



Code Composer Studio v5

Overview

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

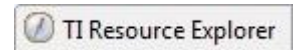
Simplified user interface

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

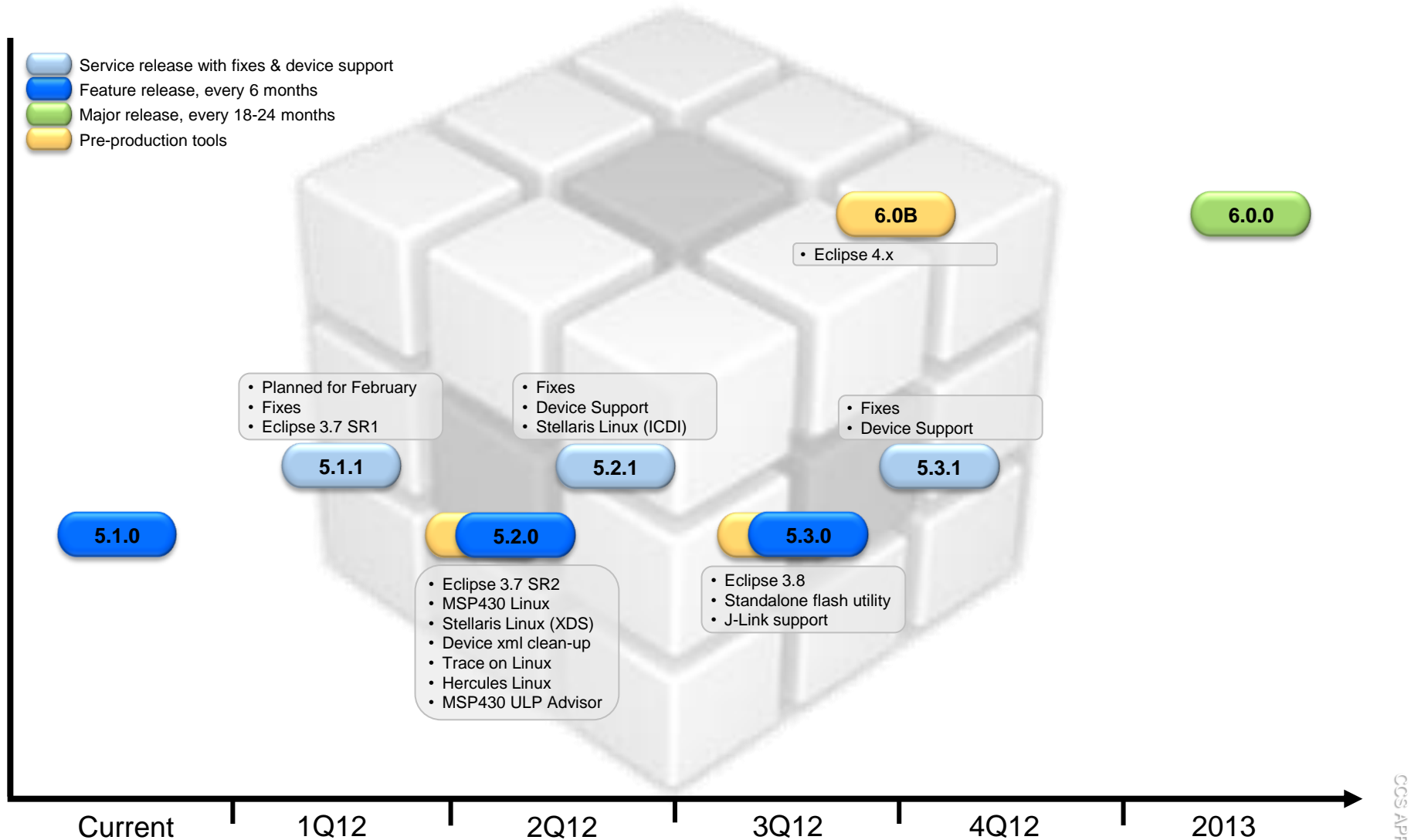
explain how to get the most out of features.



*MSP430, Stellaris MCU support available early 2012

Code Composer Studio™ Roadmap

- Service release with fixes & device support
- Feature release, every 6 months
- Major release, every 18-24 months
- Pre-production tools



Improvements for CCSv5.1

What's new in Eclipse?

- **A lot!**

- CCSv5.1 uses Eclipse 3.7, CCSv4 uses Eclipse 3.2
- 5 years of fixes & enhancements

- **Key items**

- Editor/Indexer improvements
 - Most common area of Eclipse related problems in CCSv4
 - Much faster
 - Much more reliable
- Drag & drop support
- Support for using macros when linking files (portable projects)
- Dynamic syntax checking
- Search for plug-ins from inside Eclipse

- **http://processors.wiki.ti.com/index.php/CCSv5_Changes**

Customer Feedback on CCSv4



- **Needs to be Smaller**

- The CCS DVD image is huge (>1GB to download, >4GB on disk)
- Need to download a lot of things that you do not need

- **Needs to be Faster**

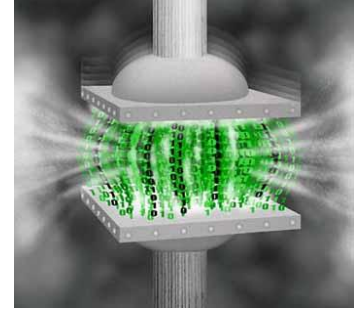
- Product is slow
- Startup times and debugger responsiveness needs to be improved

- **Needs to be Easier**

- The user interface is too cluttered
- Difficult to figure out how to get started

- **Thus the objective for 5.1 is to make CCS “Small, Fast & Easy”**

Small



- **CCSv4**

- Download size is 1.2GB
- Separate code size limited and DVD images (users often download the wrong one)
- Users have to download much more than they need

- **CCSv5.1**

- User downloads a small initial installation package
- Based on user selections the appropriate packages will be downloaded and installed dynamically
- User can add more features later
- Optionally users can download the complete DVD image

Fast

- **Speed up common tasks**

- Starting up CCS
- Launching a debug session
- Creating a new project (initial experience needs to be awesome)

- **Responsiveness**

- While using the product CCS needs to be responsive
 - Stepping (with views open)
 - Continuous refresh in real-time mode with expressions view and graphs open
 - Saving target configurations
 - Loading/Flashing programs



Easy – User Interface Modes



- **Simple Mode**

- By default CCSv5 opens in simple/basic mode
- Simplified user interface with far fewer menu items, toolbar buttons
- TI supplied Edit and Debug Perspectives
- Simplified build options

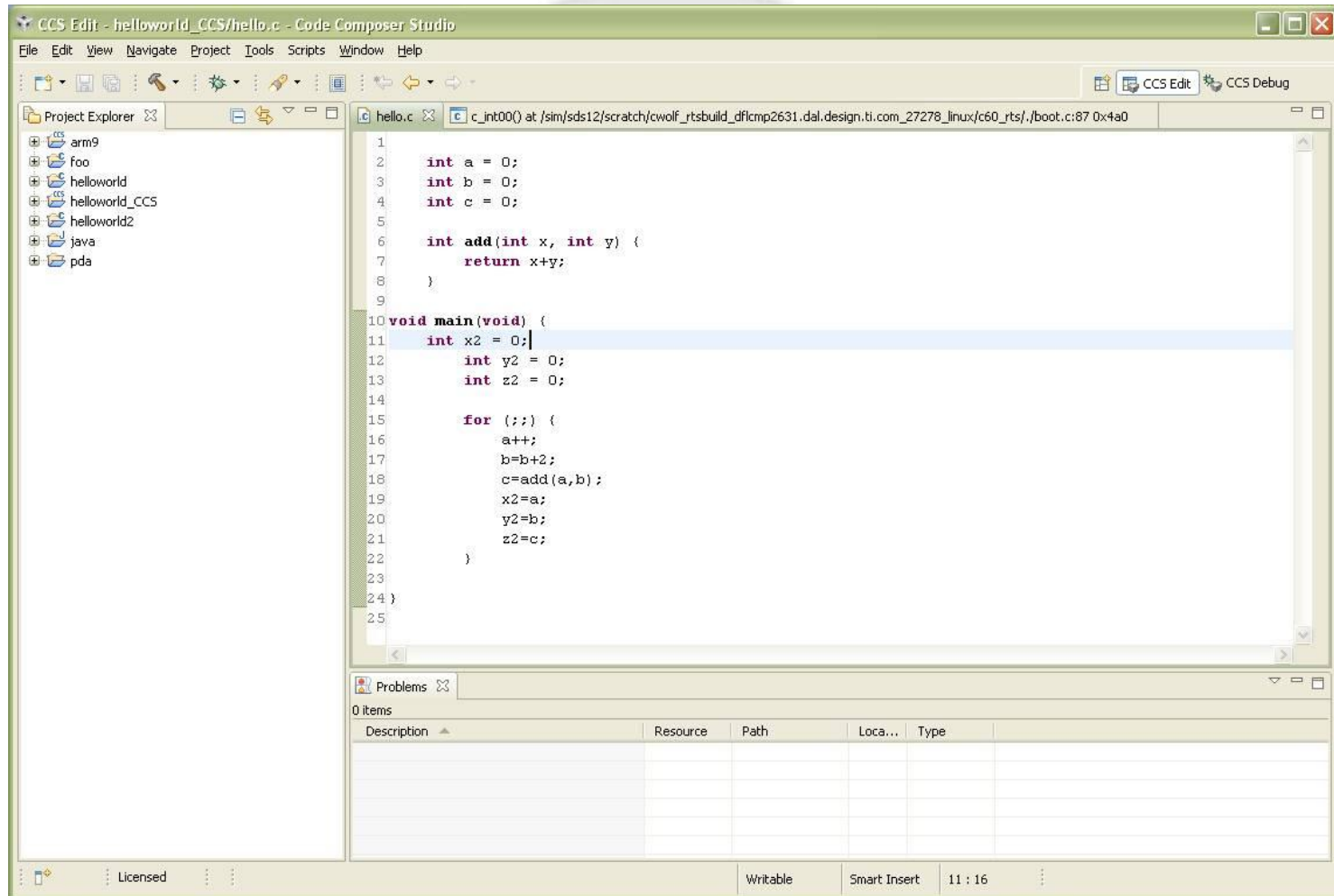
- **Advanced Mode**

- Use 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**

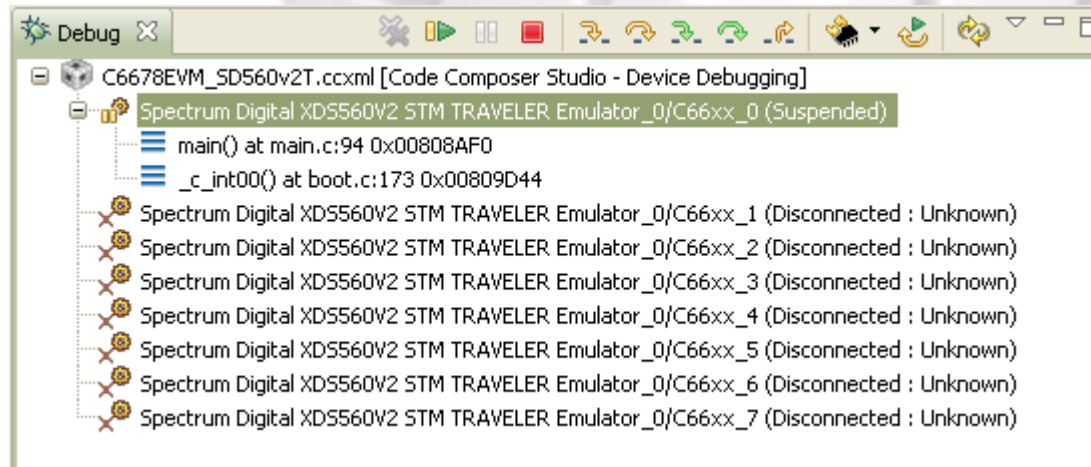
- Users may decide that they are ready to move from simple to advanced mode or vice versa

Simple Mode



Simplified Views – Debug View

- **Cleaner, simpler, and more customizable**
 - No extra “thread” node
 - No CIO/target error nodes
 - Multi-CPU devices listed directly below Project / Target Configuration node
 - Customizable to display more or less of JTAG hierarchy
 - Optionally collapsed to a single line to optimize screen real-estate (“Breadcrumb” mode)



Easy– Common tasks



- **Creating New Projects**

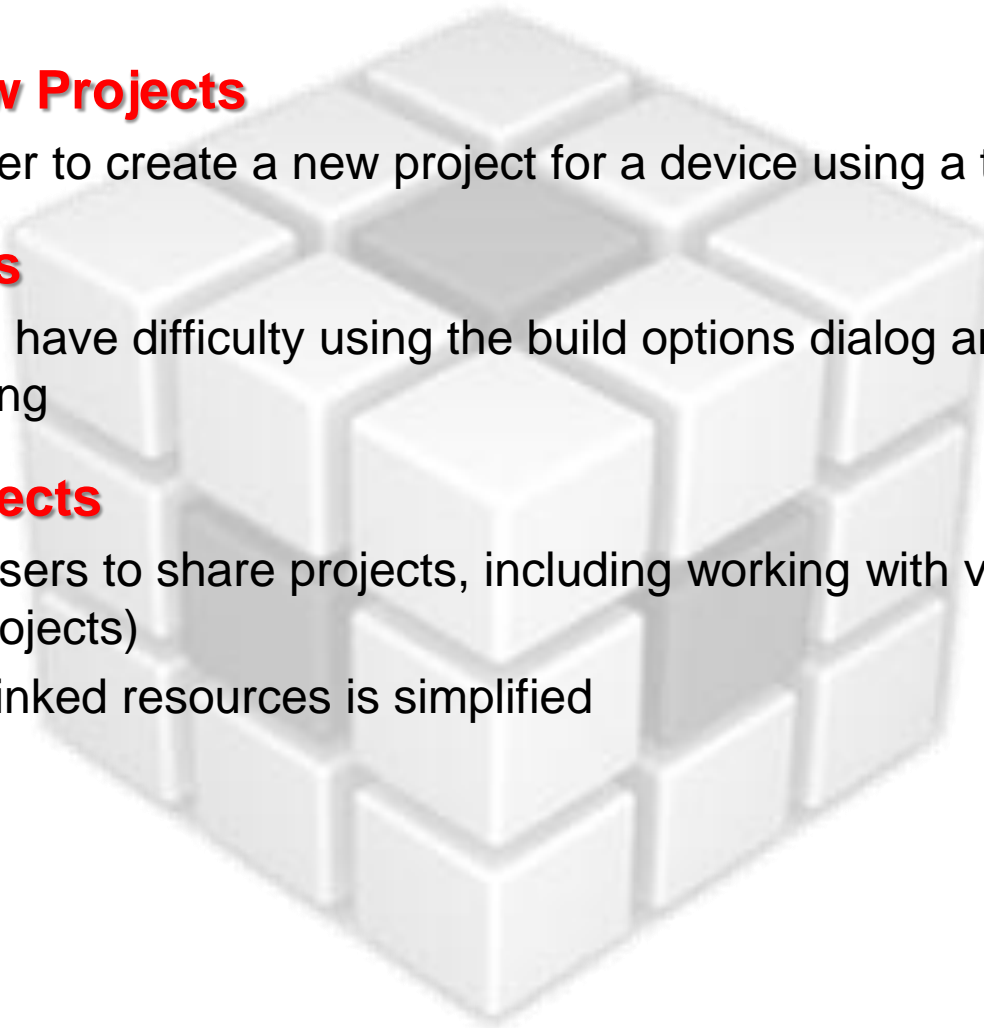
- Much simpler to create a new project for a device using a template

- **Build options**

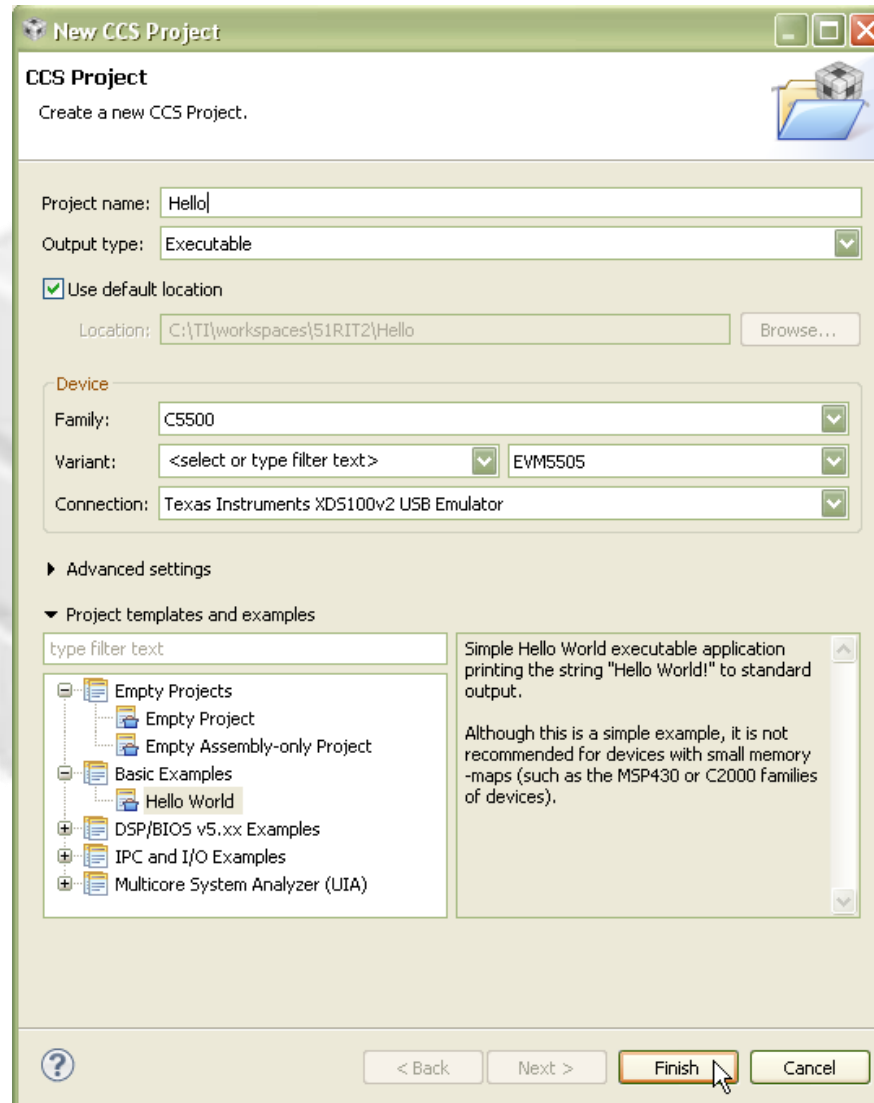
- Many users have difficulty using the build options dialog and find it overwhelming

- **Sharing projects**

- Easier for users to share projects, including working with version control (portable projects)
- Setting up linked resources is simplified



New Project Wizard – 1 page & done



The screenshot shows the 'New CCS Project' dialog box. The title bar says 'New CCS Project'. Below the title bar, it says 'CCS Project' and 'Create a new CCS Project.' with a folder icon. The 'Project name' field contains 'Hello'. The 'Output type' dropdown is set to 'Executable'. The 'Use default location' checkbox is checked. The 'Location' field shows 'C:\TI\workspaces\51RIT2\Hello' with a 'Browse...' button. The 'Device' section has 'Family' set to 'C5500', 'Variant' set to '<select or type filter text>' with a dropdown arrow, and 'Connection' set to 'Texas Instruments XD5100v2 USB Emulator' with a dropdown arrow. Below the device section, there is a 'Project templates and examples' section with a search bar containing 'type filter text'. A tree view shows 'Empty Projects' (with sub-items 'Empty Project' and 'Empty Assembly-only Project'), 'Basic Examples' (with sub-item 'Hello World'), 'DSP/BIOS v5.xx Examples', 'IPC and I/O Examples', and 'Multicore System Analyzer (UIA)'. To the right of the tree view, there is a text area describing the 'Hello World' example: 'Simple Hello World executable application printing the string "Hello World!" to standard output. Although this is a simple example, it is not recommended for devices with small memory -maps (such as the MSP430 or C2000 families of devices).' At the bottom, there are buttons for '< Back', 'Next >', 'Finish' (highlighted with a mouse cursor), and 'Cancel'.

New CCS Project

CCS Project
Create a new CCS Project.

Project name: Hello

Output type: Executable

☒ Use default location

Location: C:\TI\workspaces\51RIT2\Hello Browse...

Device

Family: C5500

Variant: <select or type filter text> EVM5505

Connection: Texas Instruments XD5100v2 USB Emulator

▶ Advanced settings

▼ Project templates and examples

type filter text

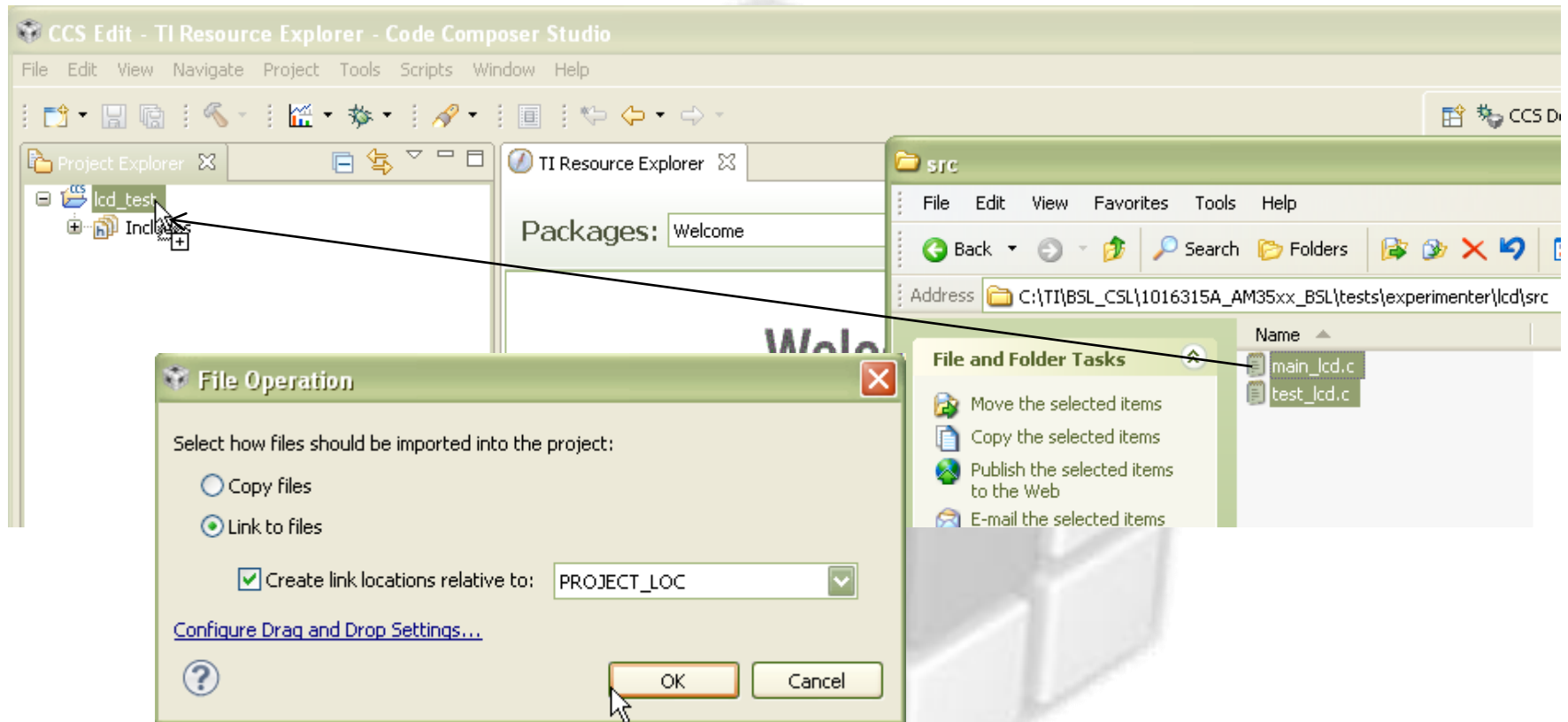
- Empty Projects
 - Empty Project
 - Empty Assembly-only Project
- Basic Examples
 - Hello World
- DSP/BIOS v5.xx Examples
- IPC and I/O Examples
- Multicore System Analyzer (UIA)

Simple Hello World executable application printing the string "Hello World!" to standard output.

Although this is a simple example, it is not recommended for devices with small memory -maps (such as the MSP430 or C2000 families of devices).

? < Back Next > Finish Cancel

Linking Source Files to Project



Easy – Help & Documentation

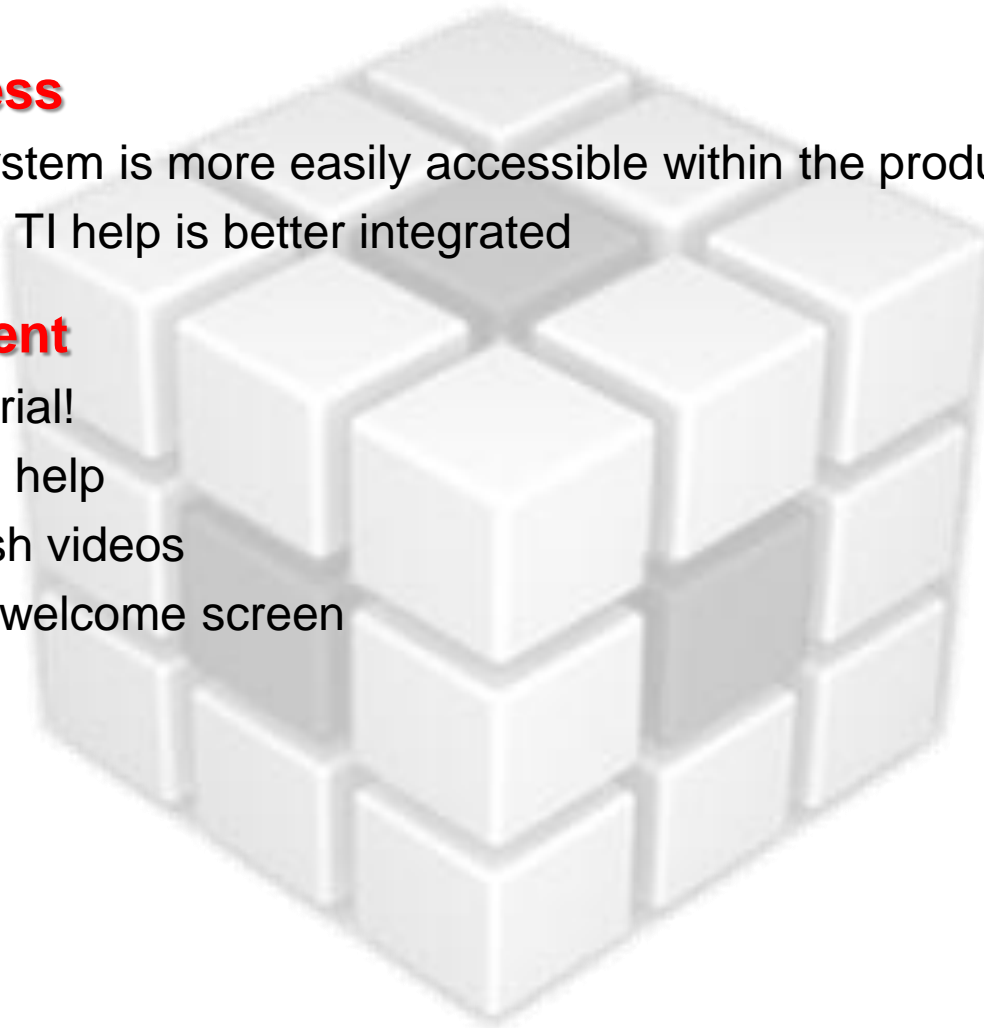


- **Easy to access**

- The help system is more easily accessible within the product
- Eclipse and TI help is better integrated

- **Quality content**

- A good tutorial!
- Task based help
- Links to flash videos
- Revamped welcome screen



Resource Explorer - Tutorials

The screenshot displays the CCS Edit - TI Resource Explorer - Code Composer Studio window. The left pane shows a tree view of example projects, with 'CPU Timer' selected under the 'Piccolo F2802x' device. The right pane shows the 'CPU Timer' tutorial page, which includes a description of the example program and three steps for importing, building, and debugging the project.

TI Resource Explorer

Packages: All Devices: All Topics: All

URL:

CPU Timer

Example program which configures CPU Timers 0, 1, and 2, and increments a counter each time the timer asserts an interrupt.

These are the steps to import the project, build the project, and debug the project.

Step 1: [Import the example project into CCS](#)

Click on the link above to import the project. The imported project is available in the **Project Explorer** view, expand the project node to browse the imported source files. To modify source code, double clicks on the source file within the project to open the source file editor.

