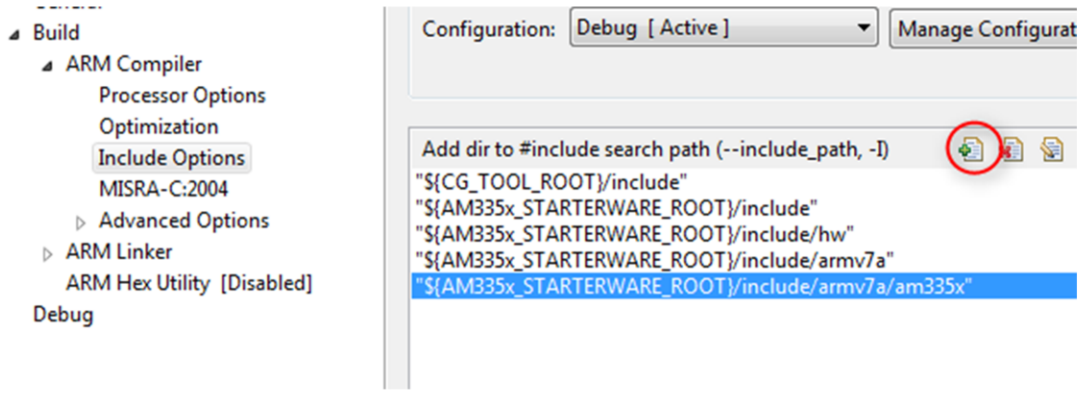
- Right-click on the C source file and check the *Properties*
  - See how the *Location* parameter references the Linked Resource Variable



To verify if the link is correct, simply right-clicking on the file reveals both its linked location and its physical location.

Modifying Project Properties

- Here we are adding paths to include files using the Build Variable
- Right-click on the project and select *Properties*
- In the compiler *Include Options*, add the following entries to the list of include search paths:
  - ${AM335x_STARTERWARE_ROOT}/include
  - ${AM335x_STARTERWARE_ROOT}/include/hw
  - ${AM335x_STARTERWARE_ROOT}/include/armv7a
  - ${AM335x_STARTERWARE_ROOT}/include/armv7a/am335x
- Click *OK*
- **'${<BUILD VARIABLE>}' is the syntax to use a Build Variable in the project properties**
- **NOTE: Linked Resource Path Variables are used when linking source files to a project. Build Variables are used in build options. In CCSv5 these variables had to be created separately. In CCSv6, creating a Path Variable also creates a Build Variable with the same name.**



Texas Instruments

For a project to be truly portable, the compiler and linker include directories also must be set – this way the build tools will be able to recognize the variable created before.

One common mistake is to use relative paths in these options (a sequence of ../../..), but these break the portability if the relative location of the included files changes – a common scenario if the software packages are installed in different directories, for example.

Project vs Workspace Level Variables

- *Linked Resource Path Variables* and *Build Variables* can be set at the project level
- This current lab sets 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!

The approach shown uses build and linked resource path variables set in the workspace and not directly on the project. The main benefit is that you can share the project without having to modify it, therefore rendering identical copies across a team.

Also, setting these variables at the workspace level allows to accommodate for variations that each user may have on their system – meaning install directories of libraries, CCS itself, etc. Including these on the project itself will force users to modify the project's variables before being able to use them.

Any questions?

LAB 2: UART ECHO EXAMPLE

30 MINUTES

Please refer to the file C:\TI\Cheat_sheet.txt for paths for the variables

Texas Instruments

Now let's start working on the lab. 30 minutes are typically needed to finish everything.