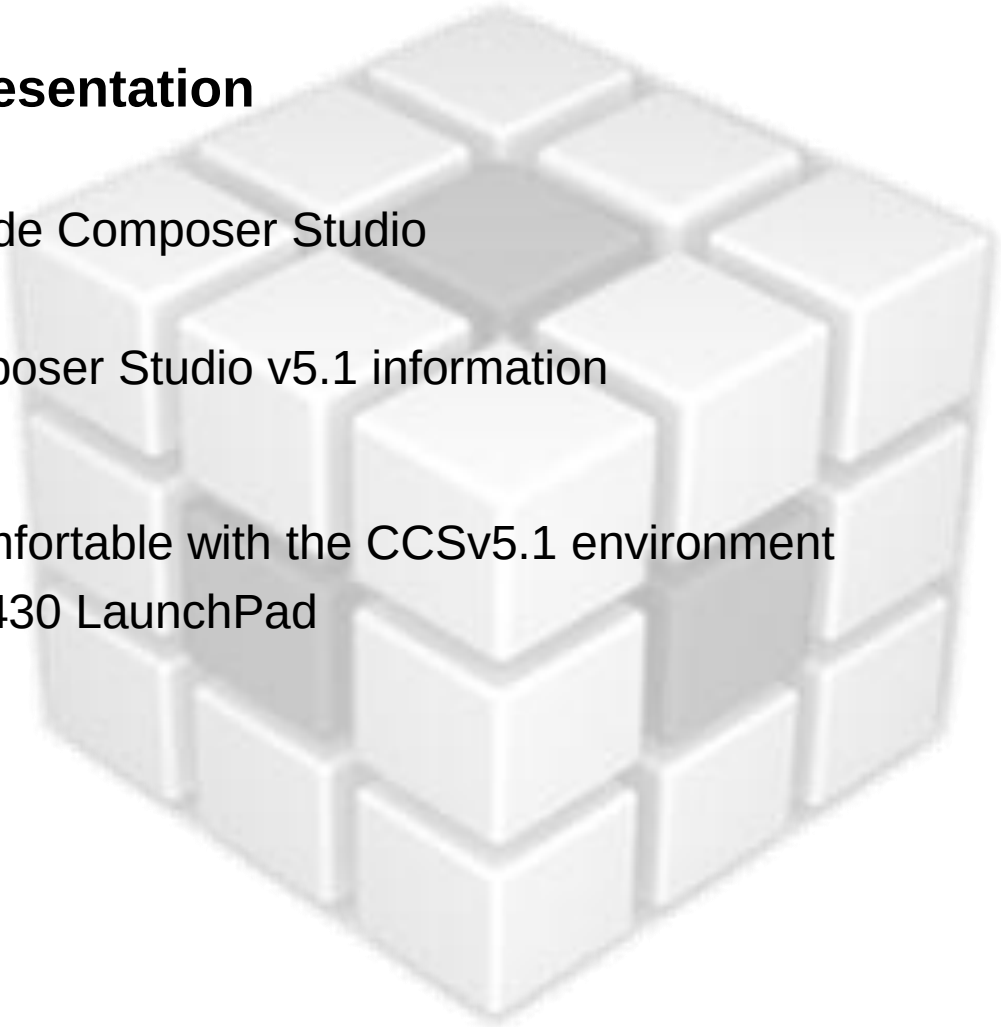
Code Composer Studio Overview Workshop

Agenda



- **Overview presentation**
  - Survey
  - What is Code Composer Studio
  - Roadmap
  - Code Composer Studio v5.1 information
- **Workshop**
  - Getting comfortable with the CCSv5.1 environment
  - Uses MSP430 LaunchPad

Survey

- What is your experience with CCS?
- What device(s) are you working with?