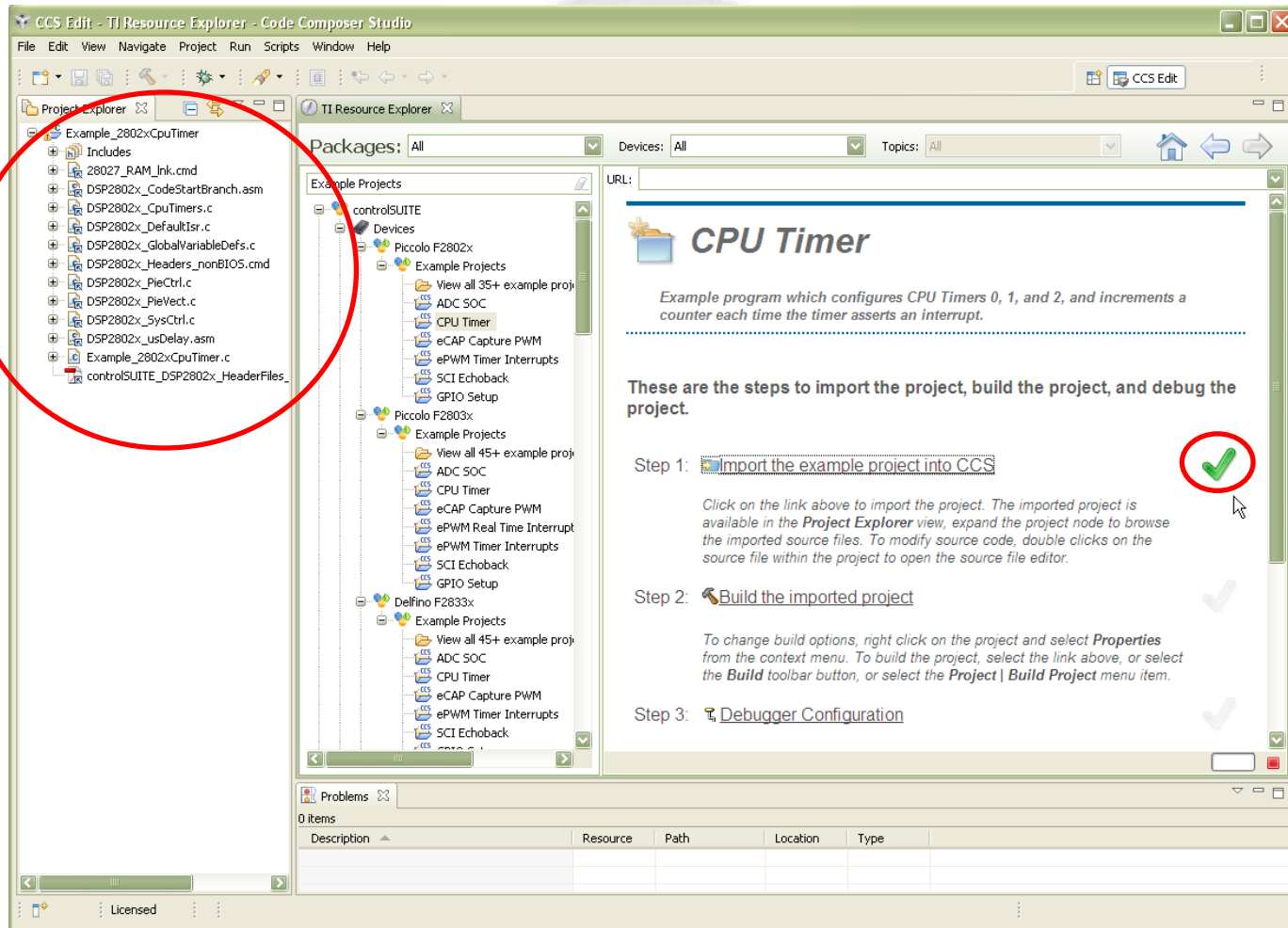
Step 2: [Build the imported project](#)

To change build options, right click on the project and select **Properties** from the context menu. To build the project, select the link above, or select the **Build** toolbar button, or select the **Project | Build Project** menu item.

Step 3: [Debugger Configuration](#)

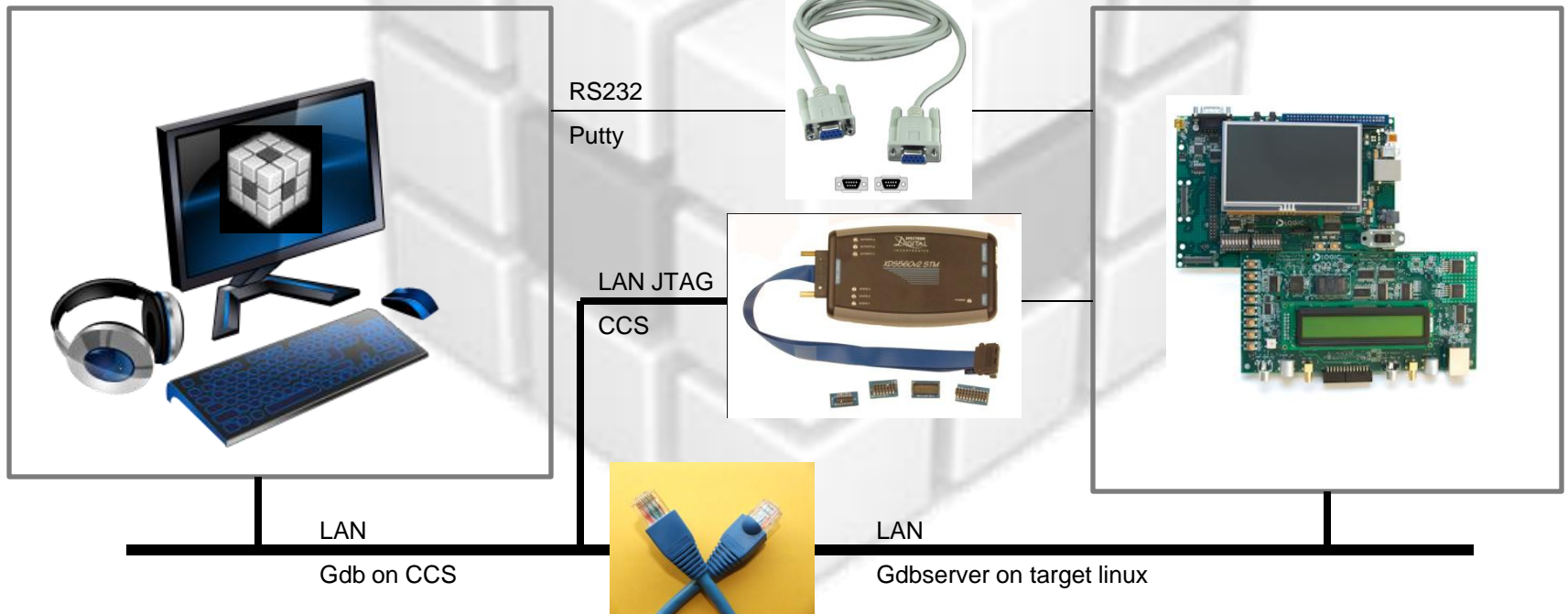
Connection: **none**
Click on the link above to change the device connection. Additionally, this option is

Resource Explorer - Tutorials



Linux Development

- CCS supports both Windows and Linux host PCs
- Linux application debug supported via integrated GDB
- Linux kernel debug supported via JTAG debug



Linux Application/Kernel Debug

The screenshot displays the CCS Debug interface for a C/C++ Remote Application. The left pane shows the project structure with 'GCC_sinewave_A8 Debug' selected. The right pane shows the 'Variables' window with a table of variables.

Name	Type	Value
howlongtosleep	int	35052

A callout box points to the 'main()' entry in the project tree, stating: "Change debug context to switch between application debug and kernel debug".

The bottom pane shows the source code for 'sinewave_float.c' with the following code:

```
14 float results2[SIZE_DATASET];
15 unsigned int count = 0;
16
17 int main(void)
18 {
19     int howlongtosleep = 1;
20
21     printf("Value of PI is: %f", PI);
22     for (count = 0; count < SIZE_DATASET; count++)
23     {
24         results1[count] = 0;
25         results2[count] = 0;
26     }
27     for (count = 0; count < SIZE_DATASET; count++)
28     {
29         data1[count] = 4 * PI * ((float)count / 1024);
```

The bottom right pane shows the terminal output for the 'GCC_sinewave_A8 Debug' session:

```
am3517-evm login: root
root@am3517-evm:~# gdbserver :10000 GCC_sinewave_A8
Process GCC_sinewave_A8 created; pid = 1810
Listening on port 10000
Remote debugging from host 158.218.99.169
```


Migration

- **Moving from CCSv3 to CCSv4 was hard**
 - Completely different environment
 - New project system
 - Target configuration changes
 - The CCS world changed...
- **CCSv4 to CCSv5.1 migration is much smoother**
 - Environment is very similar
 - Project system is the same (simple import)
 - Target configuration is the same
- **CCSv3 to CCSv5.1 migration**
 - The team continues to make improvements to the v3 import wizard
 - UI simplifications help with the learning curve
 - Improved documentation helps people get up to speed

Pricing



- **Free**

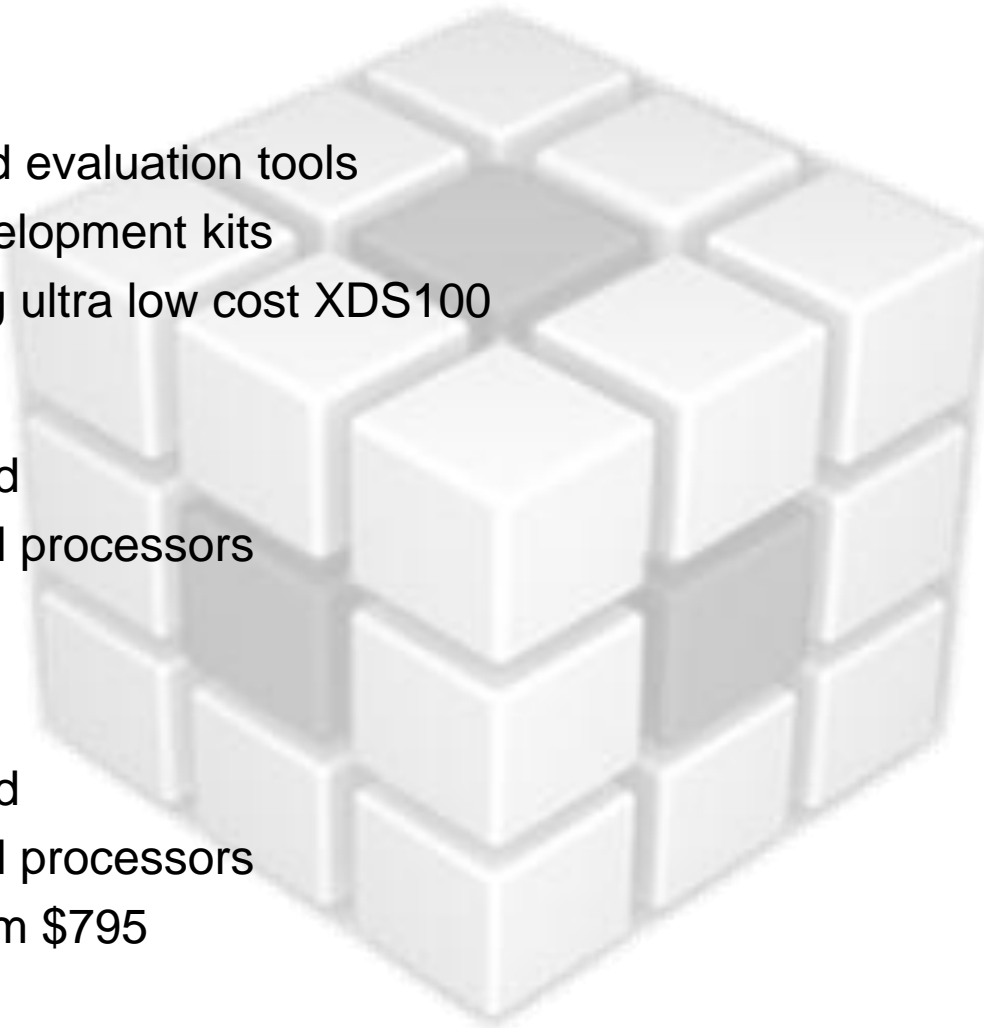
- Time limited evaluation tools
- Tied to development kits
- When using ultra low cost XDS100

- **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](#)
 - Targeted response time within 24 hours
- **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

Trusted Code Composer Studio IDE features and benefits

Code development environment

eases and speeds design and troubleshooting.

Advanced GUI framework

simplifies project management.

Multi-processor debugging

manages status and information from multiple cores.

Flexible project environment

aids control of compiler & SYS/BIOS™ sw kernel foundation.

Debug server scripting interface

enables the automation of common tasks.

Update manager tool

automatically manages tool updates.



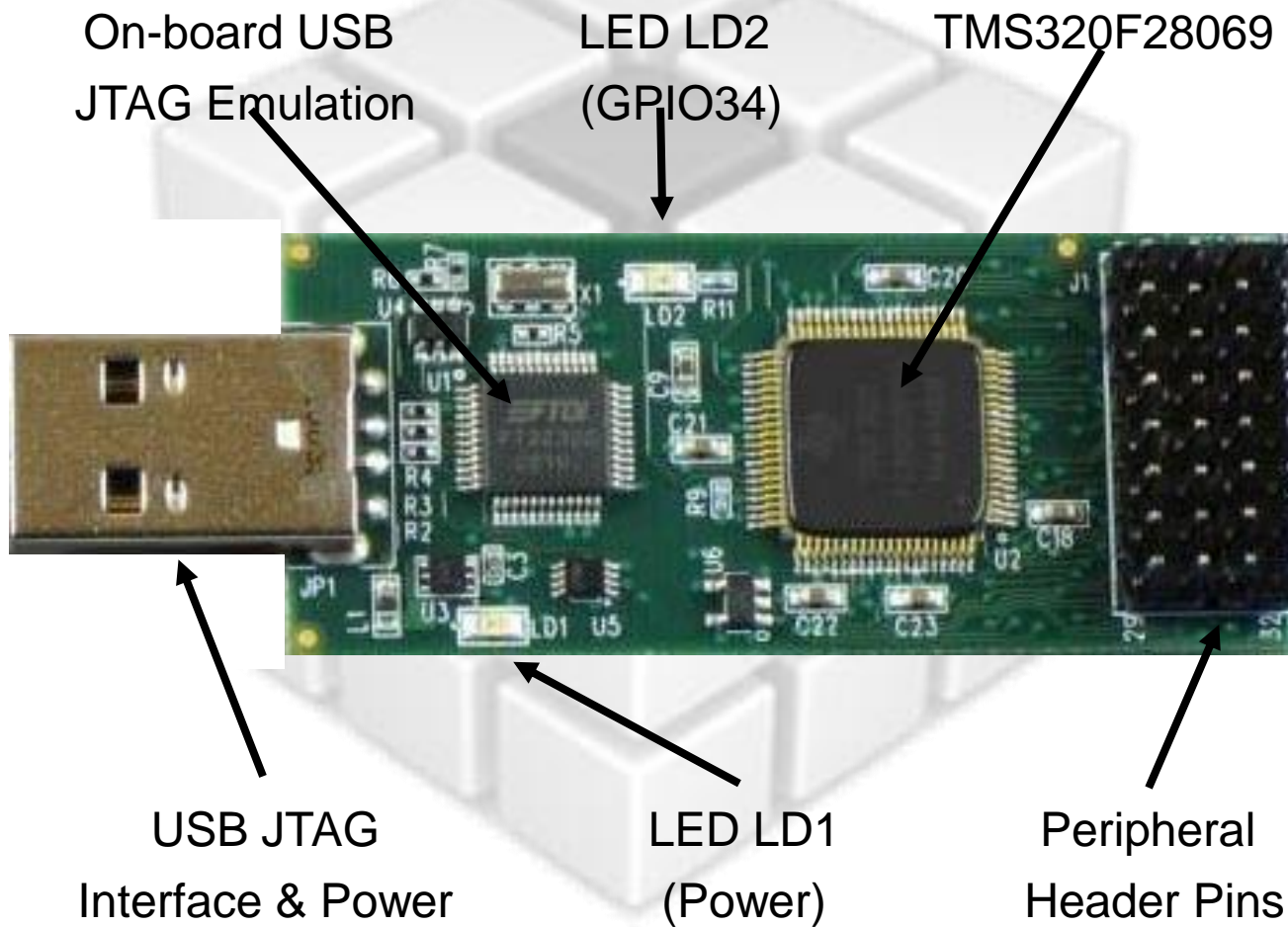
Getting Started with CCSv5 and TMS320F28069 controlSTICK

What is TMS320F28069 controlSTICK?

- Piccolo F28069 microcontroller
- Quick and easy evaluation of all of the advanced capabilities of TI's new Piccolo F2806x microcontroller for just \$39
- Convenient and easy-to-use GUI provides hands-on experimentation with the floating point capabilities of the Piccolo F2806x MCU.
- Slightly larger than a memory stick
- On-board emulation, access to all I/O pins
- Detailed example software and documentation
- Complete hardware schematics, gerber files, etc



TMS320F28069 controlSTICK



Workshop Instructions

- This workshop is a mix of presentation material and activities
- Whenever you see the “Let’s Do it” picture on a slide, there are actions for you to perform





BLINKING LED EXAMPLE

Blinking LED Example: Exercise Summary

- **Key Objectives**

- Create and build a simple program to blink LD2
- Start a debug session and load/flash the program on the controlSTICK
- Run the program to blink LD2

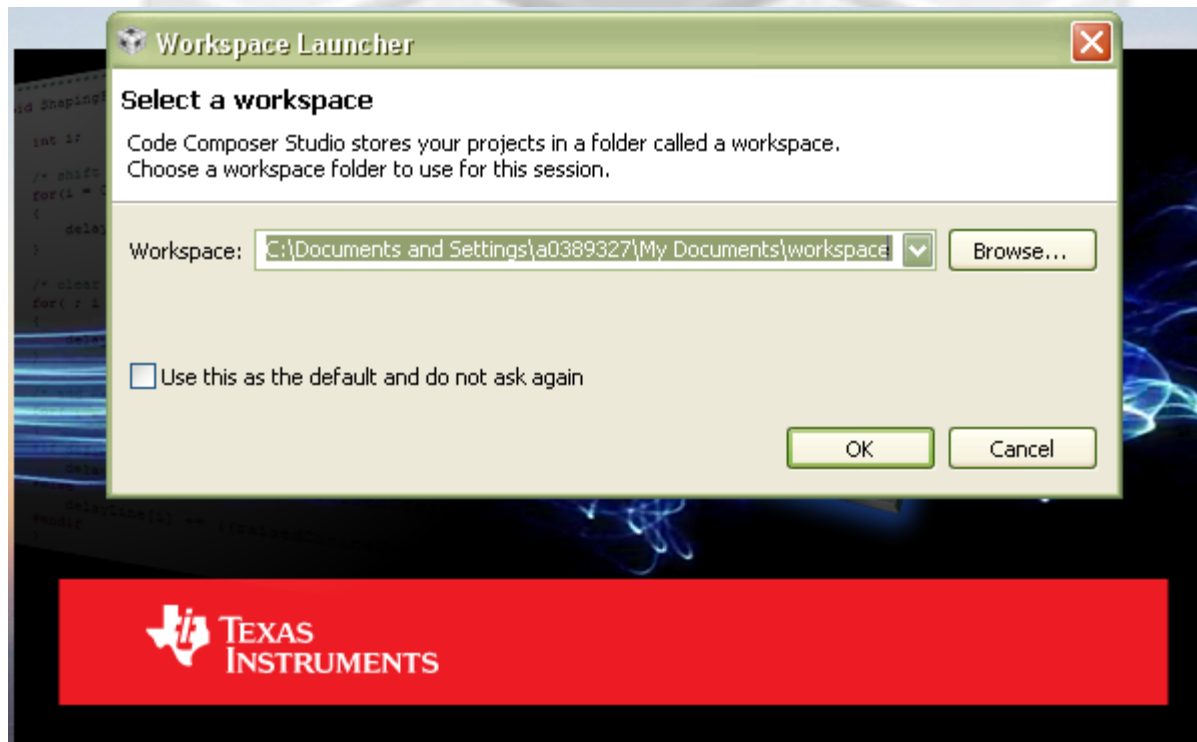
- **Tools and Concepts Covered**

- Workspaces
- Welcome screen / Resource Explorer
- Project concepts
- Basics of working with views
- Debug launch
- Debug control
- Profile Clock
- Local History
- Debugging existing code in flash (Loading Symbols)
- Build Properties
- Changing compiler versions


Workspace



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



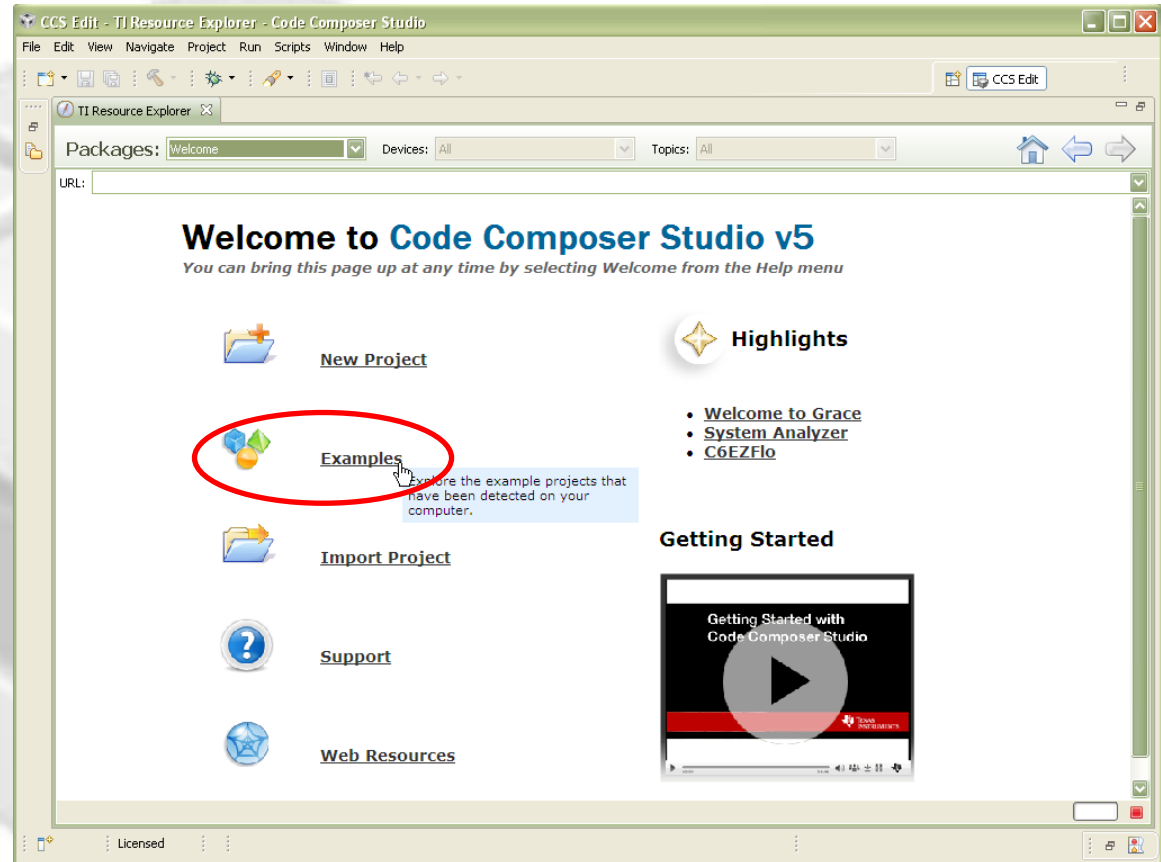
Eclipse Concept: Workspaces

- **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**
-  **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

Resource Explorer: Welcome



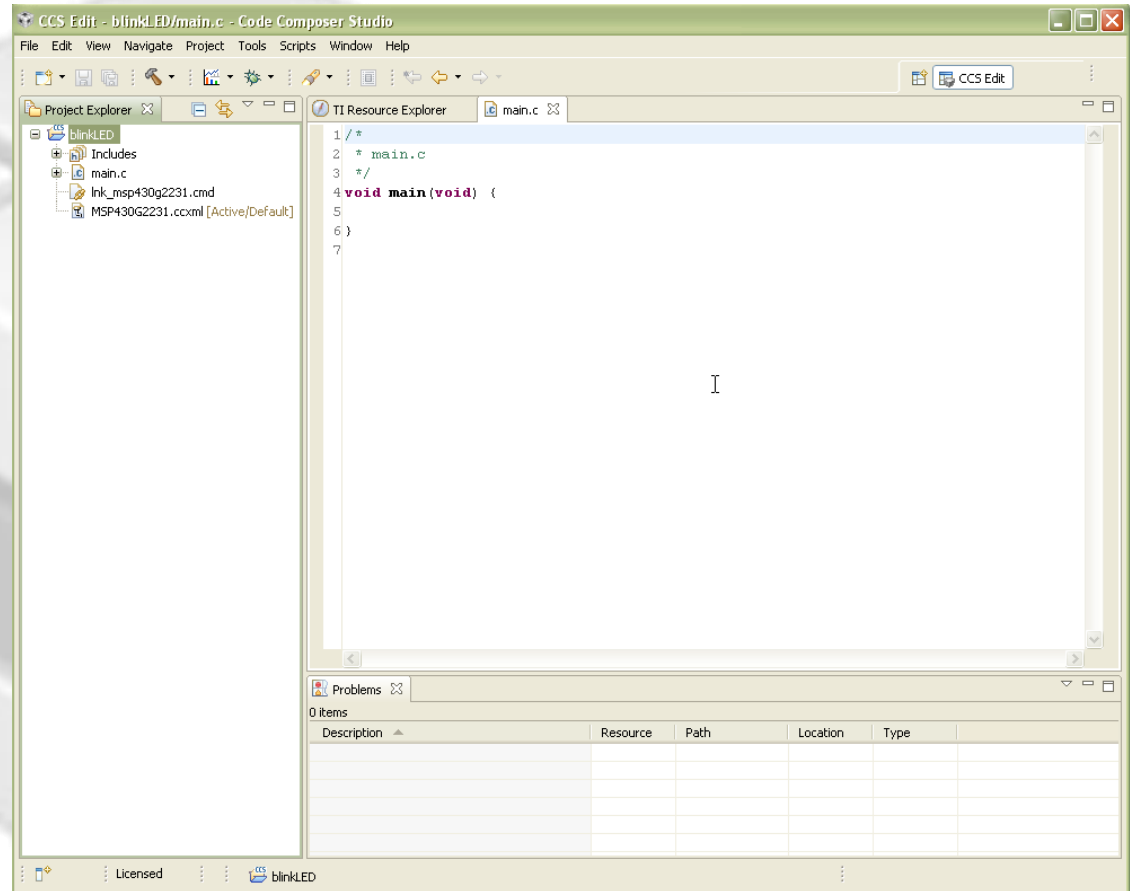
- The 'TI Resource Explorer' will display the 'Welcome' page the first time CCS is used with a new workspace
- Getting Started video introduces you to CCS
- Contains links to documentation, examples, support resources
- Buttons to create a new project or import an existing one into your workspace
- 'Help->Welcome to CCS' to return to the Resource Explorer in the future



Eclipse Concept: Workbench



- **‘Workbench’** refers to the main CCSv5 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**



Workbench (Comparison with CCSv3.x)

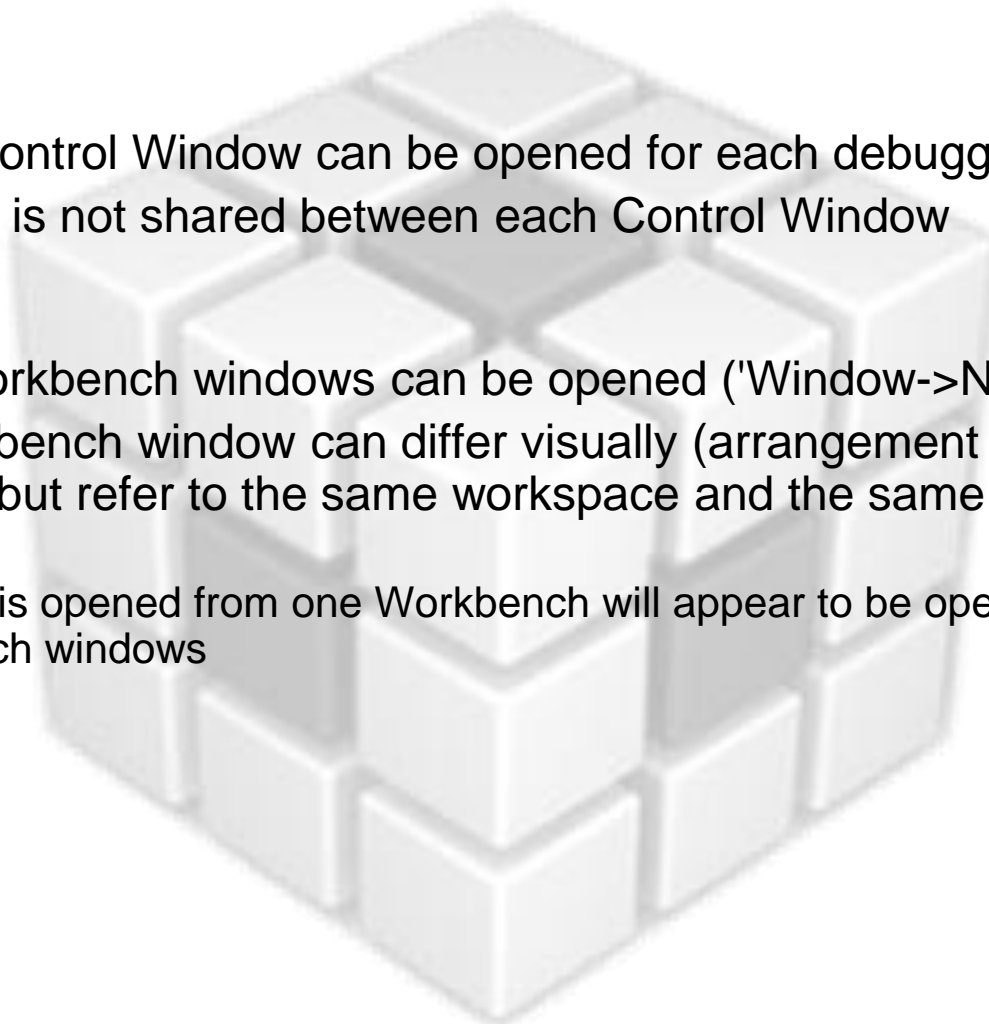


- **CCS 3.x**

- Only one Control Window can be opened for each debuggable CPU
- Information is not shared between each Control Window

- **CCS 4/5**

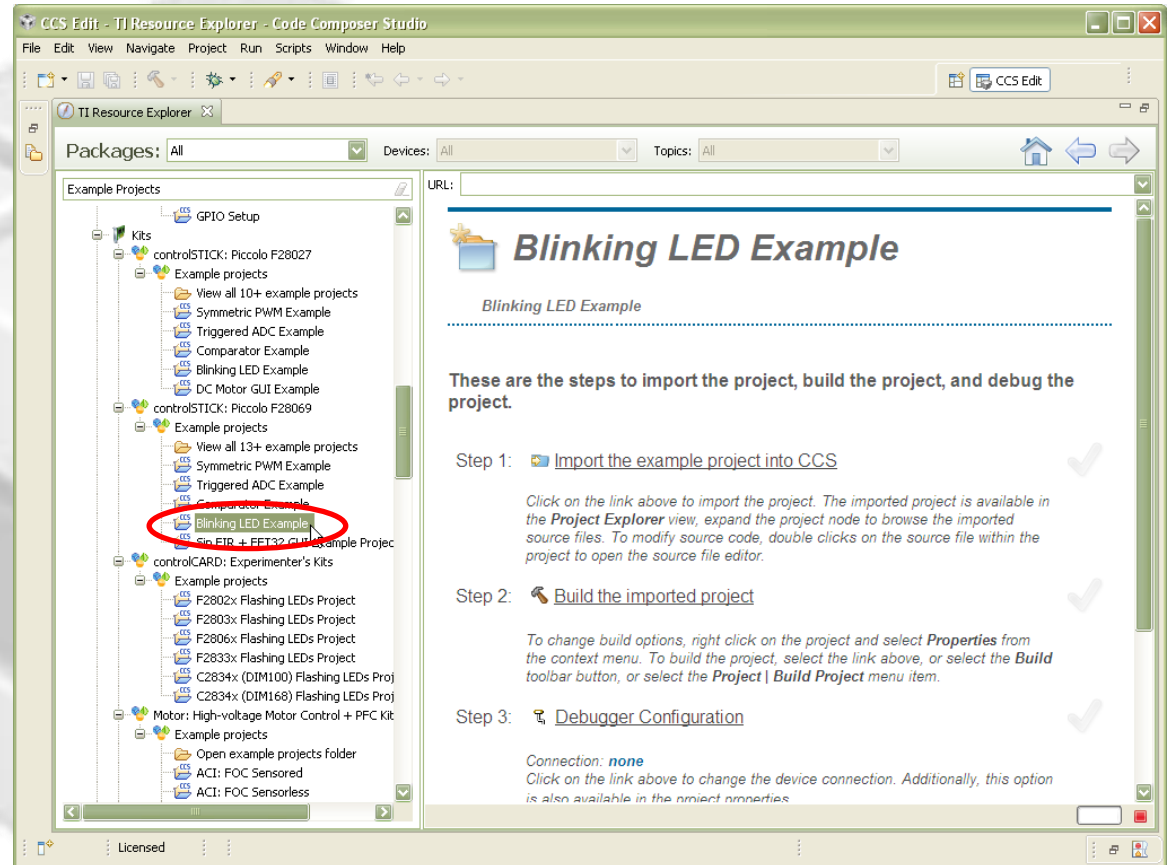
- Multiple Workbench windows can be opened ('Window->New Window')
- Each Workbench window can differ visually (arrangement of views, toolbars and such), but refer to the same workspace and the same running instance of CCSv4
 - A project is opened from one Workbench will appear to be open in all the Workbench windows



Resource Explorer: Examples



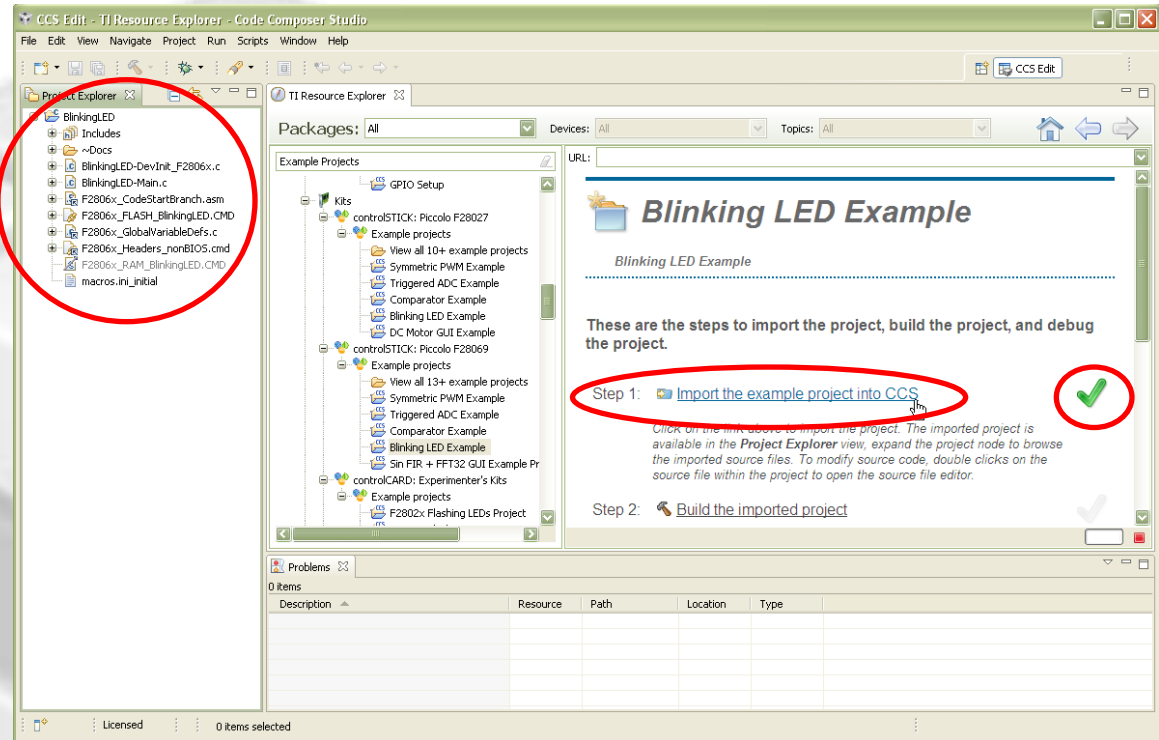
- Displays all example projects discovered by the Resource Explorer
- Step by step instructions on how to build/run the examples
- Select the “Blinking LED Example” project under “controlSUITE -> Kits -> controlSTICK: Piccolo F28069”



Step 1: Import 'Blinking LED' Project





- Follow Step 1 and import the "Blinking LED" project into the CCS Workspace by clicking on the "Import the example project into CCS" link
- The project will appear in the 'Project Explorer' if the import is successful and a green checkmark will appear next to the link to indicate success



Eclipse Concept: Projects

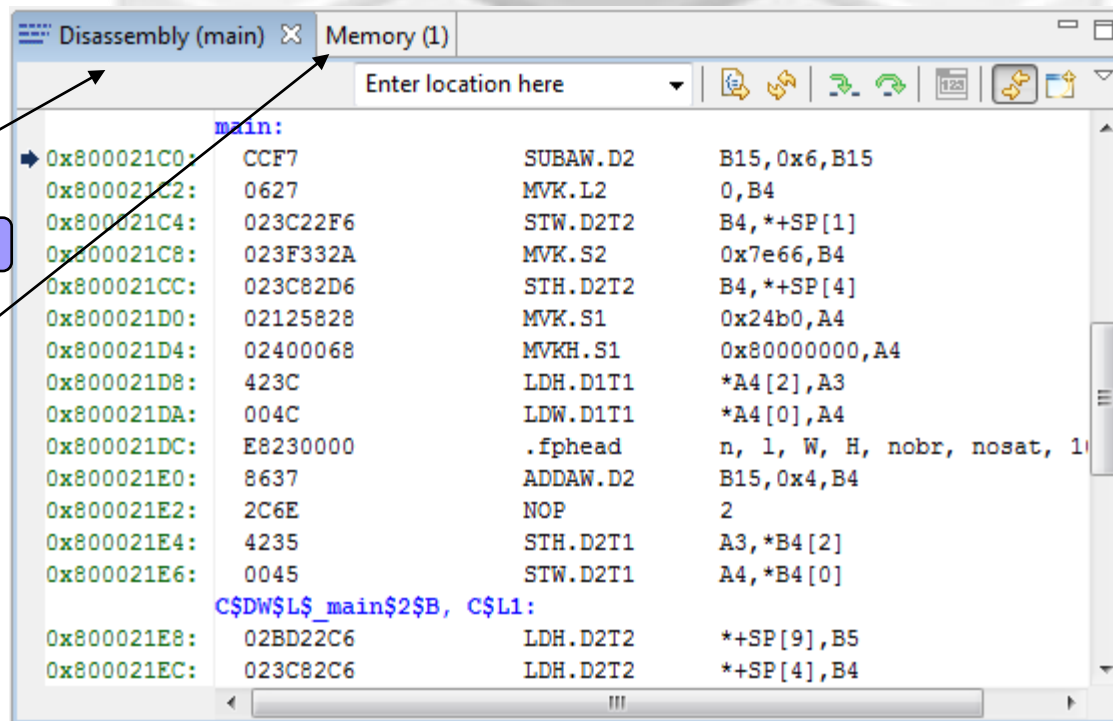


- **Projects map to directories in the file system**
- **Files can be added or linked to the project**
 - Adding file to project
 - Copies the file into your project folder
 - Linking file to project
 - Makes a reference to the file in your project
 - File stays in its original location CCS 3.x project 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, CCE projects and legacy CCSv3 projects can be imported into the workspace

Eclipse Concept: Views

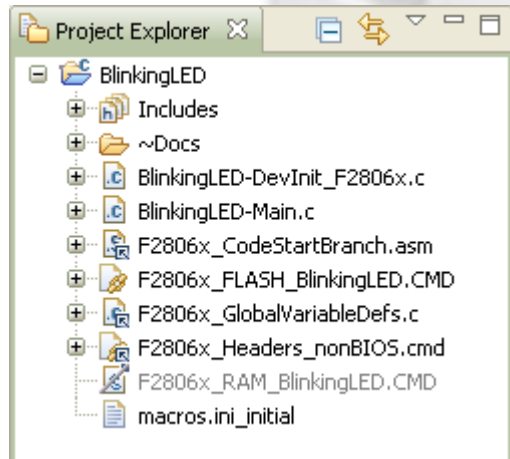


- **Views are windows within the main Workbench window that provide visual representation of some specific information.**
 - Most views can be accessed in the menu View
 - Can be identified by the organization in tabs



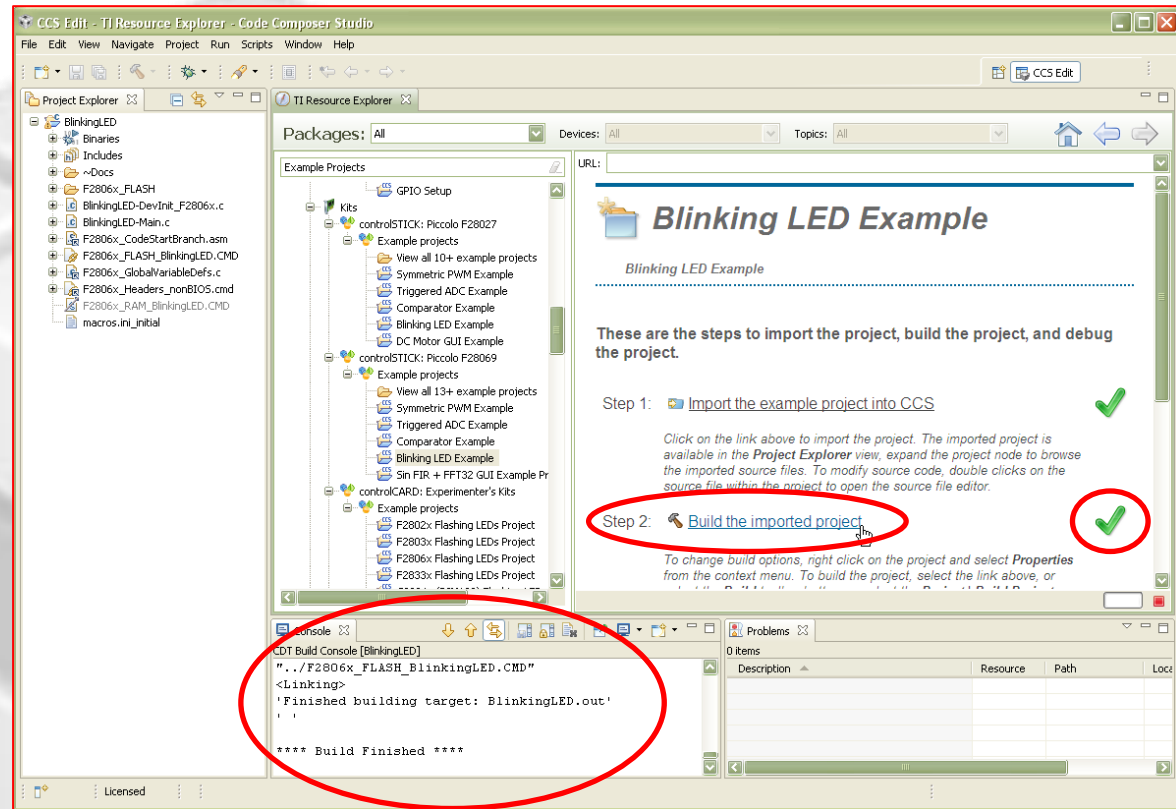
View: Project Explorer

- 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 (*.*)



Step 2: Build 'Blinking LED' Project

- Follow Step 2 and build the “Blinking LED” project by clicking on the “Build the imported project” link
- The build ‘Console’ view will appear along the bottom with build messages as the project builds
- If the build is successful, a green checkmark will appear next to the link to indicate success



View: Console

- **Multiple contexts**

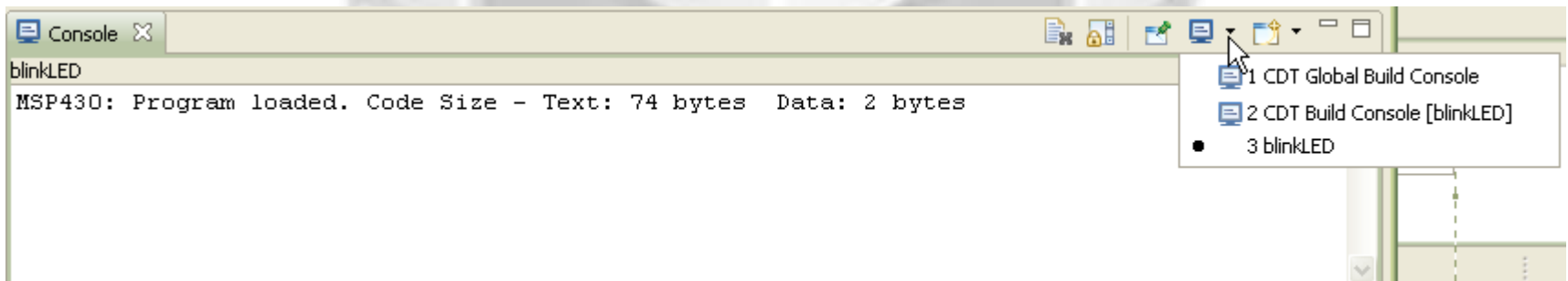
- Can display build messages or debug messages (including CIO) depending on which console is selected
- Automatically switches contexts when a new message occurs
 - Can use the “Pin” option to prevent this



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



Step 3: Specify 'Debugger Configuration'



- Note that the 'Connection' type is undefined (none) for Step 3
- Follow Step 3 and click on the "Debugger Configuration" link
- A dialog box will appear with a list of connections
 - Choose 'Texas Instruments XDS100v1 USB Emulator'
 - Select 'OK' when finished

The screenshot shows the TI Resource Explorer interface. On the left, a tree view lists various example projects under 'Kits'. On the right, a 'URL' pane displays instructions for Step 1, Step 2, and Step 3. Step 3, 'Debugger Configuration', is circled in red. Below the instructions, a 'Debugger Configuration' dialog box is open, showing a list of connections. The connection 'Texas Instruments XDS100v1 USB Emulator' is highlighted with a red circle. A yellow arrow points from the 'Debugger Configuration' link in the URL pane to the dialog box.

Step 1: [Import the example project into CCS](#)

Click on the link above to import the project. The imported project is available in the **Project Explorer** view, expand the project node to browse the imported source files. To modify source code, double clicks on the source file within the project to open the source file editor.

Step 2: [Build the imported project](#)

To change build options, right click on the project and select **Properties** from the context menu. To build the project, select the link above, or select the **Build** toolbar button, or select the **Project | Build Project** menu item.

Step 3: [Debugger Configuration](#)

Connection: **none**

Click on the link above to change the device connection. Additionally, this option is also available in the project properties.

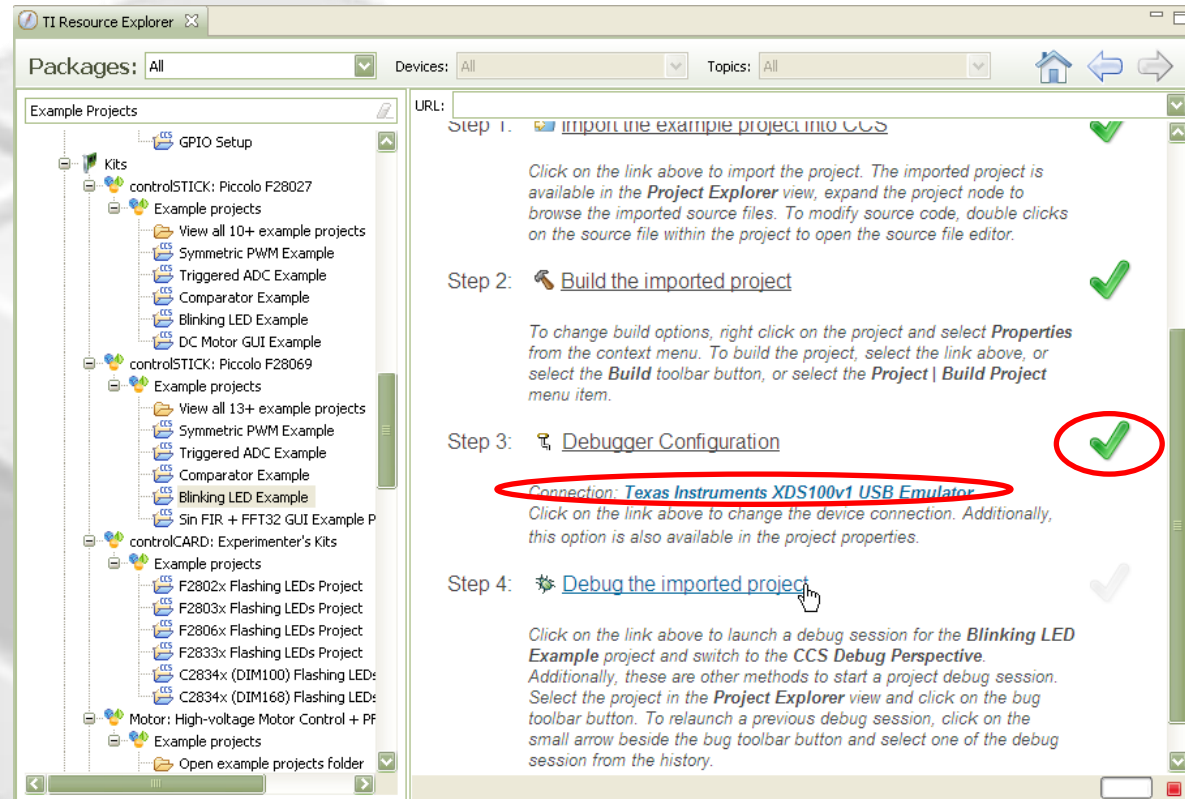
Debugger Configuration

Connections:

- Blackhawk PCI560 Emulator
- Blackhawk PCI560 Emulator, 20-pin JTAG Cable
- Blackhawk USB2000 Controller
- Blackhawk USB510 Emulator
- Blackhawk USB510L Emulator
- Blackhawk USB560 Emulator
- Blackhawk USB560-20-pin JTAG Cable
- Blackhawk USB560-BP Emulator
- Blackhawk USB560-M Emulator
- Blackhawk USB560-M Emulator, 20-pin JTAG Cable
- Blackhawk XDS560v2-LAN System Trace Emulator
- Blackhawk XDS560v2-USB Mezzanine Emulator
- Blackhawk XDS560v2-USB System Trace Emulator
- Data Snapshot Viewer
- Spectrum Digital C2000 XDS510LC Emulator
- Spectrum Digital XDS510 Parallel Port-PCI Emulator
- Spectrum Digital XDS510USB Emulator
- Spectrum Digital XDS510USB-PLUS Emulator via J5C
- Spectrum Digital XDS560V2 STM LAN Emulator
- Spectrum Digital XDS560V2 STM TRAVELER Emulator
- Spectrum Digital XDS560V2-STM USB Emulator
- Texas Instruments XDS100v1 USB Emulator**
- Texas Instruments XDS100v2 USB Emulator
- Texas Instruments XDS100v3 USB Emulator

Step 3: Specify 'Debugger Configuration'

- 'Connection' type field will be updated to show the chosen connection
- A green checkmark will appear next to the link to indicate the step is complete



Step 4: Debug 'Blinking LED' Project



- Follow Step 4 and debug the project by clicking on the “Debug the imported project” link
- Several actions are done automatically
 - Prompt to save source files
 - Build the project (incrementally)
 - Start the debugger (CCS will switch to the ‘CCS Debug’ perspective)
 - Connect CCS to the target
 - Load (flash) the program on the target
 - Run to ‘main’
- Don't want it to do all of the above at once? You can configure it to skip some steps (Debugger Options)

The screenshot shows the TI Resource Explorer window with the 'Blinking LED Example' project selected under the 'controlSTICK: Piccolo F28069' kit. The right pane displays a four-step guide to debug the project, with Step 4, 'Debug the imported project', circled in red. The steps are:

- Step 1: Import the example project into CCS** (Green checkmark)
Click on the link above to import the project. The imported project is available in the **Project Explorer** view, expand the project node to browse the imported source files. To modify source code, double clicks on the source file within the project to open the source file editor.
- Step 2: Build the imported project** (Green checkmark)
To change build options, right click on the project and select **Properties** from the context menu. To build the project, select the link above, or select the **Build** toolbar button, or select the **Project | Build Project** menu item.
- Step 3: Debugger Configuration** (Green checkmark)
Connection: **Texas Instruments XDS100v1 USB Emulator**
Click on the link above to change the device connection. Additionally, this option is also available in the project properties.
- Step 4: Debug the imported project** (Grey checkmark)
Click on the link above to launch a debug session for the **Blinking LED Example** project and switch to the **CCS Debug Perspective**. Additionally, these are other methods to start a project debug session. Select the project in the **Project Explorer** view and click on the bug toolbar button. To relaunch a previous debug session, click on the small arrow beside the bug toolbar button and select one of the debug session from the history.

Step 4: Debug 'Blinking LED' Project

CCS Debug - BlinkingLED/BlinkingLED-Main.c - Code Composer Studio

File Edit View Search Project Tools Run Scripts Window Help

Debug

BlinkingLED [Code Composer Studio - Device Debugging]

- Texas Instruments XDS100v1 USB Emulator/C28xx (Suspended)
- main() at BlinkingLED-Main.c:60 0x3F4181
- _args_main() 0x3F41BC (_args_main has only skeletal debug info)
- Texas Instruments XDS100v1 USB Emulator/CLA_0 (Disconnected : Unknown)

TI Resource Explorer

BlinkingLED-Main.c

```
50 //*****
51
52 // Used for running BackGround in flash and the ISR in RAM
53 extern Uint16 RamfuncsLoadStart, RamfuncsLoadEnd, RamfuncsRunStart;
54
55
56 //*****
57 // MAIN CODE - start
58 //*****
59 void main(void)
60 {
61
62 //=====
63 //  INITIALISATION - General
64 //=====
65
66     DeviceInit(); // Device Life support & GPIO mux settings
67
68 // Only used if running from FLASH
69 // Note that the variable FLASH is defined by the compiler (-d FLASH)
70 #ifdef FLASH
71 // Copy time critical code and Flash setup code to RAM
72 // The RamfuncsLoadStart, RamfuncsLoadEnd, and RamfuncsRunStart
```

Switched to 'CCS Debug' perspective

Program counter at 'main'

Eclipse Concept: Perspectives

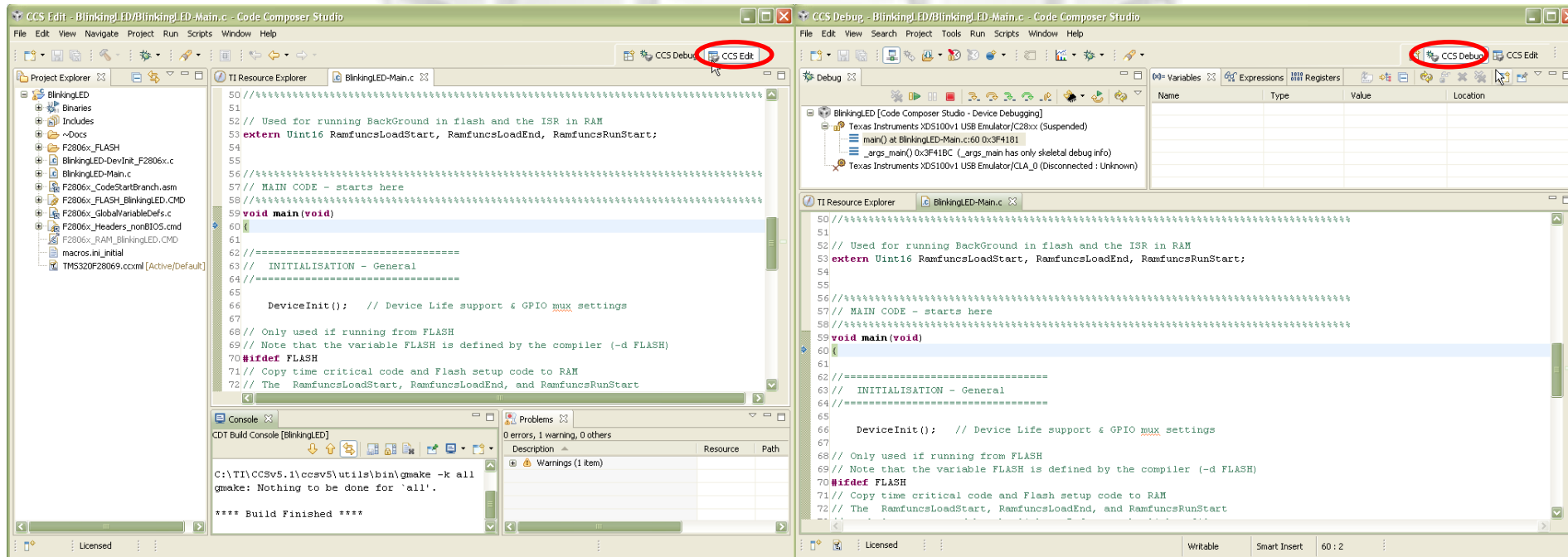


- Defines the initial set and layout of views in the Workbench window



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)

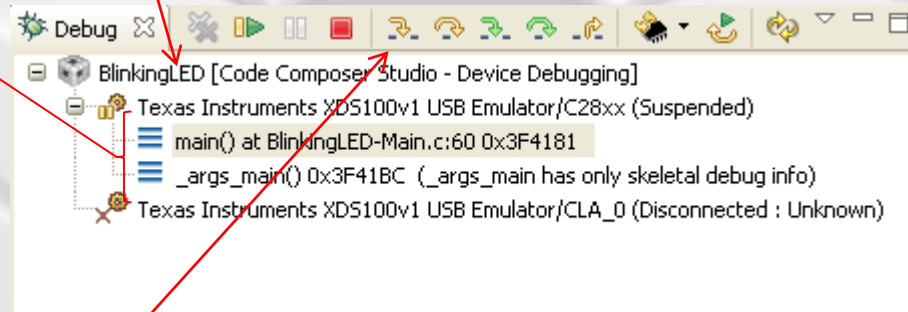
- Each perspective provides a set of functionality aimed at accomplishing a specific type of task ('CCS Edit' for project development, 'CCS Debug' for debugging, etc)
- Can create custom perspectives



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**

Blink LD2




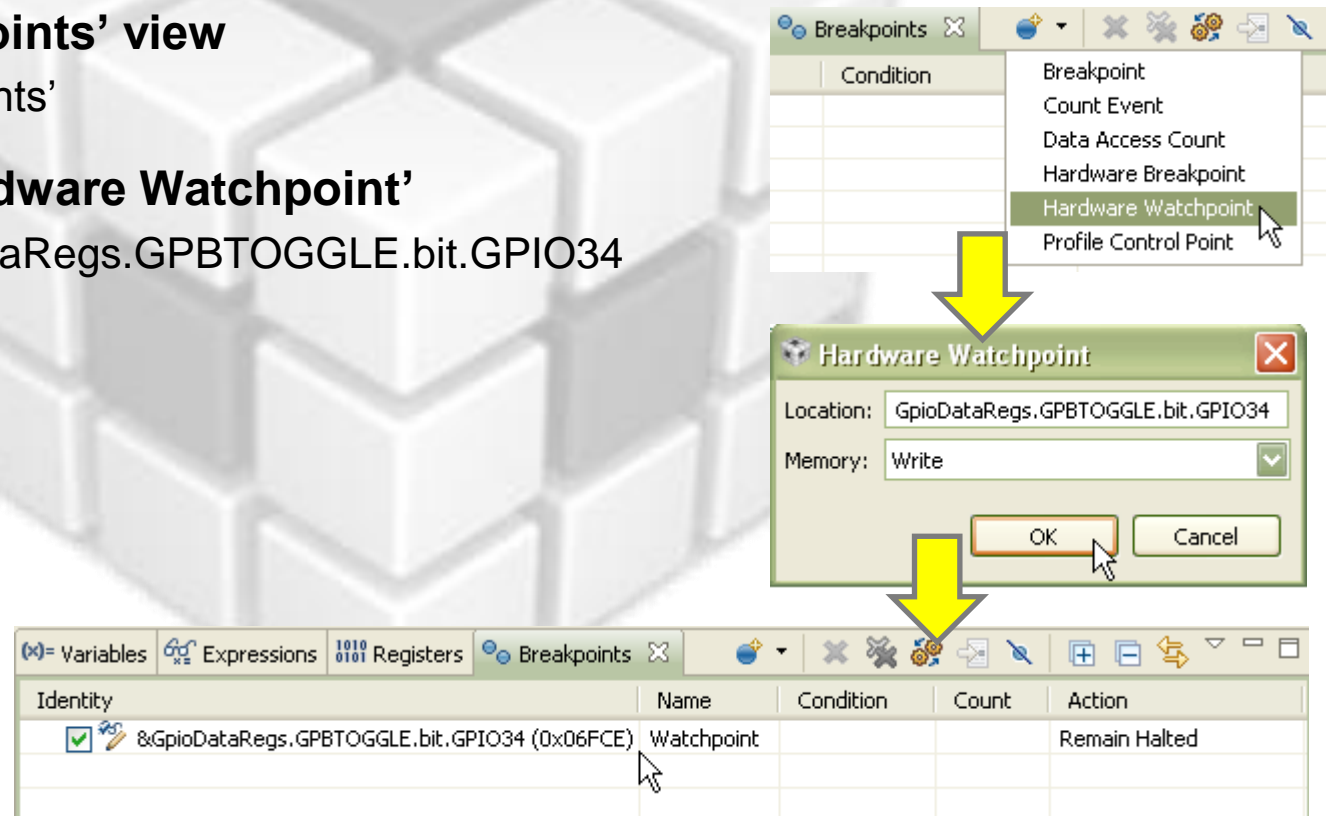
- Press the 'run' button  to run the program
 - LD2 on the controlSTICK should now be blinking



Debugging: Using Watchpoints



- Press the halt/suspend button  to halt the running program
 - The code should stop somewhere inside the 'for' loop
- Open the 'Breakpoints' view
 - 'View -> Breakpoints'
- Create a new 'Hardware Watchpoint'
 - Location: GpioDataRegs.GPBTOGGLE.bit.GPIO34
 - Memory: Write



Debugging: Using Watchpoints





- Run the target again. Execution will automatically halt when LED LD2 is toggled (GpioDataRegs.GPBTOGGLE.bit.GPIO34 is written to)
- Notice that the program counter is currently halted at some time after the write occur (and not at the exact moment). Why is that?

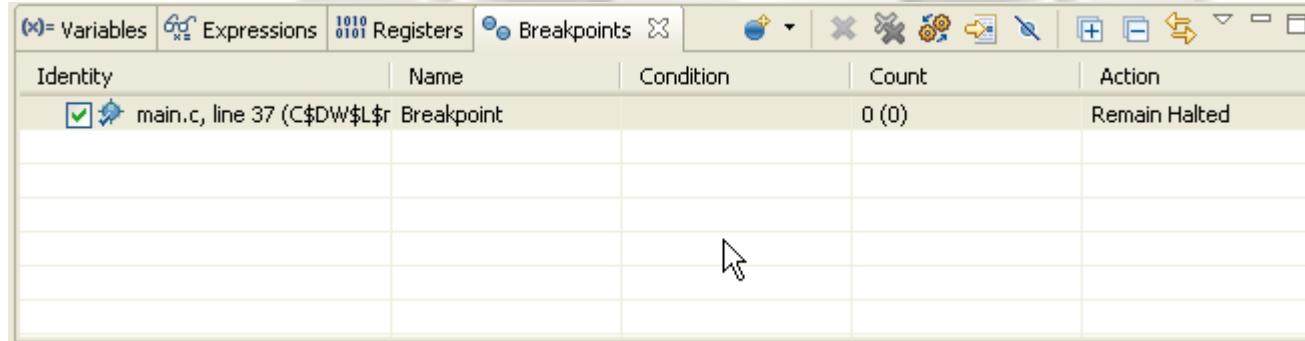
The screenshot shows the CCS Debug interface for the BlinkingLED project. The left pane displays the project tree with the main function selected. The right pane shows the watchpoints table with one entry: &GpioDataRegs.GPBTOGGLE.bit.GPIO34 (0x06FCE) set as a Watchpoint with the action 'Remain Halted'. The bottom pane shows the source code for BlinkingLED-Main.c, with line 105 highlighted: `if(CpuTimer0Regs.TCR.bit.TIF == 1)`.


Identity	Name	Condition	Count	Action
<input checked="" type="checkbox"/> &GpioDataRegs.GPBTOGGLE.bit.GPIO34 (0x06FCE)	Watchpoint			Remain Halted

```
97 // Enable Global Interrupts
98
99 //=====
100 // Forever LOOP
101 //=====
102
103 for(;;) //infinite loop
104 {
105     if(CpuTimer0Regs.TCR.bit.TIF == 1)
106     {
107         CpuTimer0Regs.TCR.bit.TIF = 1; // clear flag
108
109         //-----
110         GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; //Toggle GPIO34 (LD2)
111         //-----
112     }
```


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)



Identity	Name	Condition	Count	Action
<input checked="" type="checkbox"/> 	main.c, line 37 (C\$DW\$L\$r Breakpoint		0 (0)	Remain Halted

More Debugging



- **Investigate other debugging views (Open via 'View' menu)**

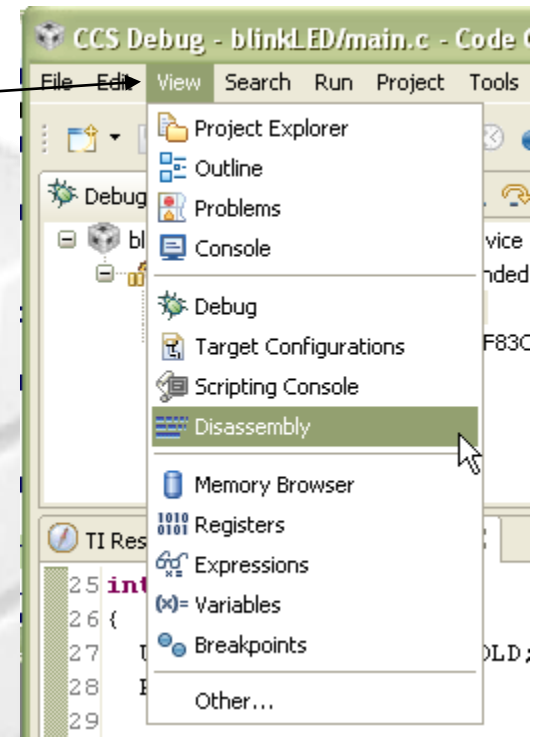
- Memory Browser
- Registers
- Disassembly (see next slide)

- **Set breakpoints**

- Double click on a source line to set/clear
- See list of breakpoints with the 'Breakpoints' view
- Note that the 28x only support two hardware breakpoints
 - A watchpoint uses a hardware breakpoint resource

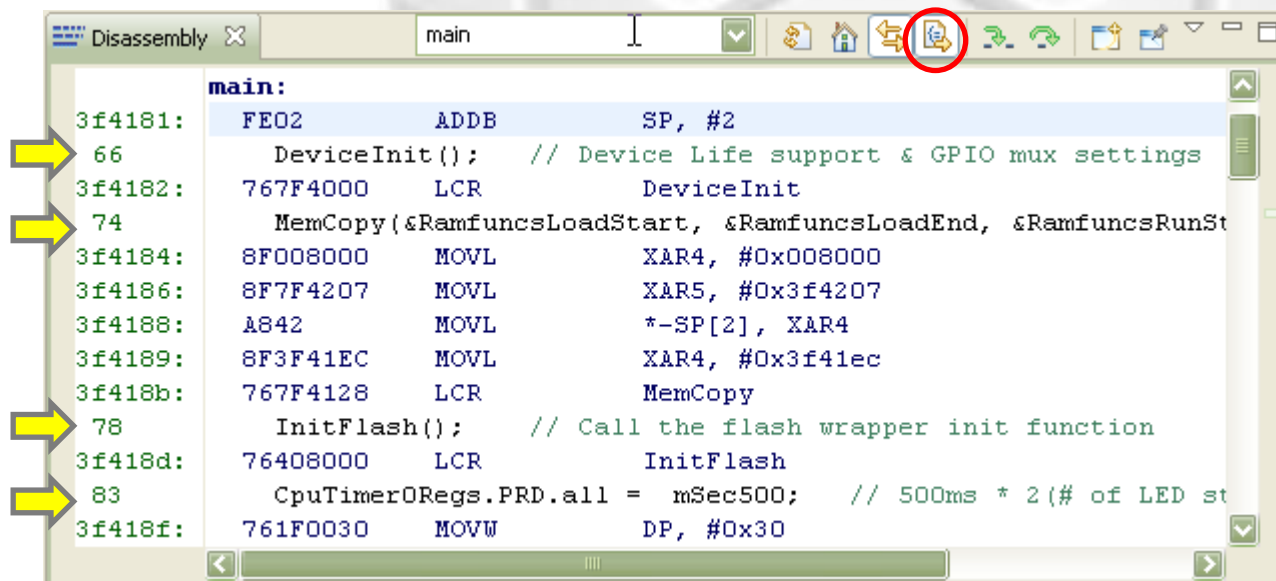
- **Use the buttons in the 'Debug' view to:**

- Restart the program
- Source stepping
- Assembly stepping



View: Disassembly

- Go to the 'main' symbol in the Disassembly view by typing it in the address field and hit 'ENTER'
- Toggle the 'Show Source' button
- Note the toggling of interleaved source with the disassembly



```

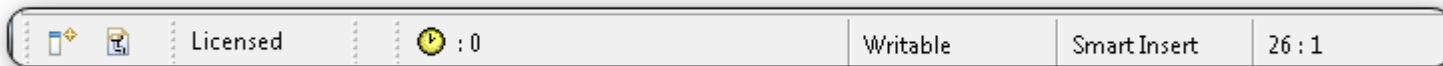
main:
3f4181: FE02      ADDB      SP, #2
66      DeviceInit(); // Device Life support & GPIO mux settings
3f4182: 767F4000   LCR       DeviceInit
74      MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunSt
3f4184: 8F008000   MOVL      XAR4, #0x008000
3f4186: 8F7F4207   MOVL      XAR5, #0x3f4207
3f4188: A842      MOVL      *-SP[2], XAR4
3f4189: 8F3F41EC   MOVL      XAR4, #0x3f41ec
3f418b: 767F4128   LCR       MemCopy
78      InitFlash(); // Call the flash wrapper init function
3f418d: 76408000   LCR       InitFlash
83      CpuTimer0Regs.PRD.all = mSec500; // 500ms * 2(# of LED st
3f418f: 761F0030   MOVW      DP, #0x30
  
```


Debugger Options

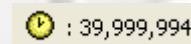
- Many debugging features can be enabled/disabled from the Debugger Options
- During a debug session in the 'CCS Debug' perspective:
 - 'Tools->Debugger Options->Generic Debugger Options'
- Configure a variety of debug options like enable/disable:
 - auto-run to main
 - auto-connect to a HW target
 - real-time options (on supported HW like C2000)
 - program verification on load
 - etc...
- Use the "Remember My Settings" option to have the settings apply for subsequent debug sessions

Counting Cycles (Profile Clock)

- **The profile clock**
 - Available on most devices and can be used to count cycles
 - On some targets it can be used to count other events as well
- **Enable the Clock**
 - Run -> Clock -> Enable
 - The clock will now be displayed on the status bar



- **Check that the watchpoint that is watching for writes to 'GpioDataRegs.GPBTGGLE.bit.GPIO34' is enabled**
- **Click the 'run' button**
 - Clock should now show ~40M cycles
- **TIP: Double-clicking on the clock icon will reset the count to '0'**

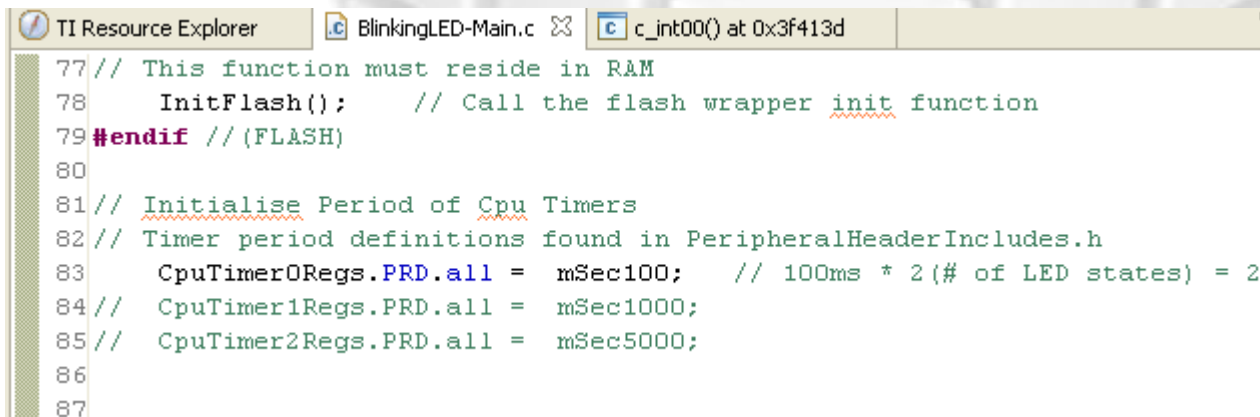


A screenshot of the status bar showing the clock icon followed by the text ': 39,999,994'.

Increase LD2 Blink Rate



- The blink rate of LED L2 can be increased by changing the value of the timer
- **Modify line 83 of BlinkingLED-Main.c**
 - From: `CpuTimer0Regs.PRD.all = mSec500;` // 500ms * 2(# of LED states) = 1s blink rate
 - To: `CpuTimer0Regs.PRD.all = mSec100;` // 100ms * 2(# of LED states) = 200ms blink rate
- **Save file**

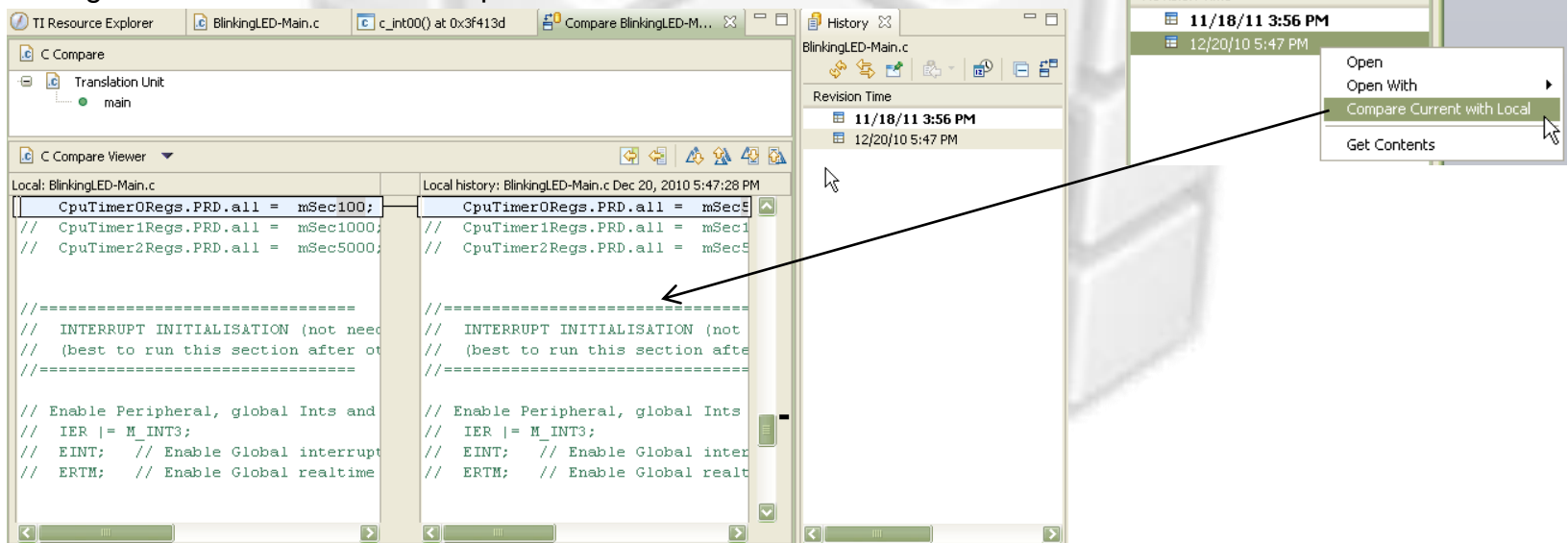
A screenshot of the TI Resource Explorer window showing the file BlinkingLED-Main.c. The code is displayed in a text editor with line numbers 77 to 87. Line 83 is highlighted, showing the change from mSec500 to mSec100. The background of the slide features a large, faint image of a keyboard.

```
77// This function must reside in RAM
78    InitFlash();    // Call the flash wrapper init function
79#endif // (FLASH)
80
81// Initialise Period of Cpu Timers
82// Timer period definitions found in PeripheralHeaderIncludes.h
83    CpuTimer0Regs.PRD.all = mSec100;    // 100ms * 2(# of LED states) = 2
84//    CpuTimer1Regs.PRD.all = mSec1000;
85//    CpuTimer2Regs.PRD.all = mSec5000;
86
87
```


View: Local History



- **CCS keeps a local history of source changes**
 - Switch to the 'CCS Edit' perspective
 - 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**
 - Double-click on a revision to open it in the editor
 - Right-click on a revision to compare that revision to the current version

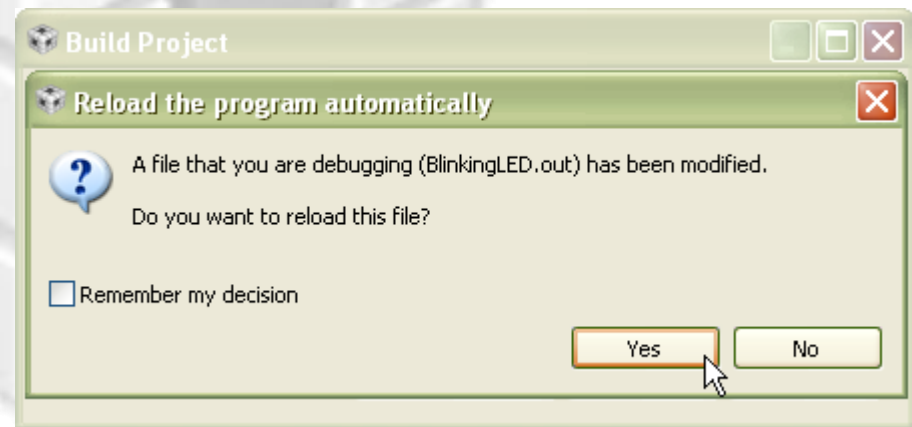
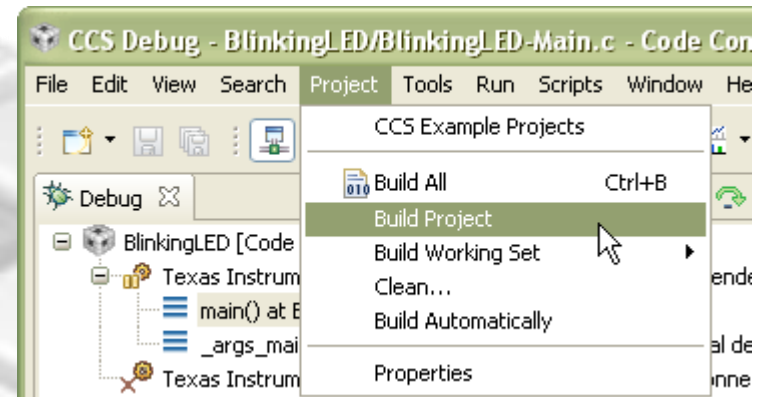


- **CCS also keeps project history**
 - Recover files deleted from the project
 - Right-click on the project and select "Recover from Local History" in the context menu

Rebuild Project



- Switch to the 'CCS Debug' perspective
- Rebuild the project
 - Project -> Build Project
- CCS will automatically detect that the currently program being debugged has changed/rebuilt and ask if it should reload the file
 - Select 'Yes' and CCS will reload/reflash the program and run to 'main'



Blink LD2



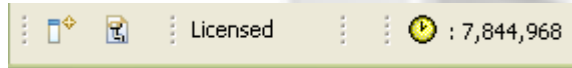
- Press the 'run' button  to run the program
 - LD2 on the controlSTICK should now be blinking at a much quicker rate



Counting Cycles (Profile Clock) – Part 2

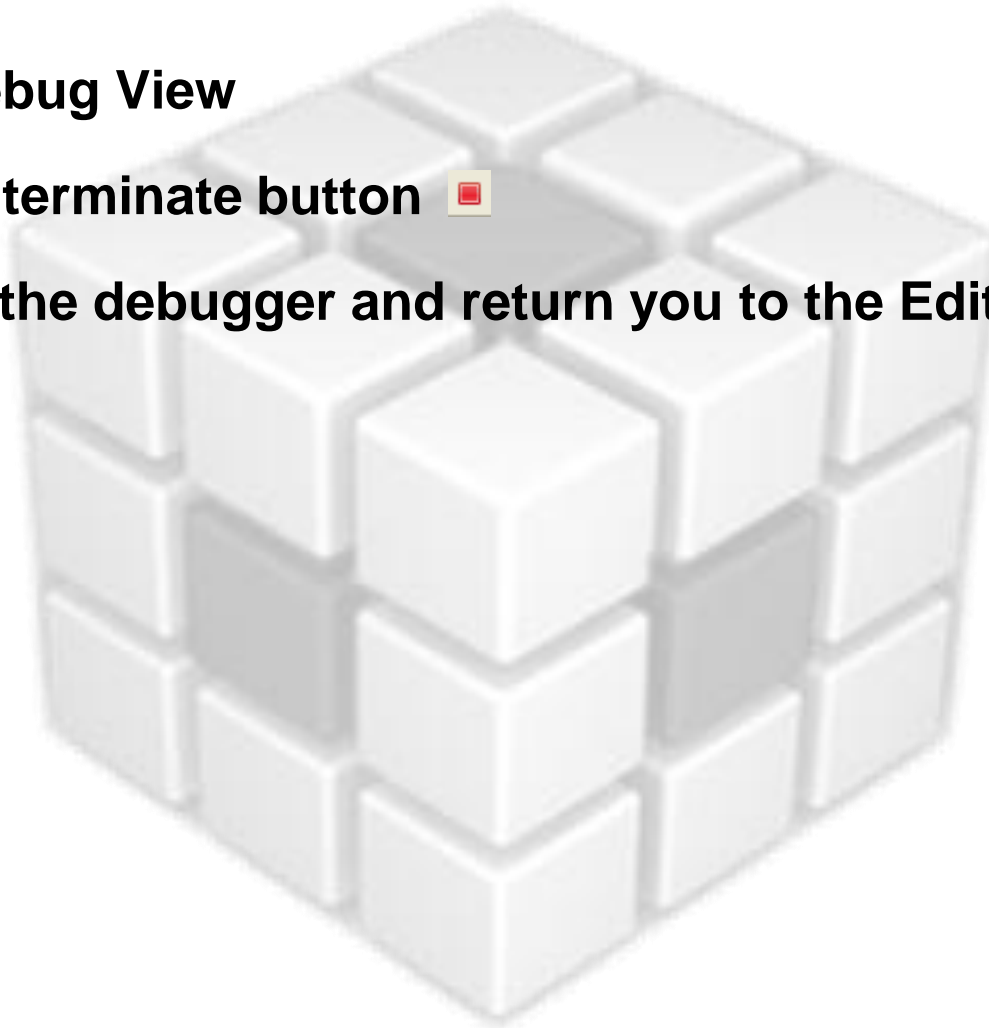


- **Pause execution of the program**
- **Double-click on the clock icon in the status bar to reset the value to '0'**
- **Enable the watchpoint for 'GpioDataRegs.GPBTOGGLE.bit.GPIO34'**
 - Watchpoint got reset when the program was reloaded
- **Click the 'run' button**
 - Clock should now show ~7.85M cycles (1/5 of 40M)



Terminate the Debug Session

- Go to the Debug View
- Click on the terminate button 
- This will kill the debugger and return you to the Edit perspective



Loading Symbols for Flashed Program



- If the program is already flashed in CCS and just wish to debug the existing code flashed on the target, you can configure CCS to debug the project by loading symbols only
- Select the drop-down menu next to the 'bug' button and select 'Debug Configurations..'



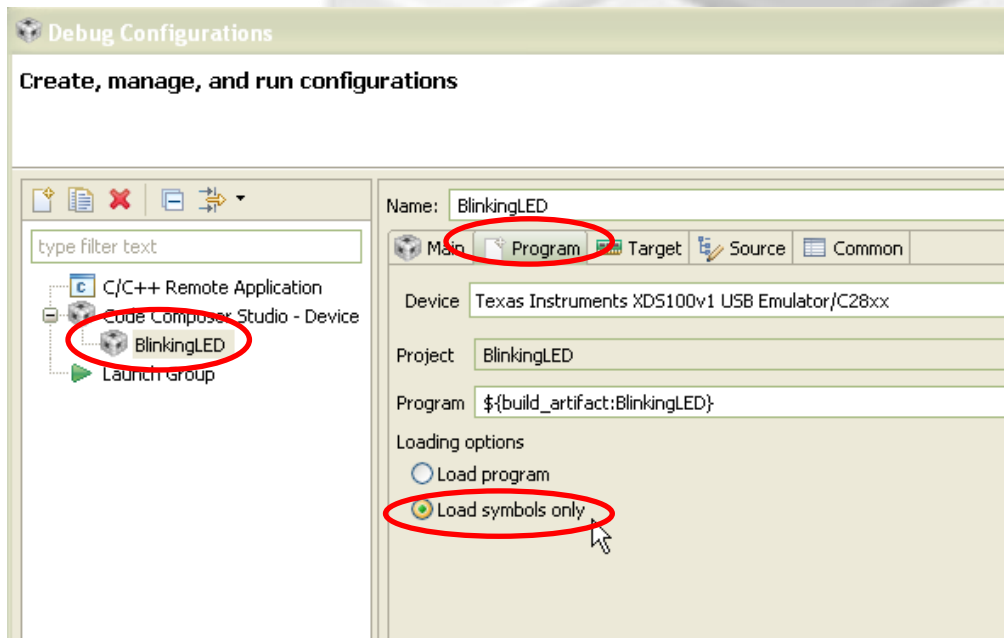
Eclipse Concept: Debug Configurations - Cached information created when a debug session is first launched for a project or target configuration. Information cached includes which target configuration to use, debug settings...



Loading Symbols for Flashed Program



- Select 'BlinkingLED' in the left panel and the 'Program' tab in the right panel
- Under 'Loading options', select 'Load symbols only'

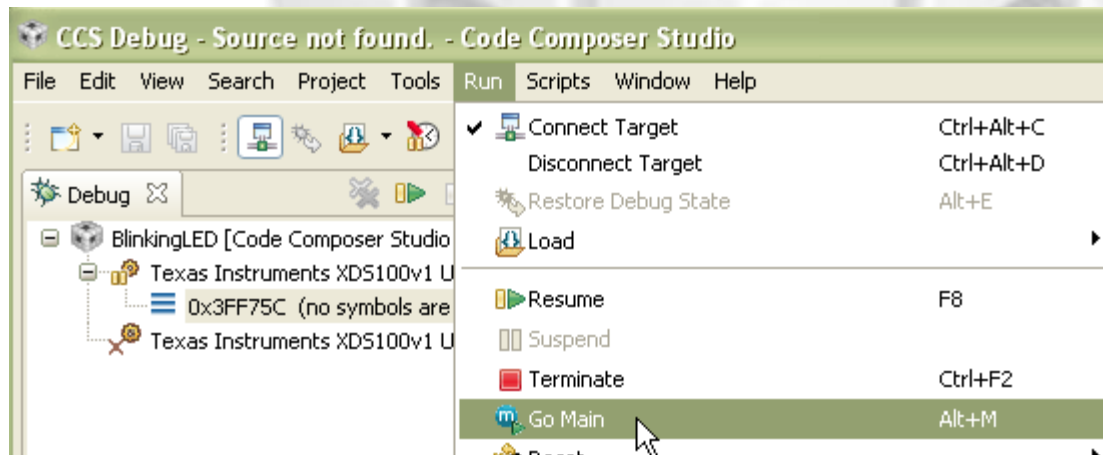


- Then select 'Apply' and then 'Debug'

Loading Symbols for Flash Program



- The debugger will start up, connect to the target, and load only the symbols for the program for the debugger (no code is loaded/flushed on the target)
- The program counter will be set to the entry point of the code and not at 'main'
 - 'Run->Go Main' will run the target to 'main'



Loading Symbols for Flash Program

The screenshot displays the CCS Debug interface for a project named "BlinkingLED". The interface is divided into several panes:

- Callstack:** Located at the top left, it shows the current callstack with "main() at BlinkingLED-Main.c:60 0x3f4181" selected. A blue callout box points to it with the text "Callstack displayed".
- Source Code:** The main pane on the left shows the C source code for "BlinkingLED-Main.c". The function "main()" is visible, starting with "void main(void)". A blue callout box points to the code with the text "Source code is found automatically".
- Disassembly:** The pane on the right shows the disassembly of the code, starting with "main:" and "FE02 ADDDB SP, #2".
- Console:** The bottom pane shows the console output, which includes the command "C:\TI\CCSV5.1\ccsv5\utils\bin\gmake -k all" and the response "gmake: Nothing to be done for 'all'." followed by "**** Build Finished ****".

At the bottom right, there is a vertical text label "CCS APPS".

Changing Build Options

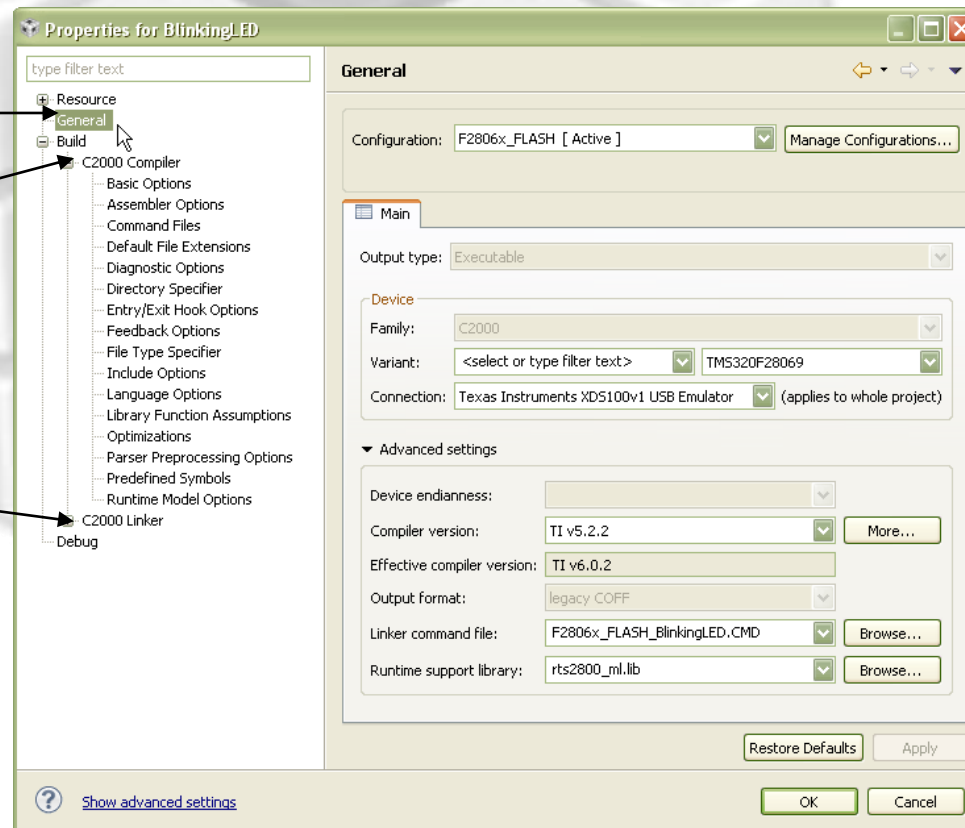


- Terminate the debug session
- Right click on the 'BlinkingLED' project and select 'Properties'

Device and high level settings

Compiler Options

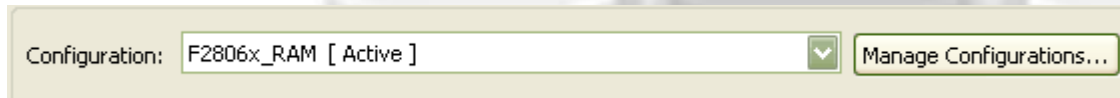
Linker Options



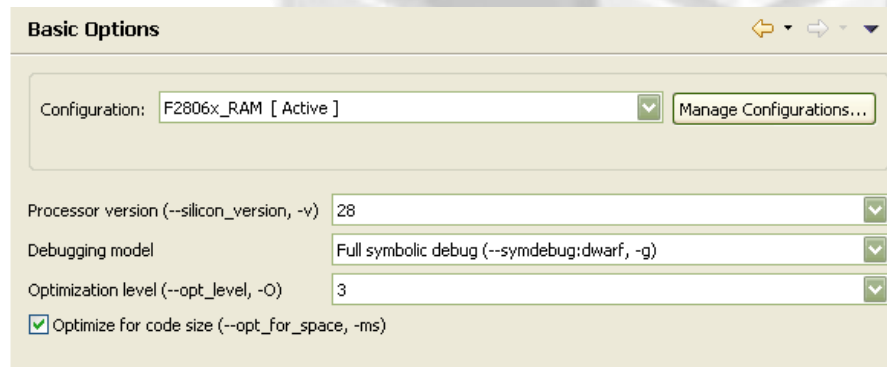
Changing Build Options



- Build options are set per build configuration
- Change your Configuration to “F2806x_RAM”



- Change the optimization settings
 - Go to the Basic Option Group
 - Change the optimization level to 3
 - Enable ‘Optimize for code size’



- Click OK

Changing Build Options



- **Change the active configuration to 'F2806x_RAM'**
 - Right click on the Project
 - Select Build Configurations -> Set Active -> F2806x_RAM
- **Build the project by clicking the build button**
 - In the console view you will see that the 'F2806x_RAM' configuration has been built
- **You can also change the configuration and build it by click on the arrow beside the build button and selecting the configuration you want to build**
 - Select 'F2806x_RAM' and it will build the F2806x_RAM configuration
 - The active configuration is indicated by the Checkmark

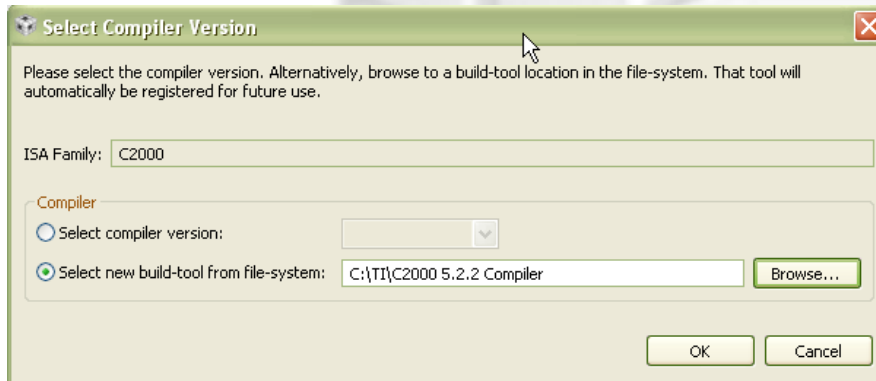


Compiler Versions

- **Each project specifies which version of the compiler to use**
 - Actually set on a per configuration level
 - I.e. Debug can use one version and release another
- **When you download a new compiler if you want to use it you have to change the compiler version specified by your project**
- **CCS will allow you to select from all the compilers that it knows about on your computer**
- **When you install a new compiler via the Update Manager in CCS it will automatically know where the compiler is**
 - However you can also tell CCS where another version of the compiler is located

Changing the Compiler Version

- Right click on your project and select Properties
- Click on General
- Click on the More button beside the Compiler version “TI 5.2.2”
- Browse to the path shown in the dialog and click OK



- CCS will determine what compiler is located there and select it for your active configuration. You will see that TI v5.2.2 is now specified
- Build your project



TARGET CONFIGURATIONS

Target Configuration Files - Basics

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



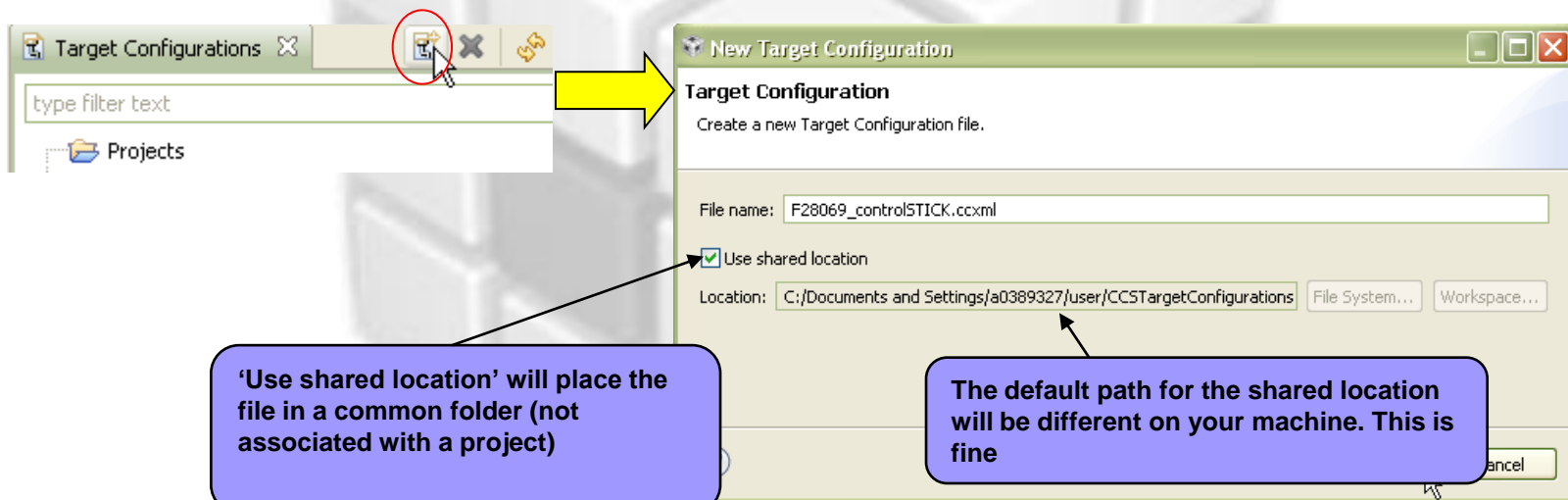
Equivalent of the CCS 3.x configuration file (*.ccs extension)

- **‘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**

Creating a Target Configuration



- Open the 'Target Configuration' view
 - View -> Target Configurations
- Create a new configuration:



Creating a Target Configuration



- In the 'Basic' tab, select the options for:
 - Connection: Texas Instruments XDS100v1 USB Emulator
 - Device: controlSTICK – Piccolo F28069

The screenshot shows the 'Basic' tab of the Target Configuration tool. The 'General Setup' section includes a 'Connection' dropdown menu set to 'Texas Instruments XDS100v1 USB Emulator' and a 'Board or Device' list with 'controlSTICK - Piccolo F28069' selected. The 'Advanced Setup' section includes a 'Save Configuration' button and a 'Test Connection' button. A note at the bottom states: 'Note: Support for more devices may be available from the update manager.'

General Setup
This section describes the general configuration about the target.

Connection: Texas Instruments XDS100v1 USB Emulator

Board or Device: 28069

- ☐ Experimenters Kit - Piccolo F28069
- ☒ controlSTICK - Piccolo F28069
- ☐ TM5320F28069

Piccolo F28069 controlSTICK

Advanced Setup

[Target Configuration](#): lists the configuration options for the target.

Save Configuration

Save

Test Connection

To test a connection, all changes must have been saved, the configuration file contains no errors and the connection type supports this function.

Test Connection

Press 'Save' to save the target configuration file

Validate the target configuration file with the 'Test Connection' button

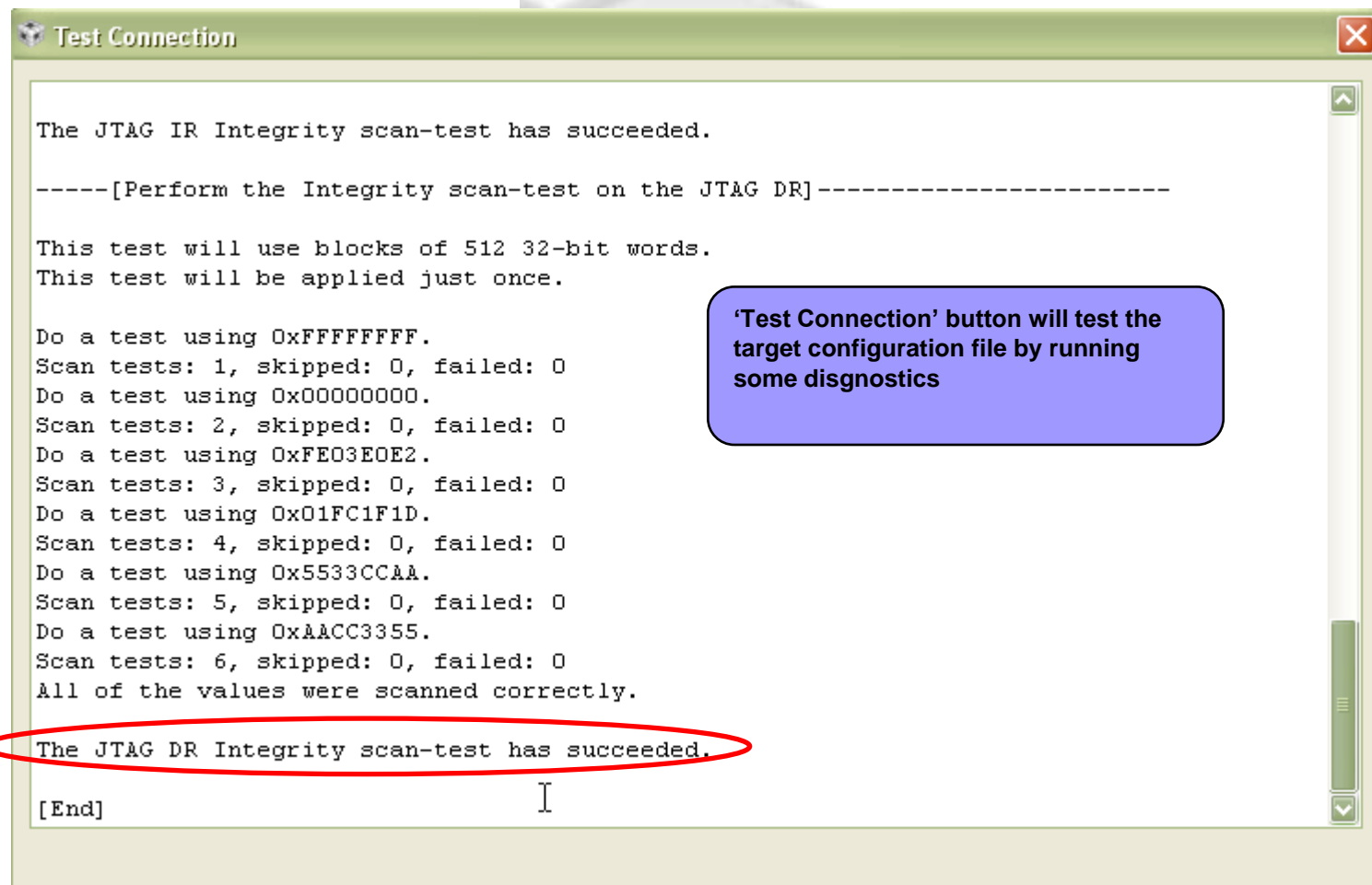
Basic Advanced Source

Target Configurations - Advanced

Test Connection

To test a connection, all changes must have been saved, the configuration file contains no errors and the connection type supports this function.

Test Connection



Target Configurations - Advanced

- **Use 'Advanced Setup' utility if option for your emulator/target is missing from the 'Basic' tab**
 - Select from a list of available 'Connections' to specify the connection type
 - Then select from the list of components ('Devices', 'CPUs', 'Routers') to add to the connection
- **Use the 'Advanced Setup' to create a single target configuration using two emulators**
- **Must have good knowledge of the device to correctly use 'Advanced Setup' utility to build a configuration**

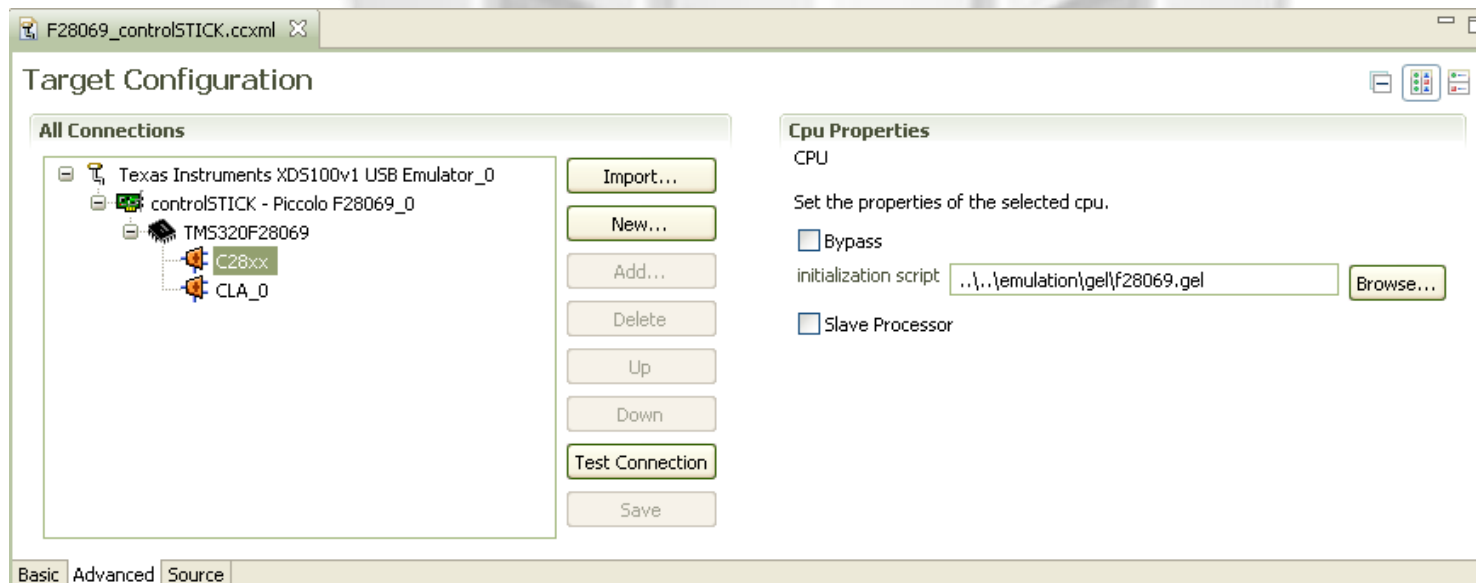
Target Configurations - Advanced

- **Adjust default properties of the target configuration:**

- Specify initialization files (GEL startup files)
- Specify IcePick subpath port numbers
- Bypass CPU
- Set JTAG TCLK Frequency
- etc...

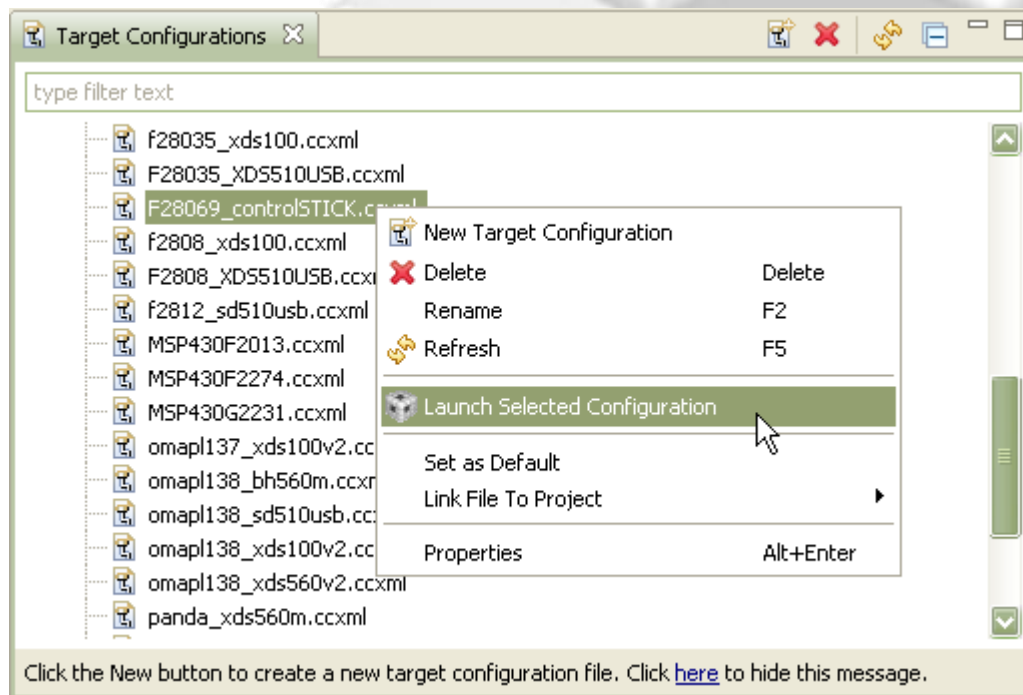


Similar to the CCS 3.x CCS Setup utility



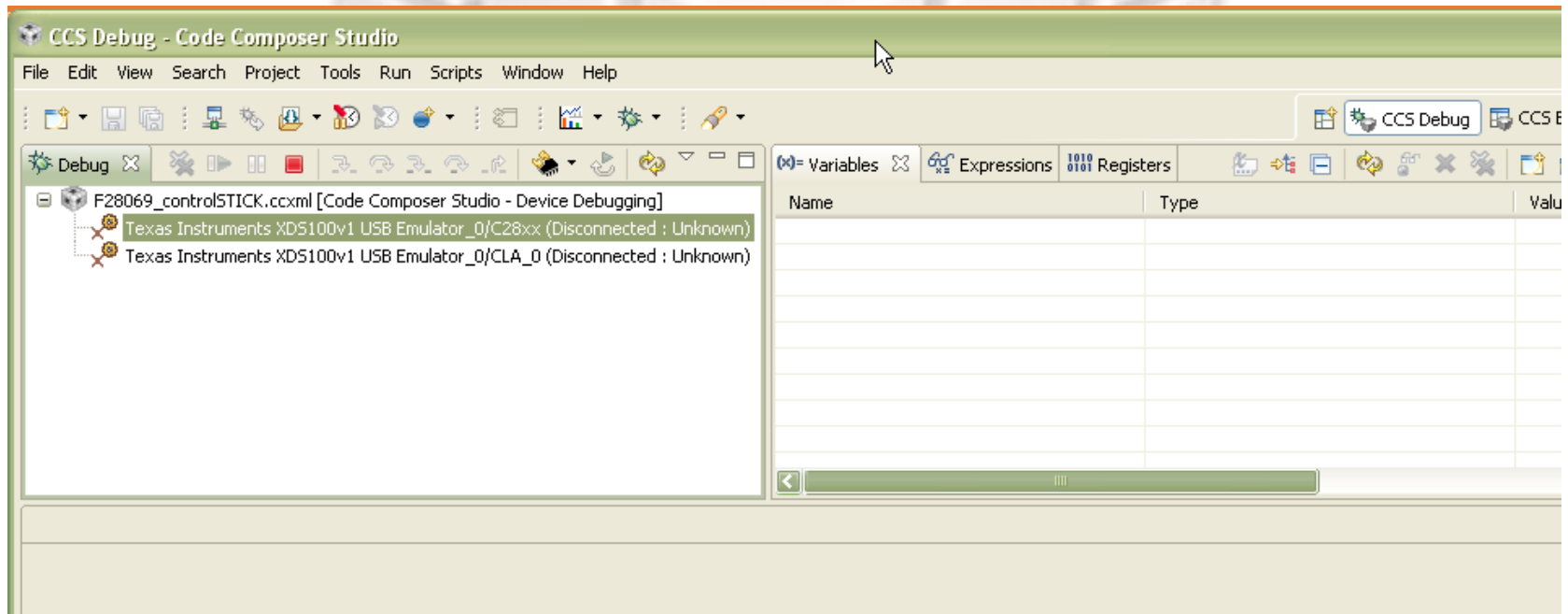
Launch a Debug Session

- Right-click on the newly created target configuration file and select 'Launch Selected Configuration' to start a debug session



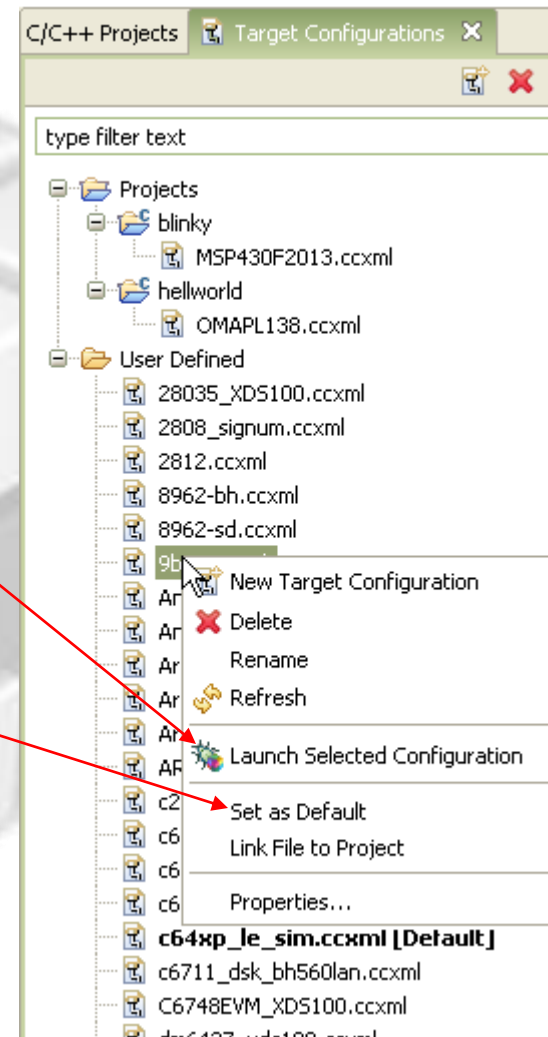
Project-less Debug Session

- This debug session is sometimes called a “project-less debug session” because there is no project association
- Use project-less debug sessions to debug executables without a CCS project (ex. Projects built outside CCS via standard make)
 - Debug programs already flashed by just loading symbols (‘Run -> Load -> Load Symbols’)



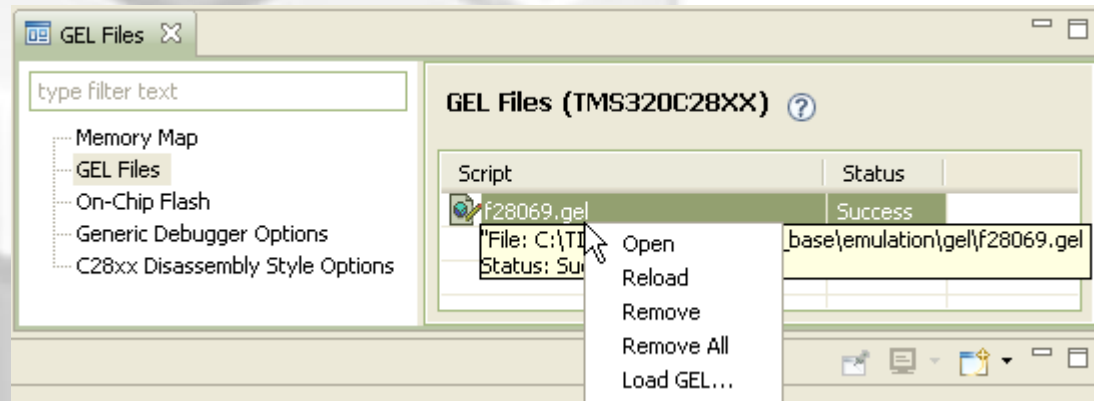
View: Target Configurations

- 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



View: GEL Files

- Displays all loaded GEL files for the current debug session and the status of each
 - NOTE: This view is only available when a debug session is active
- Actions available:
 - Open file in the editor
 - Reload the file
 - Remove/unload the file
 - Remove/unload all files
 - Load additional GEL files
- Newly loaded GEL files appear in the view
- The status will indicate if the load was successful





CPU TIMER EXAMPLE

CPU Timer Example: Exercise Summary

- **Key Objectives**

- Create a new portable project based on the CPU Timer example
- Build the program
- Start a debug session and load the program on the controlSTICK
- Run the program
- Enable real-time mode to observe variables updating in real-time

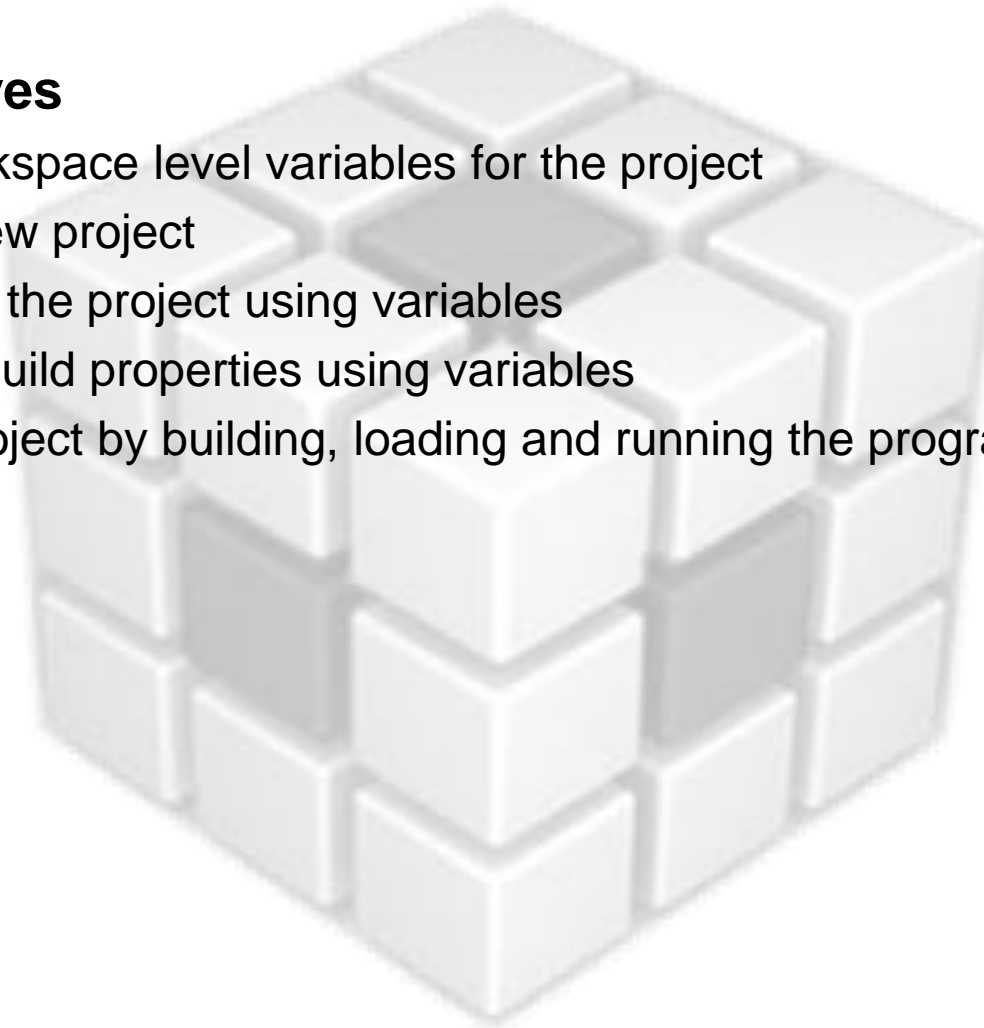
- **Tools and Concepts Covered**

- Portable Projects
- Linked resources
- Linked resource path variables
- Build variables
- Real-time Mode

Create a New Portable Project: Briefing

- **Key Objectives**

- Create workspace level variables for the project
- Create a new project
- Link files to the project using variables
- Configure build properties using variables
- Validate project by building, loading and running the program





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 – “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

CPU Timer: Create a New Project



- **Launch 'New CCS Project' Wizard**
 - Select 'New Project' from the Welcome page
- **Fill in the fields as shown in the right**
- **Select 'Finish' when done**
- **Generated project will appear in the Project Explorer view**
- **Remove the generated 'main.c' file from the project**

New CCS Project

CCS Project
Create a new CCS Project.

Project name: **CPUTimer**

Output type: Executable

☒ Use default location
Location: C:\TI\workspaces\51GREE\CPUTimer

Device

Family: **C2000**

Variant: 2806x Piccolo **controlSTICK - Piccolo F28069**

Connection: **Texas Instruments XDS100v1 USB Emulator**

Advanced settings

Project templates and examples

type filter text

- Empty Projects
 - Empty Project
 - Empty Assembly-only Project
 - Empty RTSC Project
- Basic Examples
 - Hello World
- DSP/BIOS v5.xx Examples
- IPC and I/O Examples
- SVS/RTOS

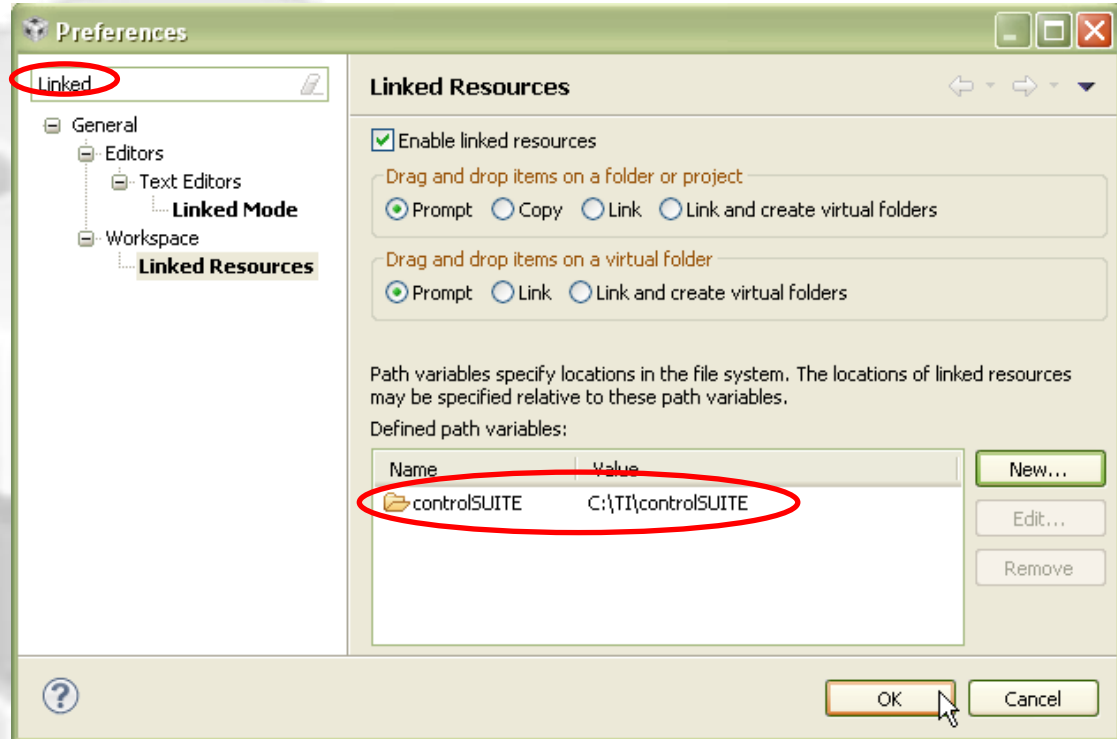
Creates an empty project fully initialized for the selected device.

Finish

CPU Timer: Create a Linked Resource Path Variable



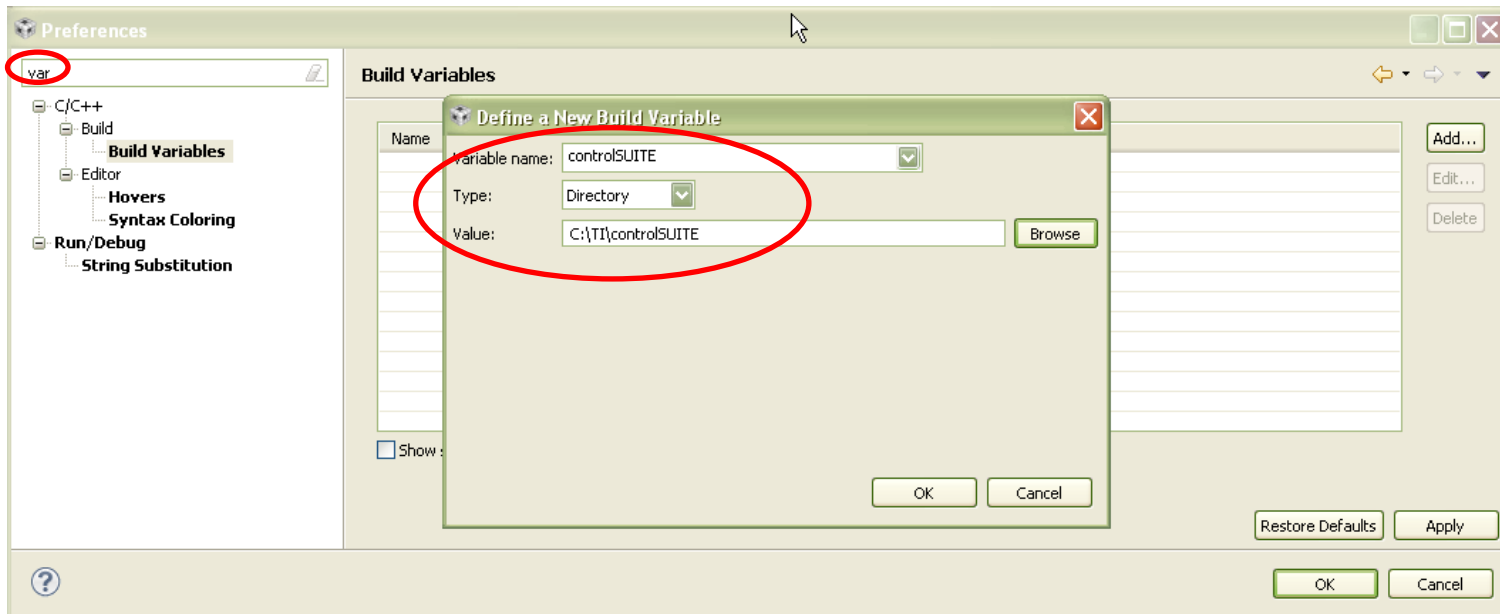
- Open the workspace preferences
 - Window -> Preferences
- Go to the 'Linked Resources' preferences
 - Type 'Linked' in the filter field to make it easier to find
- Use the 'New' button to create a 'Linked Resource Variable' that points to the root location of the controlSUITE directory
- Hit 'OK' when finished



CPU Timer: Create a Build Variable



- 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' that points to the root location of the controlSUITE directory
- Hit 'OK' when done



CPU Timer: Link Source Files to Project

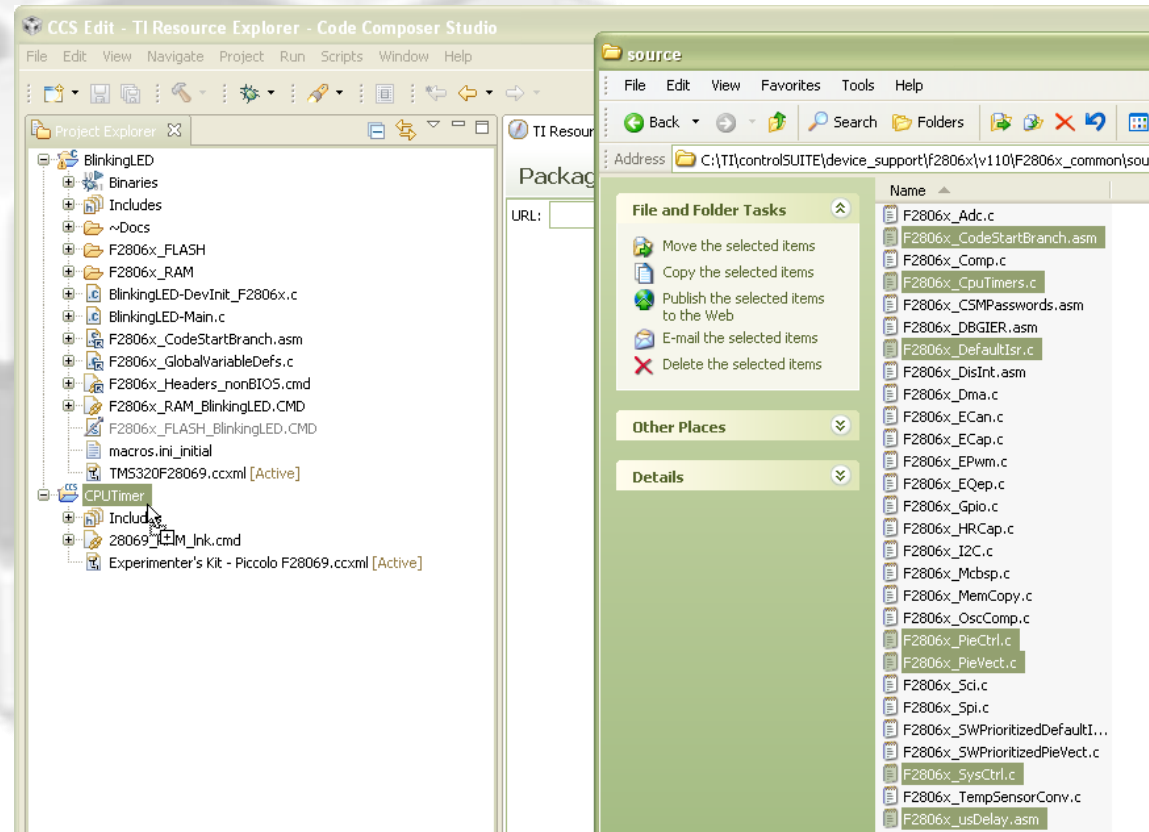


- **Open Windows Explorer and browse to:**

- C:\TI\controlSUITE\device_support\f2806x\v110\F2806x_common\source

- Drag and drop the following source files into the new project in the CCS Project Explorer view

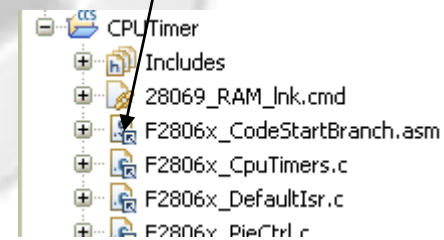
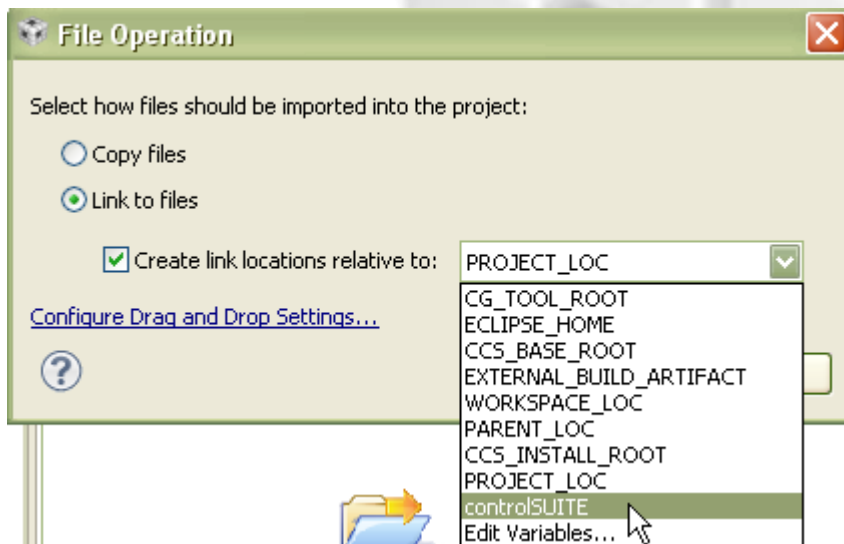
- F2806x_CodeStartBranch.asm
 - F2806x_CpuTimers.c
 - F2806x_DefaultIsr.c
 - F2806x_PieCtrl.c
 - F2806x_PieVect.c
 - F2806x_SysCtrl.c
 - F2806x_usDelay.asm



CPU Timer: Link Source Files to Project



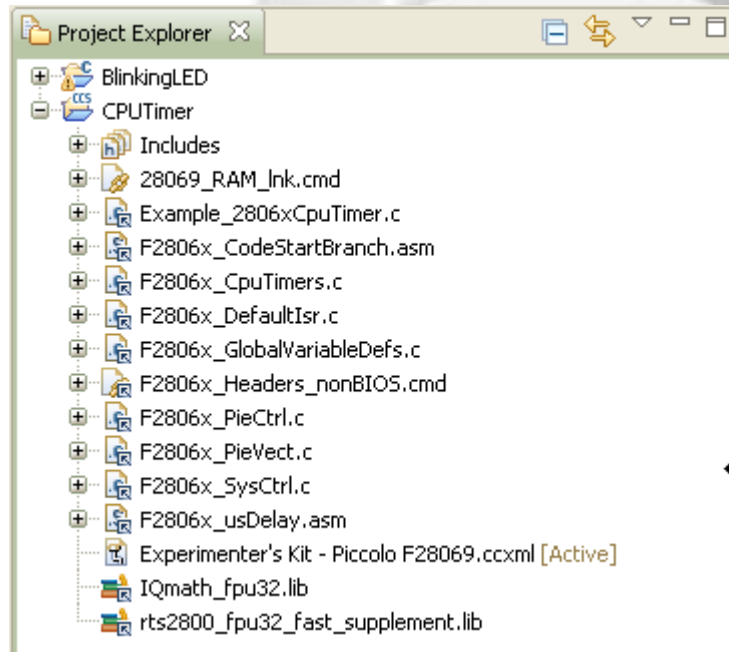
- **A dialog will appear asking if you wish to Copy or Link the files:**
 - Select 'Link to files'
 - Select 'Create link locations relative to:'
 - Use the new Linked Resource variable we created (controlSUITE)
 - Hit 'OK'
- **Files will now appear in the Project Explorer with the 'link' icon**



CPU Timer: Link Files to Project



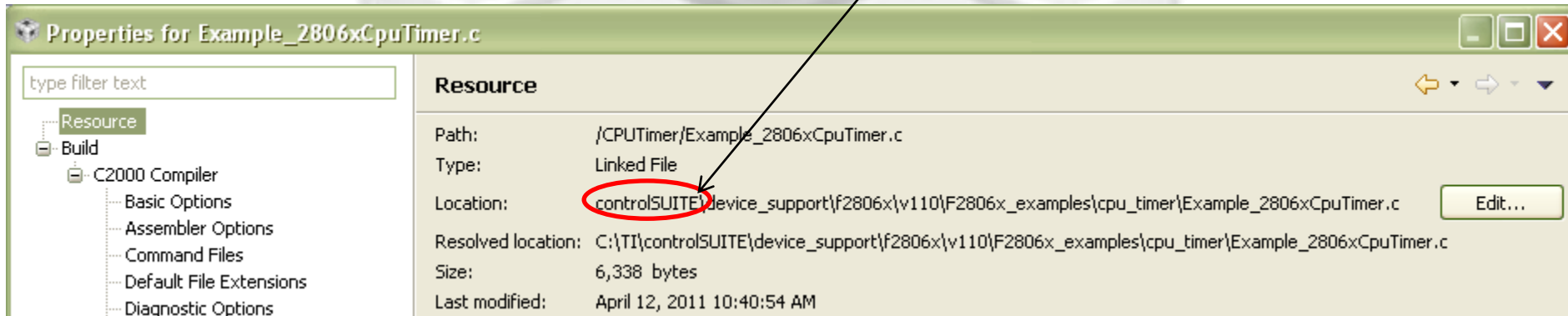
- **Link the rest of the needed files the same way:**
 - C:\TI\controlSUITE\device_support\f2806x\v110\F2806x_examples\cpu_timer\Example_2806xCpuTimer.c
 - C:\TI\controlSUITE\device_support\f2806x\v110\F2806x_headers\source\F2806x_GlobalVariableDefs.c
 - C:\TI\controlSUITE\device_support\f2806x\v110\F2806x_headers\cmd\F2806x_Headers_nonBIOS.cmd
 - C:\TI\controlSUITE\libs\math\IQmath\v15c\lib\rts2800_fpu32_fast_supplement.lib
 - C:\TI\controlSUITE\libs\math\FPUfastRTS\V100\lib\IQmath_fpu32.lib



CPU Timer: Link Files to Project



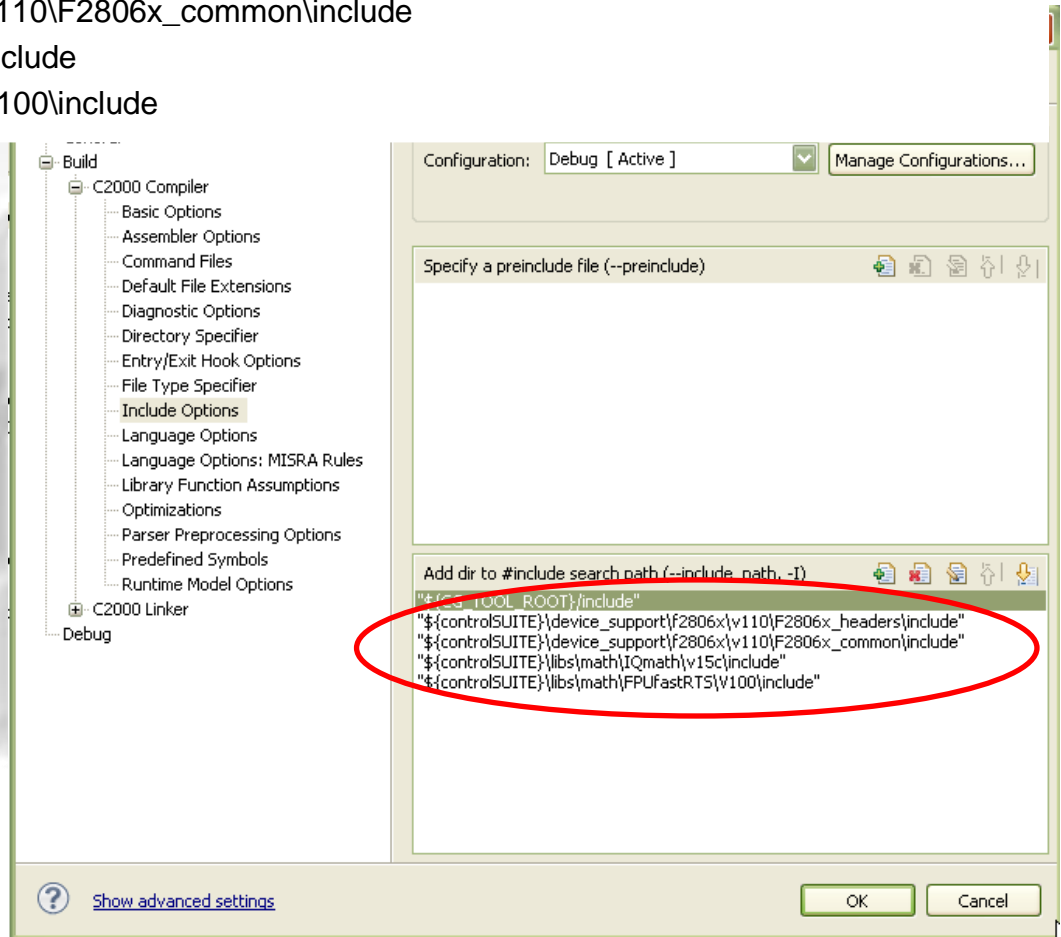
- Right-click on a source file and check the 'Properties'
- See how the 'Location' parameter references the Linked Resource Variable



CPU Timer: Modifying Project Properties



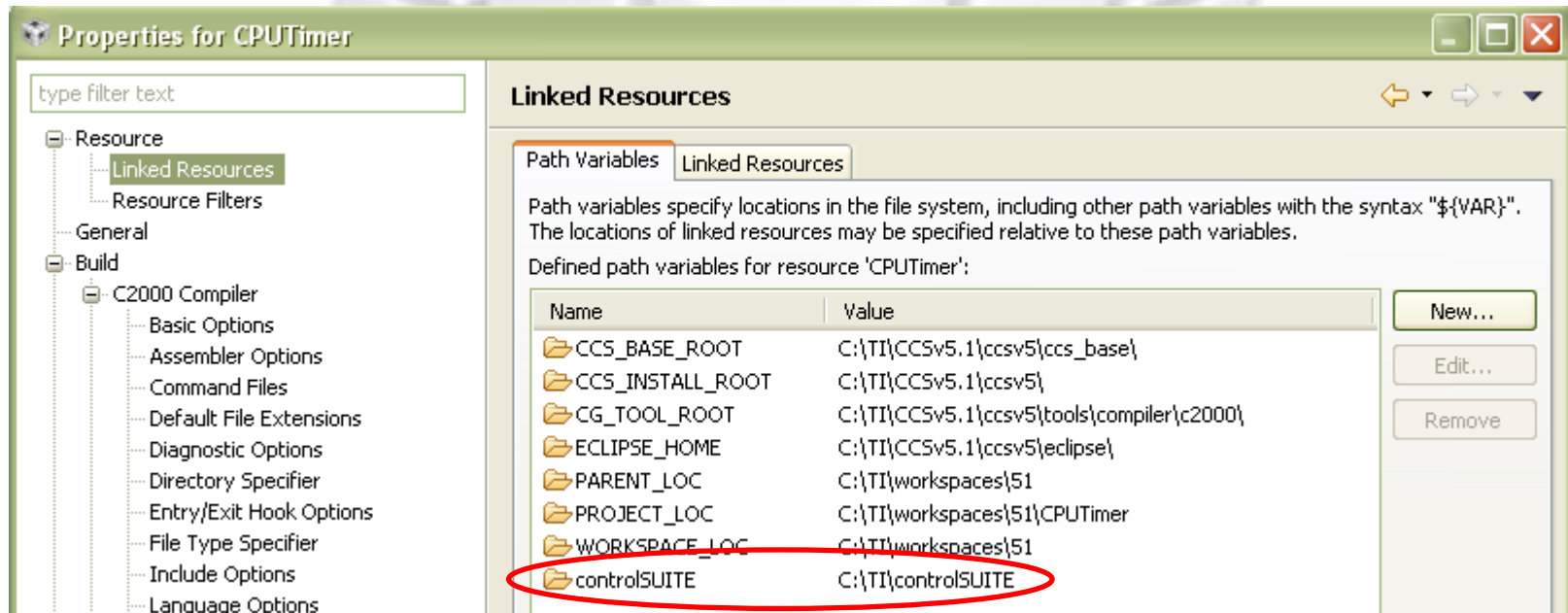
- Right-click on the project and select 'Properties'
- In the compiler 'Include Options', add the following entries to the list of include search paths:
 - `${controlSUITE}\device_support\f2806x\v110\F2806x_headers\include`
 - `${controlSUITE}\device_support\f2806x\v110\F2806x_common\include`
 - `${controlSUITE}\libs\math\IQmath\v15c\include`
 - `${controlSUITE}\libs\math\FPUfastRTS\V100\include`
- **'\${<BUILD VARIABLE>}' is the syntax to use a Build Variable in the project properties**
- **WARNING: Linked Resource Path Variables are only used when linking source files to a project. They can not be used for build options. Use Build Variables when modifying build options**



CPU Timer: Project Properties



- Go to 'Resource -> Linked Resources' to see all the Linked Resource Path Variables that is available to the project
 - This will show all variables created at the project level and workspace level
- See 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)



CPU Timer: 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

Properties for CPUTimer

type filter text

Resource

- Linked Resources
- Resource Filters
- General
- Build
- C2000 Compiler
 - Basic Options
 - Assembler Options
 - Command Files
 - Default File Extensions
 - Diagnostic Options
 - Directory Specifier
 - Entry/Exit Hook Options
 - File Type Specifier
 - Include Options
 - Language Options
 - Language Options: MISRA Rules
 - Library Function Assumptions
 - Optimizations

Linked Resources

Path Variables Linked Resources

Linked resources in project 'CPUTimer':

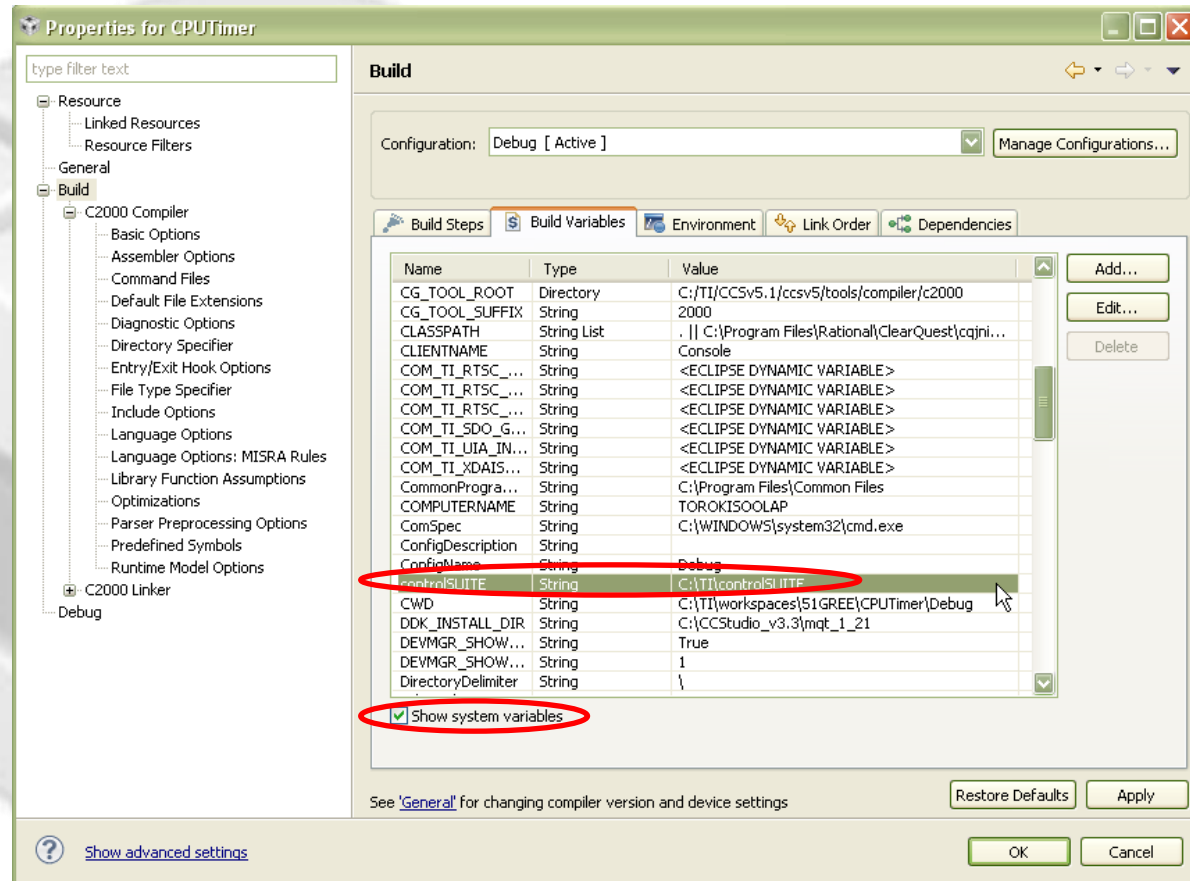
Resource Name	Location
Variable Relative Location	
Example_2806xCpuTimer.c	controlSUITE\device_support\f2806x\v110\F2806x_examples\cpu_timer\Example_2806xCpuTimer.c
F2806x_CodeStartBranch.asm	controlSUITE\device_support\f2806x\v110\F2806x_common\source\F2806x_CodeStartBranch.asm
F2806x_CpuTimers.c	controlSUITE\device_support\f2806x\v110\F2806x_common\source\F2806x_CpuTimers.c
F2806x_DefaultIsr.c	controlSUITE\device_support\f2806x\v110\F2806x_common\source\F2806x_DefaultIsr.c
F2806x_GlobalVariableDefs.c	controlSUITE\device_support\f2806x\v110\F2806x_headers\source\F2806x_GlobalVariableDefs.c
F2806x_Headers_nonBIOS.cmd	controlSUITE\device_support\f2806x\v110\F2806x_headers\cmd\F2806x_Headers_nonBIOS.cmd
F2806x_PieCtrl.c	controlSUITE\device_support\f2806x\v110\F2806x_common\source\F2806x_PieCtrl.c
F2806x_PieVect.c	controlSUITE\device_support\f2806x\v110\F2806x_common\source\F2806x_PieVect.c
F2806x_SysCtrl.c	controlSUITE\device_support\f2806x\v110\F2806x_common\source\F2806x_SysCtrl.c
F2806x_usDelay.asm	controlSUITE\device_support\f2806x\v110\F2806x_common\source\F2806x_usDelay.asm
IQmath_fpu32.lib	controlSUITE\libs\math\IQmath\v15c\lib\IQmath_fpu32.lib
rts2800_fpu32_fast_supplement.lib	controlSUITE\libs\math\FPUfastRTS\v100\lib\rts2800_fpu32_fast_supplement.lib

Edit... Convert... Delete...

CPU Timer: Project Properties



- Go to 'CCS Build' to see all the Build Variables that is available to the project
 - Only project level variables will be listed by default
 - 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








Project vs Workspace Level Variables

- The last few slides shows that ‘Linked Resource Path Variables’ and ‘Build Variables’ can be set at the project 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!

Version Control – Check in Which Files?

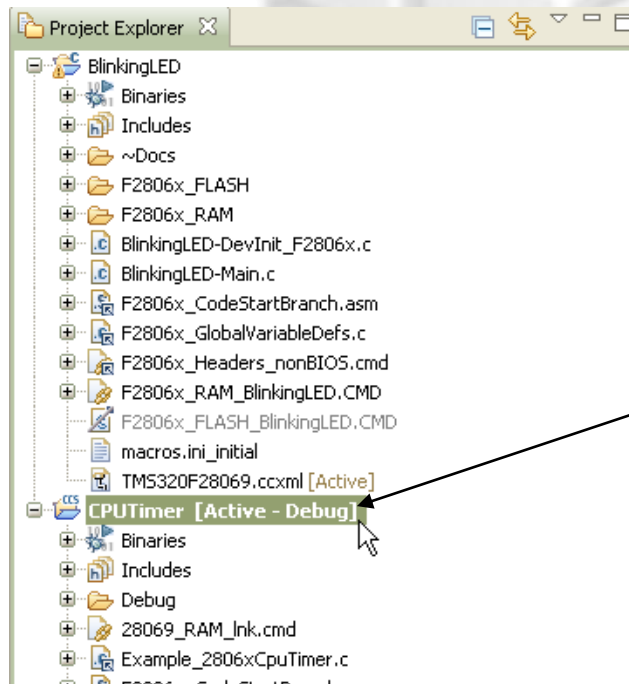
- **Several files/folders are generated by CCS inside the project folder**
 - .CPROJECT and .PROJECT are Eclipse project files and should be checked in
 - .CCSPROJECT is a CCS specific project file that should be checked in
 - .settings folder is a standard Eclipse folder that contains settings that apply to the project. This folder should be checked in
 - .launches directory is generated when you start a debug session. It is not related to the build of your project and is not necessary to check in
 - The contents of the project configuration folder (Debug/Release) do not need to be checked in

Name ▲	Size	Type
 .settings		File Folder
 Debug		File Folder
 .ccsproject	1 KB	CCSPROJECT File
 .cproject	17 KB	CPROJECT File
 .project	4 KB	PROJECT File

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.

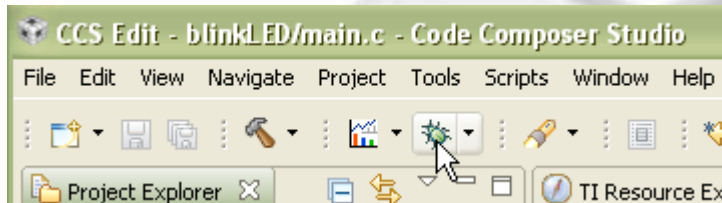


'CPU Timer' project is in 'Focus' since it has been selected. So pressing the 'Debug' button will build the project and start the debugger for the 'CPU Timer' project. Note how the project in focus is also highlighted in bold and with the work 'Active' next to it for easy identification

Build and Load the Program



- Make sure the 'CPU Timer' project is in 'Focus'. Then use the 'Debug' button



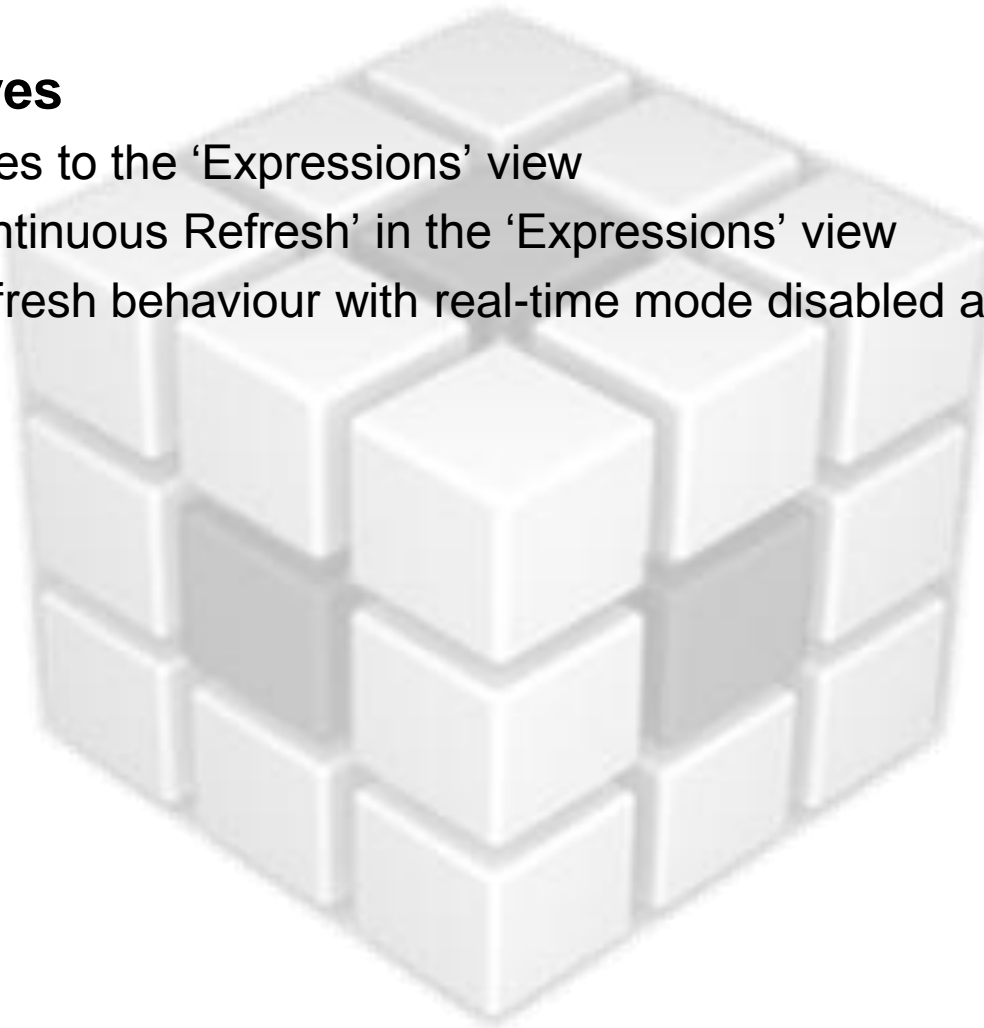


REAL-TIME MODE

Using Real-time Mode: Briefing

- **Key Objectives**

- Add variables to the 'Expressions' view
- Enable 'Continuous Refresh' in the 'Expressions' view
- Observe refresh behaviour with real-time mode disabled and enabled



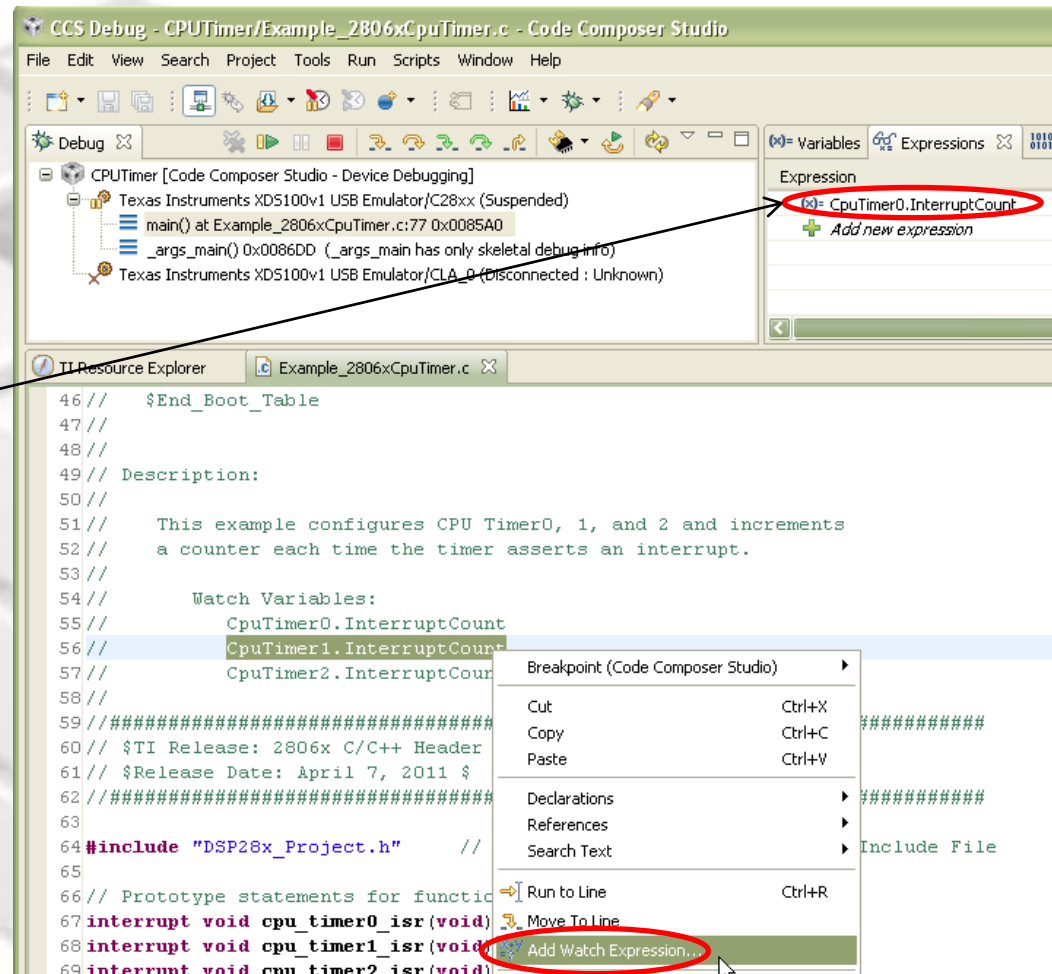
What is Real-time Mode?

- **Target normally need to be halted for debugger access (read/write memory/registers, etc)**
 - Many cases the debugger will "silently halt" a running processor – automatically halt the CPU, reads/writes memory, and then run again
 - This was the behavior in CCS 3.3 when you clicked "refresh" on a watch window at run-time.
 - If debugging a real-time audio routine, this would be perceptible as a glitch in the audio. However, for control algorithms such as spinning a motor or regulating a power supply it is very undesirable for the processor to momentarily halt
- **Real-time mode is a mode of operation that allows contents of memory/peripheral/register locations to be modified while the CPU is running and executing code and servicing interrupts**
 - User can also single step through non-time critical code while enabling time-critical interrupts to be serviced without interference
 - No software monitor is required (supported via HW)

Add Watch Variables to Watch Window



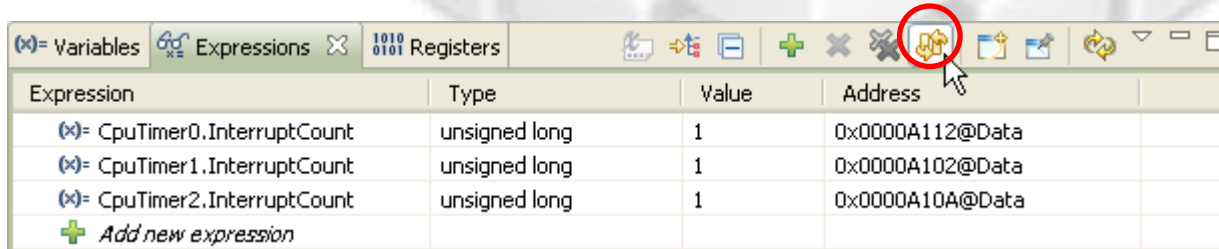
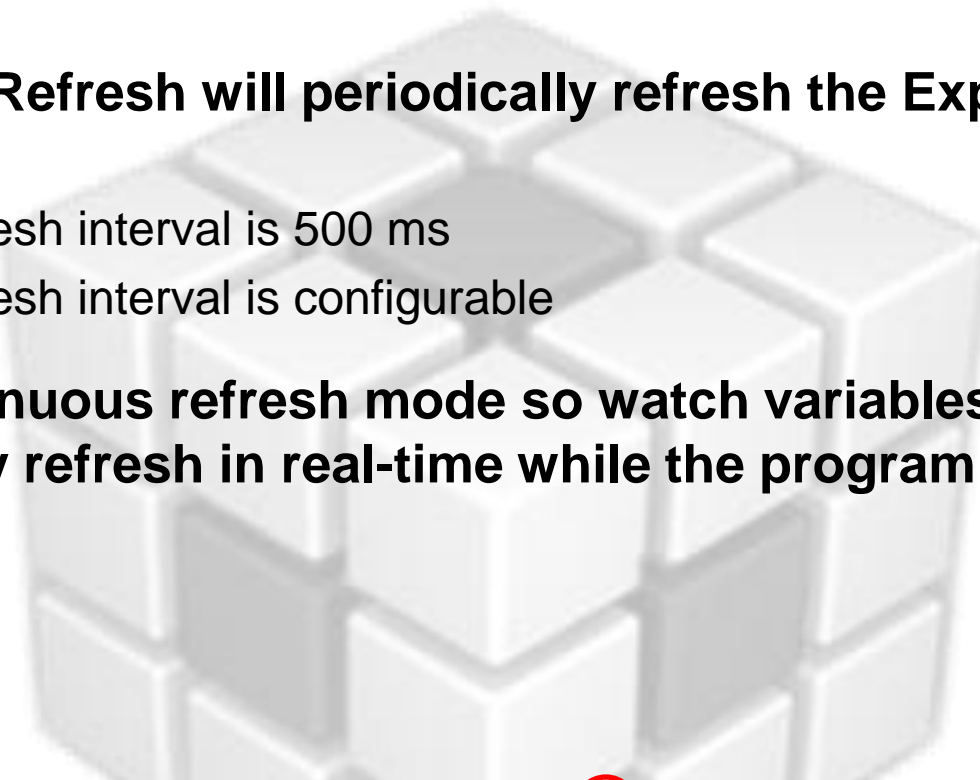
- Go to line 55 in “Example_2806xCpuTimer.c”
- Highlight ‘CpuTimer0.InterruptCount’, right-click, and select “Add Watch Expression”
 - See the variable appear in the ‘Expressions’ view
- Repeat for:
 - CpuTimer1.InterruptCount (line 56)
 - CpuTimer2.InterruptCount (line 57)



Expressions View: Continuous Refresh



- **Continuous Refresh will periodically refresh the Expressions view**
 - Default refresh interval is 500 ms
 - Default refresh interval is configurable
- **Enable continuous refresh mode so watch variables will continuously refresh in real-time while the program is running**

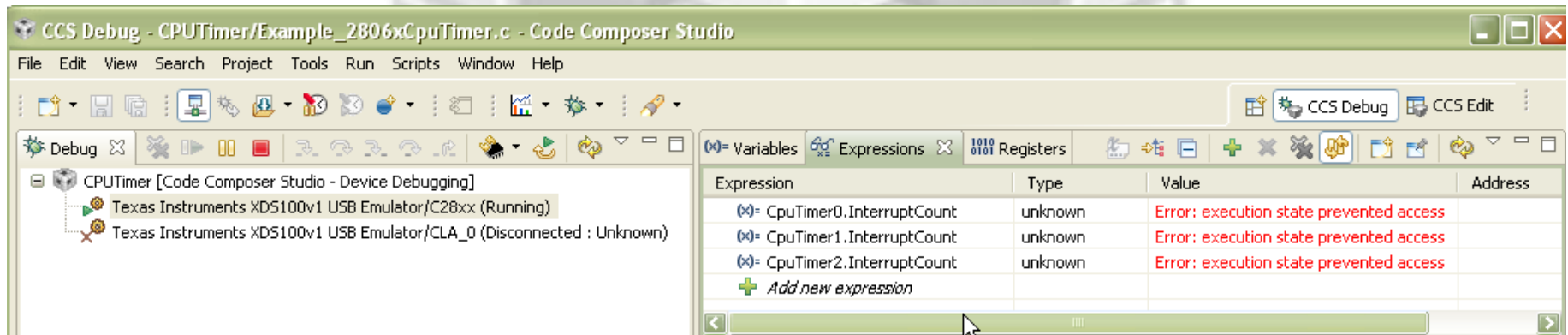


Expression	Type	Value	Address
(x)= CpuTimer0.InterruptCount	unsigned long	1	0x0000A112@Data
(x)= CpuTimer1.InterruptCount	unsigned long	1	0x0000A102@Data
(x)= CpuTimer2.InterruptCount	unsigned long	1	0x0000A10A@Data
+ Add new expression			

Run without Real-time Mode



- Run the program
- Note that the 'Expressions' view will not be able to update the values during the continuous refresh
- Without real-time mode, target must be halted for CCS to access target memory



Enable Real-time Mode



- Halt the target and then enable real-time mode to monitor the watch variables as they are changing

The screenshot shows the CCS Debug interface. A blue callout box labeled "Toggles real-time mode" points to a button in the toolbar. Below the toolbar, the "Debug" tab is active, showing a tree view of the target hierarchy. The "Texas Instruments XDS100v1 USB Emulator/C28xx" is selected. The "TI Resource Explorer" shows the source code for "Example_2806xCpuTimer.c". A dialog box titled "Texas Instruments XDS100v1 USB Emulator/C28xx" is open, asking "Do you want to enable realtime mode? Can't enter realtime mode unless debug events are enabled. Bit 1 of ST1 must be 0." A blue callout box labeled "Prompt will appear asking for confirmation" points to the dialog. The "Yes" button in the dialog is circled in red.

CCS Debug - CPUtimer/Example_2806xCpuTimer.c

File Edit View Search Project Tools Run Script

Debug

CPUTimer [Code Composer Studio - Device Debugging]

- Texas Instruments XDS100v1 USB Emulator/C28xx (Suspended)
- main() at Example_2806xCpuTimer.c:77 0x0085A0
- _args_main() 0x0086DD (_args_main has only skeletal debug info)
- Texas Instruments XDS100v1 USB Emulator/CLA_0 (Disconnected : Unknown)

TI Resource Explorer

Example_2806xCpuTimer.c

```
46 // $End_Boot_Table
47 //
48 //
49 // Description:
50 //
51 // This example configures CPU
52 // a counter each time the time
53 //
54 // Watch Variables:
55 // CpuTimer0.InterruptCount
```

Expression

Expression	Type
CpuTimer0.InterruptCount	unsig
CpuTimer1.InterruptCount	unsig
CpuTimer2.InterruptCount	unsig
Add new expression	

Do you want to enable realtime mode?
Can't enter realtime mode unless debug events are enabled.
Bit 1 of ST1 must be 0.

Yes No

Run with Real-time Mode



- Run the program
- Watch the variables in the Expressions view increment once per second as the CPU timers interrupt
- Real-time mode allows CCS to access target memory while the target is still running

The screenshot shows the CCS Debug window for the project 'CPUTimer/Example_2806xCpuTimer.c'. The 'Expressions' view is active, displaying a table of variables and their values. The 'Value' column for the three interrupt count variables is circled in red, showing the value '4' for each, indicating they have incremented once per second.

Expression	Type	Value	Address
CpuTimer0.InterruptCount	unsigned long	4	0x0000A112@Data
CpuTimer1.InterruptCount	unsigned long	4	0x0000A102@Data
CpuTimer2.InterruptCount	unsigned long	4	0x0000A10A@Data
Add new expression			

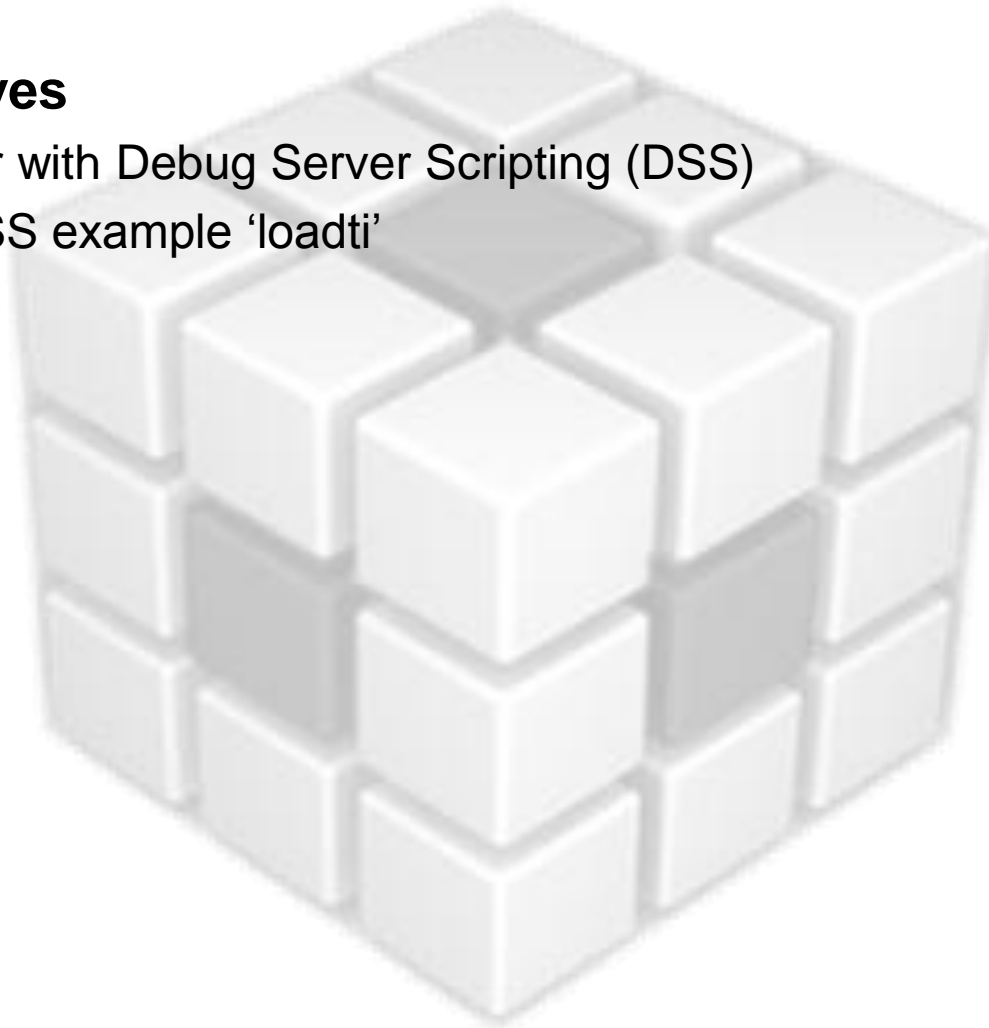


DEBUG SERVER SCRIPTING

Debug Server Scripting: Briefing

- **Key Objectives**

- Get familiar with Debug Server Scripting (DSS)
- Run the DSS example 'loadti'



Debug Server Scripting: Overview

CCS:

Development environment
based on Eclipse



```
add: RETURN
load: ENTRY nAddress: 0 nPage: 0 nLength: 500 sFile: .\sinewave\sine.dat
toAbsolutePath: ENTRY pathName: .\sinewave\sine.dat
toAbsolutePath: RETURN C:\ti\tools\DSS_1.0.0.4a\DebugServer\scripting\examples\C6
4\sinewave\sine.dat
load: Loading memory contents from file
load: RETURN
save: ENTRY nAddress: 0 nPage: 0 nLength: 500 sFile: .\sinewave\dump.dat nIOForm
at: 1 hAppend: false
toAbsolutePath: ENTRY pathName: .\sinewave\dump.dat
toAbsolutePath: RETURN C:\ti\tools\DSS_1.0.0.4a\DebugServer\scripting\examples\C6
4\sinewave\dump.dat
save: Saving memory contents to file
save: RETURN
readWord: ENTRY nPage: 0 nAddress: 0x0 nNumWords: 500 hSigned: false
readWord: Calculating size of a word
readWord: Word Size: 32 bits
readData: ENTRY nPage: 0 nAddress: 0x0 nTypeSize: 32 nNumValues: 500 hSigned: fa
lse
readData: Validating page
readData: Validating start address
readData: Getting memory object from debug session
readData: Setting start address: 0x0
readData: Setting buffer length: 2000
readData: Reading 500 value(s) from target
readData: Generating 500 element Java array from raw memory buffer
readWord: RETURN [J#1670940
15
30
45
TEST_PASSED: test_mem()
TEST_END: test_mem()
terminate: ENTRY
terminate: Unregistering this session from the DebugServer
stop: ENTRY
stop: RETURN
terminate: Disposing of interface objects
terminate: Removing CIO listener
terminate: Unregistering this session from the DebugServer
```

DSS:

Batch scripting
environment, with no
GUI dependencies



Scripting (Java)

DebugEngine (Java)

DebugServer.DLL (C++)

Debug Server Scripting: Intro

- **What is it?**
 - Set of Java APIs into the Debug Server (Debugger)
 - Can be used to automate any of the functionality of the debugger
- **Why use it?**
 - Automate testing and performance measurement
 - Supports all CPUs/Cores connected to the debugger
- **Scriptable through available 3P tools such as:**
 - Javascript (via Rhino)
 - Perl (via “inline::java” module)
 - Python (via Jython)
 - TCL (via Jacl/Tclblend)

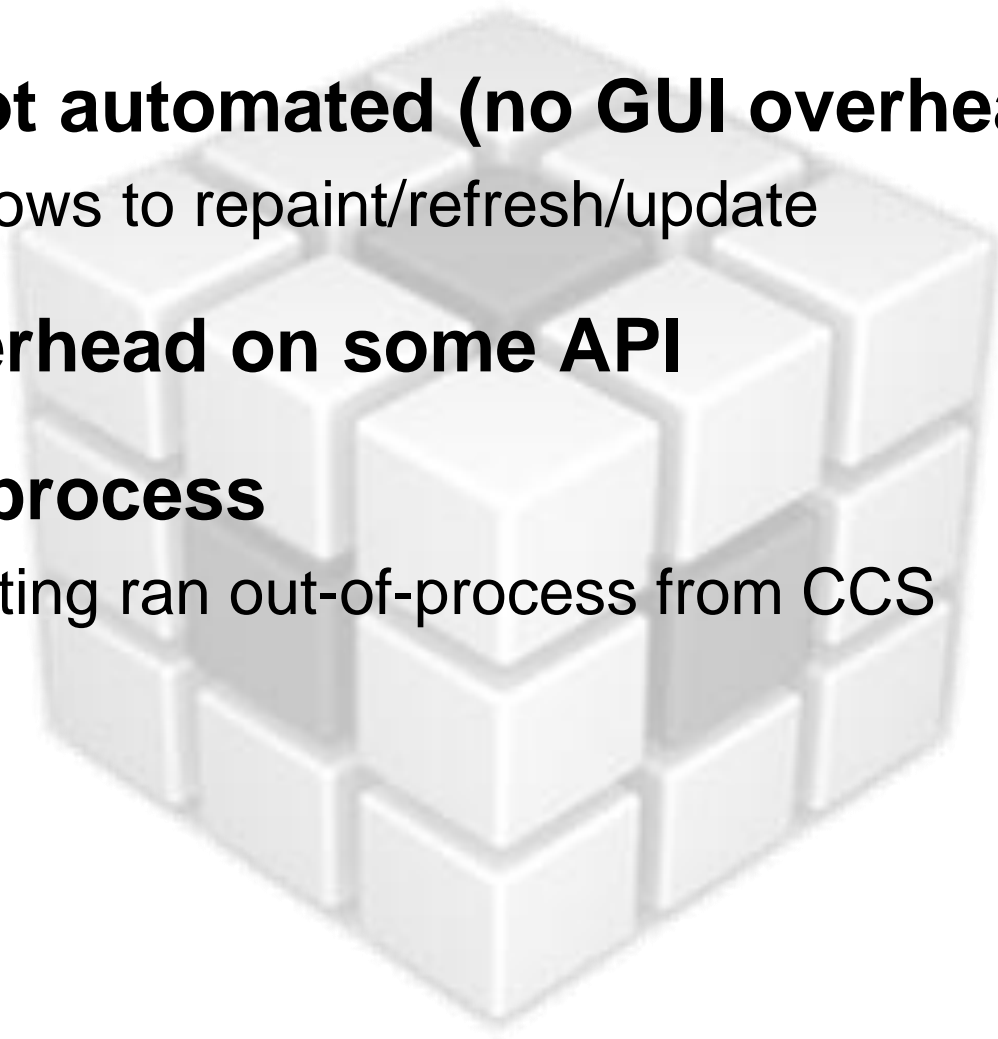
DSS vs. CCS Scripting



- **Advantages of DSS over CCScripting:**
 - Faster (No GUI overhead, runs in-process)
 - More robust and detailed logging
 - Exception handling
 - Have script handle exceptions thrown by DSS APIs without script aborting
 - Interfaces directly with Debug Server (*no debugger GUI*)
 - Standalone DSS install size much smaller than full CCStudio install
 - Light weight installs for lab machines for automation
 - Run multiple scripting instances simultaneously
 - Interactive console support from within CCS-Eclipse
 - More direct APIs supported (less reliance on GEL)
 - Pin/Port Connect APIs
 - Data Load/Save APIs + planned binary file support
 - CCS File I/O APIs (planned)
 - ATK support (planned)
 - Supported on Linux
 - CCS Scripting is Win32/COM

DSS: Faster

- **GUI is not automated (no GUI overhead)**
 - No windows to repaint/refresh/update
- **Less overhead on some API**
- **Runs in process**
 - CCScripting ran out-of-process from CCS



DSS: Benchmarks

- **Startup (launch debugger)**
 - CCScripting: ~5 s
 - DSS: ~1 s
- **Program Load API**
 - ~4 s overhead for program load API with CCScripting that does not exist with DSS
 - Loading small example *.out file with DSS takes ~65 ms
 - Same example *.out takes ~4 s with CCScripting
 - This can add up when program Load API is called constantly...
 - DSS: 200 loads = ~13 s
 - CCScripting: 200 loads = ~800 s
 - ~12 minutes saved with DSS!
- **1.5x-3x speed gains noticed on several DSS scripts migrated from CCScripting**

DSS: Logging

- **XML format (easy to parse)**
 - Can specify XSTL file to be referenced in XML log file
 - Configure how log information is displayed when opened in a web browser
 - XSTL Tutorial: http://www.w3schools.com/xsl/xsl_intro.asp
- **More information**
 - Sequence Numbers
 - Timing information (total, deltas, etc)
 - Status messages
- **Log CIO output**

DSS Log: Generated XML Log

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="SimpleTransform.xsl"?>
<log>
  <record>
    <date>2007-05-02T15:30:15</date>
    <millis>1178134215917</millis>
    <sequence>0</sequence>
    <logger>com.ti</logger>
    <level>FINER</level>
    <class>com.ti.ccstudio.scripting.environment.ScriptingEnvironment</class>
    <method>traceSetConsoleLevel</method>
    <thread>10</thread>
    <message>RETURN</message>
  </record>
  <record>
    <date>2007-05-02T15:30:15</date>
    <millis>1178134215917</millis>
    <sequence>1</sequence>
    <logger>com.ti</logger>
    <level>FINER</level>
    <class>com.ti.ccstudio.scripting.environment.ScriptingEnvironment</class>
    <method>getServer</method>
    <thread>10</thread>
    <message>ENTRY sServerName: LegacySetupServer.1</message>
  </record>
  ...

```

XSTL file to use

DSS Log: Open in Web Browser

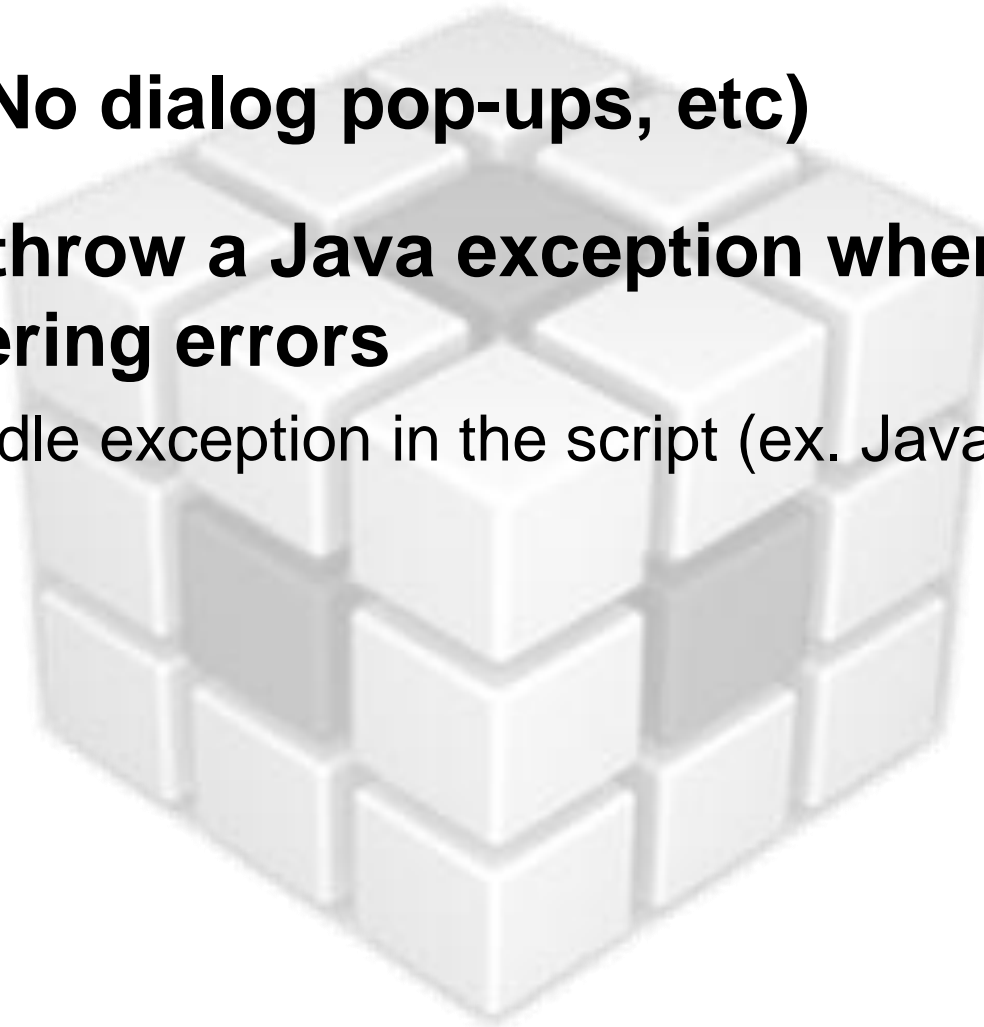
Debug Server Log

Total Execution Time: 89517 ms

Sequence	Time (ms)	Delta (ms)	Level	Method	Message
0	0		FINER	traceSetConsoleLevel	RETURN
1	0	0	FINER	getServer	ENTRY sServerName: LegacySetupServer.1
2	0	0	FINER	getServer	Getting definition for: LegacySetupServer.1
3	10	10	FINER	getServer	Constructing server
4	10	0	FINER	getServer	Starting server
5	20	10	FINER	start	ENTRY
6	50	30	FINER	start	RETURN
7	50	0	FINER	getServer	RETURN com.ti.debug.engine.scripting.LegacySetupServer@253498
8	50	0	FINER	ccsConfigClear	ENTRY
9	50	0	FINER	ccsConfigClear	Creating system setup object
10	2865	2815	FINER	ccsConfigClear	Clearing configurations
11	4166	1301	FINER	ccsConfigClear	Saving system configuration
12	6260	2094	FINER	ccsConfigClear	RETURN
13	6260	0	FINER	ccsConfigImport	ENTRY sConfig: C:\DSS\DebugServer\drivers\import\DM6446_ltl_endian_sim.ccs

DSS: Exception Handling

- **No GUI (No dialog pop-ups, etc)**
- **All APIs throw a Java exception when encountering errors**
 - Can handle exception in the script (ex. Javascript try-catch)



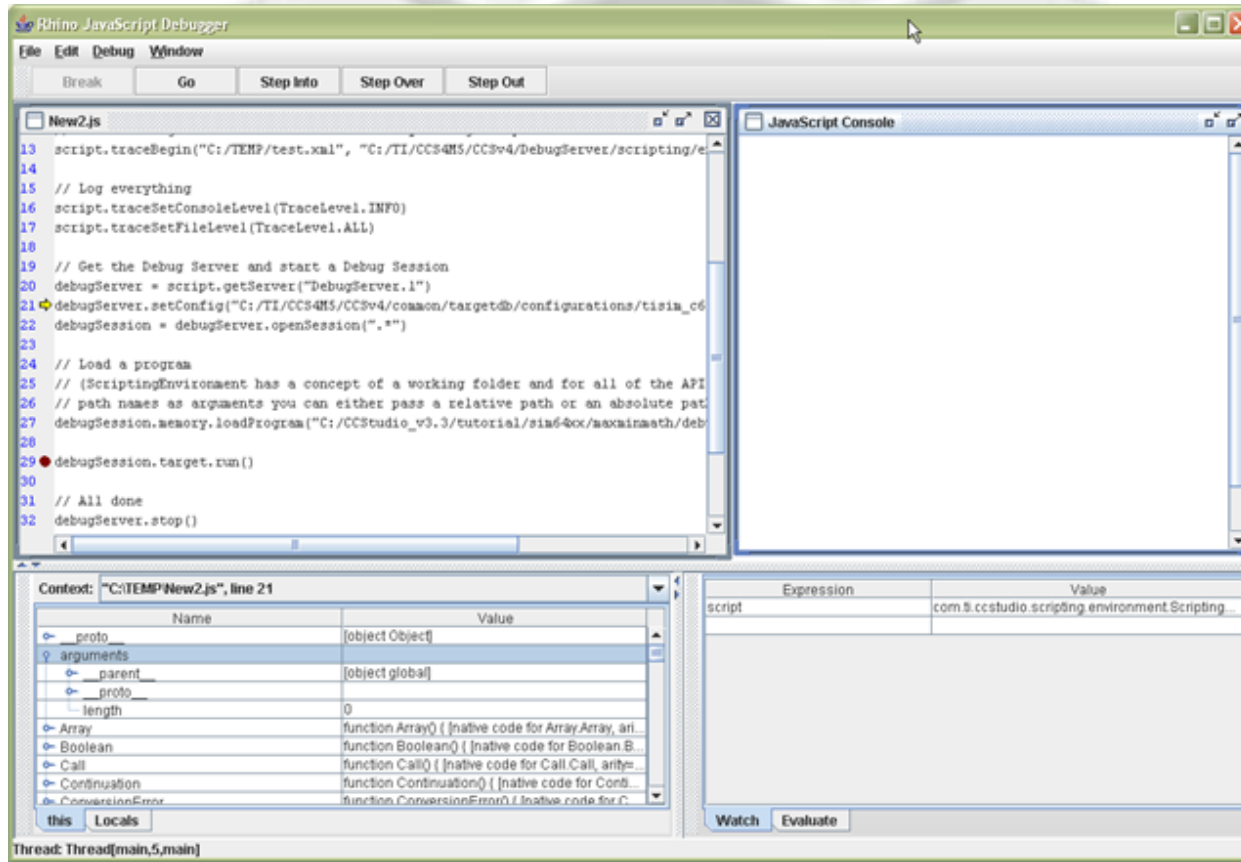
DSS: Exception Handling

- DSS APIs throw Java exceptions
- You can handle the exception in the script to fail gracefully or continue on

```
try {  
    debugSession.memory.loadProgram(testProgFile);  
} catch (ex) {  
    errCode = dssErrorHdl.getLastErrorID();  
    dssScriptEnv.traceWrite("errCode: " + errCode + " - " +  
        testProgFile + " does not exist!");  
    quit();  
}
```


DSS: Debug Scripts

- Rhino Debugger, which comes with DSS, supports:
 - Single stepping, Breakpoints, viewing script variables...



DSS: Using the Rhino Debugger

- **Enabling Rhino**

- Open 'dss.bat' in a text editor (found in <Install Dir>\ccsv5\ccs_base\scripting\bin)
- Edit line 65 where Rhino script engine is launched and replace !RHINO_SHELL! with !RHINO_DEBUGGER!



DSS Example Script (1)

```
// Import the DSS packages into our namespace to save on typing
importPackage(Packages.com.ti.debug.engine.scripting);
importPackage(Packages.com.ti.ccstudio.scripting.environment);
importPackage(Packages.java.lang);

// Modify these variables to match your environment. Use forward slashes
var ccs5InstallDir = "C:/Program Files/Texas Instruments";
var ccs5ScriptingLabDir = "C:/CCSWorkshop/labs/scripting";

// Create our scripting environment object - which is the main entry point
// into any script and the factory for creating other Scriptable Servers and Sessions
var script = ScriptingEnvironment.instance();

// Create a log file in the current directory to log script execution
script.traceBegin(ccs5ScriptingLabDir + "/test.xml", ccs5InstallDir +
    "/ccsv5/ccs_base/scripting/examples/DefaultStylesheet.xml");

// Set trace levels for console and logs
script.traceSetConsoleLevel(TraceLevel.INFO);
script.traceSetFileLevel(TraceLevel.ALL);
```


DSS Example Script (2)

```
// Get the Debug Server and start a Debug Session
debugServer = script.getServer("DebugServer.1");
debugServer.setConfig(ccs5ScriptingLabDir + "/c6416_le_sim.ccxml");
debugSession = debugServer.openSession(".*");

// Load a program
debugSession.memory.loadProgram(ccs5ScriptingLabDir + "/mainapplication.out");

// Run the target
debugSession.target.run();

// All done
debugServer.stop();

script.traceWrite("TEST SUCCEEDED!");

// Stop logging and exit.
script.traceEnd();
java.lang.System.exit(0);
```


DSS Example: loadti

- DSS JavaScript example that functions as a command-line loader which can load/run an executable *.out file on TI targets
- Included example of DSS but works right out of the box for any program and target
- The lack of GUI dependency and ease of use make it an attractive automation tool, useful for quick sanity tests and batch regressions
- **Some basic supported functionality includes:**
 - C I/O standard output can be sent to the console and scripting logs
 - Generate detailed executions logs
 - Perform basic application benchmarking (using profile clock)
 - Pass arguments to 'main()'
 - Loading/saving data from host to target memory (and vice versa)
 - Resetting the target
- **Refer to 'readme.txt' for more details on loadti**
 - Found in: <Install Dir>\ccsv5\ccs_base\scripting\examples\loadti

Using loadti: Run 'BlinkingLED.out'



- Close CCS if it is still running
- Open a DOS command window
- Browse to the location of the 'lcd_test.out' program
 - Location: <WORKSPACE>\BlinkingLED\F2806x_FLASH\BlinkingLED.out
- `> set PATH=%PATH%;<CCS_INSTALL_DIR>\ccsv5\ccs_base\scripting\examples\loadti`
 - You need to replace <CCS_INSTALL_DIR> with your install location
- `> loadti -h` (for console help)
- `> loadti -c ..\TMS320F28069.ccxml BlinkingLED.out`
- The load will take some time since it is erasing/programming flash
- You can see when the program is running by seeing LD2 blink
- 'CTRL-C' to abort

DSS: Resources

- **DSS MediaWiki documentation (recommended reading):**
 - http://processors.wiki.ti.com/index.php/Debug_Server_Scripting
- **DSS API documentation:**
 - <INSTALL DIR>\ccsv5\ccs_base\scripting\docs\GettingStarted.htm
- **Command line project management utilities MediaWiki documentation:**
 - [http://processors.wiki.ti.com/index.php/Projects -
_Command_Line_Build/Create](http://processors.wiki.ti.com/index.php/Projects_-_Command_Line_Build/Create)

Scripting Console

- **Command line operation of CCS**
- **View->Scripting Console**
- **Press TAB for a list of commands**
 - Press TAB for partially typed commands for auto-complete feature
- **To get documentation for a command**
 - `js:> help <command>`
- **JavaScript shell and has access to all DSS APIs**
- **Run DSS scripts from the console**
- **Create your own custom commands**
 - Create a JavaScript function in a *.js file
 - Load the custom Javascript file
 - `loadJSFile <full path>/myCustomConsoleCmd.js`
 - Optional boolean second parameter that will auto-load the script
 - The function can now be called by name from inside the Scripting Console

Scripting Console

- **View- > Scripting Console**
- **Press TAB for a list of commands**
 - Press TAB for partially typed commands for auto-complete feature
- **To get documentation for a command**
 - `js:> help <command>`



```
js:>
IOMEMORY_COFF      IOMEMORY_FLOAT    IOMEMORY_HEX       IOMEMORY_INT
IOMEMORY_LONG      PAGE_DATA          PAGE_DATA_BYTE     PAGE_IO
PAGE_IO_BYTE       PAGE_PROGRAM       PORT_EXTERN         PORT_NOREWIND
PORT_READ          PORT_UPDATE        PORT_WRITE         addSrcSearchPath
asmStepIn          asmStepOut         asmStepOver        assertActiveDS      ba
bpViewId           br                 bra                 buildProject        bv
cleanProject       close              closeAllEditor      cls
connect            debugActiveProject debugViewId          enableLog
disViewId          disableLog         disconnect          expAdd
eval               execCmdFile        exit                halt
expRemove          expRemoveAll       expViewId           loadData
help               launchTIDebugger   loadCoff            loadRaw
loadJSFile         loadKeyword        loadProg            mmAdd
maximize           memFill            memViewId           mmReset
mmEnable           mmRemove           mmReset            modViewId
openView           pinConnect         pinDisconnect       pinList
portConnect        portList           print               probViewId
portDisconnect     readWord           regViewId           reload
projViewId         restart            rtdxBufferConfig   rtdxConfigMode
reset              rtdxDisable       rtdxEnable         rtdxLogFile
rtdxTrace          saveCoff           saveRaw             runSync
saveData           setAutoRunToMain  setWindowBuffer    sconViewId
services           srcStepOut         symLoad             srcStepIn
srcStepOver        unloadKeyword      unloadJSFile        unloadJSFile
varViewId
js:> help loadJSFile

loadJSFile(file,store)
Description: Load a JavaScript file or all the JavaScript files in the directory.
Arguments:
    path - the JavaScript file or a directory.
    store - [optional] true, store the file(s) to the preference, the script will auto
            reload the next time the view is open.

js:> |
```


Scripting Console

- Both the Scripting Console and GEL can be used for automation
- GEL can be used only within an active debug session and (mostly) apply to a debug context
- The Scripting Console can be used anytime (though certain commands will not work without a debug session)
- Scripting Console and GEL can both add menus to the custom 'Scripts'
 - GEL: `hotmenu <function>`
 - Scripting Console: `hotmenu.addJSFunction`

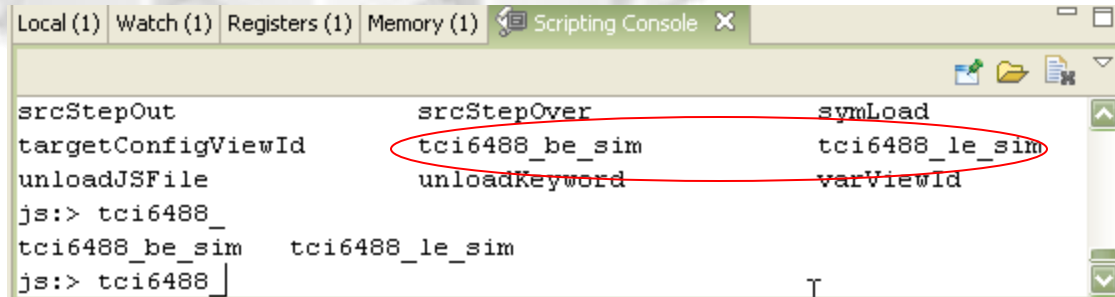
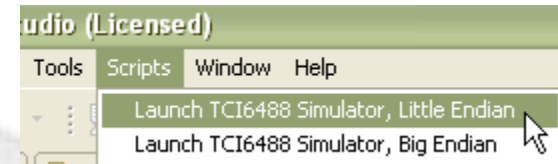
Scripting Console

```
// Add entries to the 'Scripts' menu
hotmenu.addJSFunction("Launch TCI6488 Simulator, Little Endian", "tci6488_le_sim()");
hotmenu.addJSFunction("Launch TCI6488 Simulator, Big Endian", "tci6488_be_sim()");

// Path to folder with target setup ccxml files
var setupConfigFileFolder = "C:/Documents and Settings/login/user/CCSTargetConfigurations";

// configure for a TCI6488 Symmetric Simulator, Little Endian
function tci6488_le_sim()
{
    ds.setConfig(setupConfigFileFolder + "/tci6488_le_sim.ccxml");

    debugSessionCPU1 = ds.openSession("*", "C64+_0");
    debugSessionCPU2 = ds.openSession("*", "C64+_1");
    debugSessionCPU3 = ds.openSession("*", "C64+_2");
}
```





ECLIPSE PLUG-INS

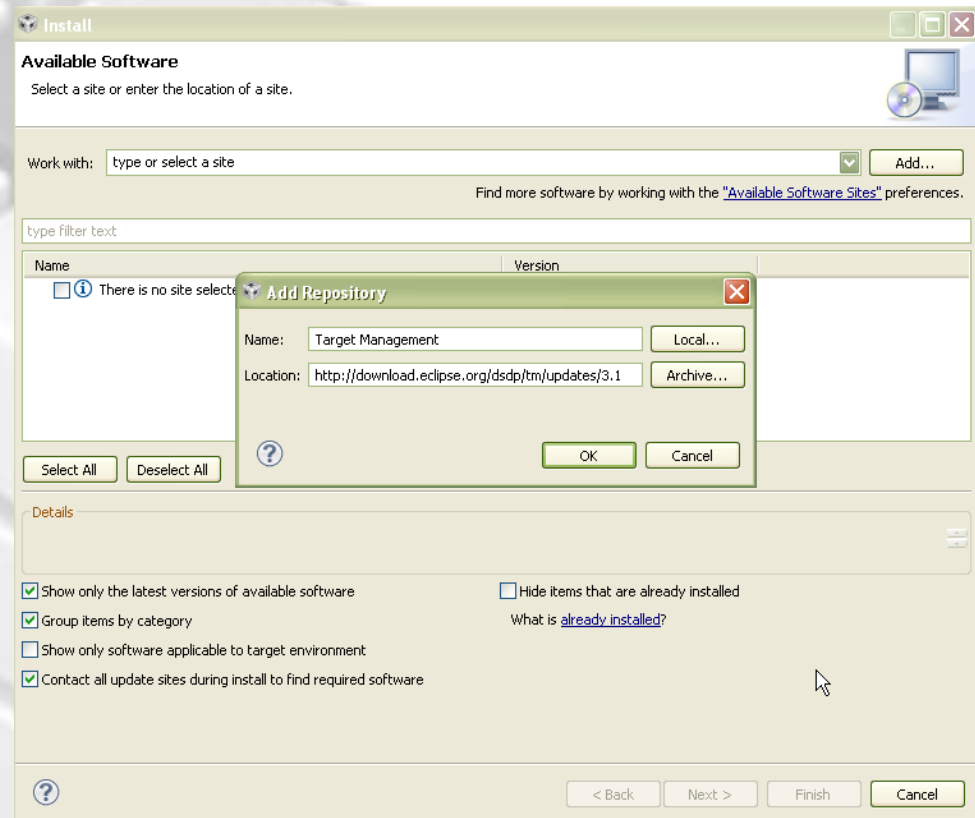
Eclipse Plug-ins - Basics

- **CCSv5 is based on Eclipse and is able to leverage many Eclipse of the huge selection of 3rd party plug-ins available**
 - <http://marketplace.eclipse.org/>
- **CCSv5 is based off Eclipse 3.7**
 - Look for plug-ins that support this version for best compatibility



Eclipse Plug-ins - Installation

- **Use the Eclipse Update Manager**
 - ‘Help -> Install New Software’ for new updates to install (specify remote site (URL) or local site (directory))
- **Drop-in**
 - Many plug-ins are simply downloaded as an archive and copied into the `.\ccsv5\eclipse\dropins` folder



controlSTICK

- **Want to learn more about the 28069 controlSTICK?**
 - Check out:
 - http://processors.wiki.ti.com/index.php/C2000_32-bit_Real-Time_MCU_Training



Questions?