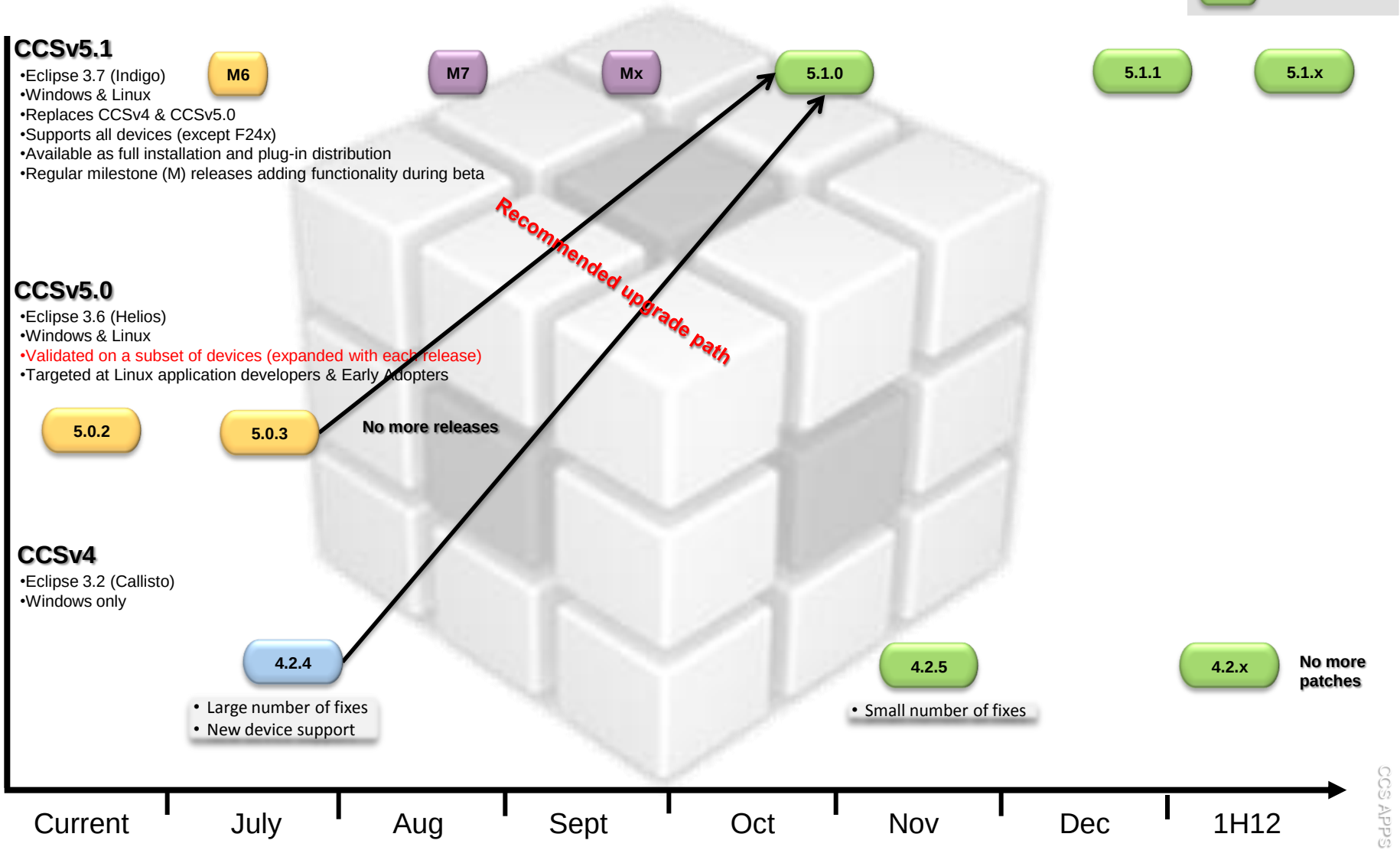
What is Code Composer Studio?

- **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…

Code Composer Studio v5

- **CCSv5 is split into two phases**
  - 5.0
    - Not a replacement for CCSv4
    - Targeted at users who are using devices running Linux & multi-core C6000
    - Addresses a need (Linux debug) that is not supported by CCSv4
    - Available today
  - 5.1
    - replacement for CCSv4 and is targeted at all users
    - Available fall 2011
- **Supports both Windows & Linux**
  - Note that not all emulators will be supported on Linux
    - SD DSK/EVM onboard emulators, XDS560 PCI are not supported
  - Most USB/LAN emulators will be supported
    - XDS100, SD 510USB/USB+, 560v2, BH 560m/bp/lan
  - http://processors.wiki.ti.com/index.php/Linux_Host_Support

Code Composer Studio Roadmap



Improvements for CCSv5.1

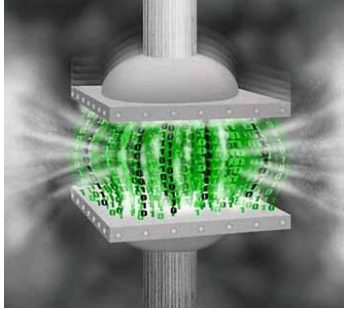
What's new in Eclipse?

- **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 <u>Smaller</u>**
  - 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 <u>Faster</u>**
  - Product is slow
  - Startup times and debugger responsiveness needs to be improved
- **Needs to be <u>Easier</u>**
  - 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



- **Today**
  - 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 will use a dynamic download**
  - 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

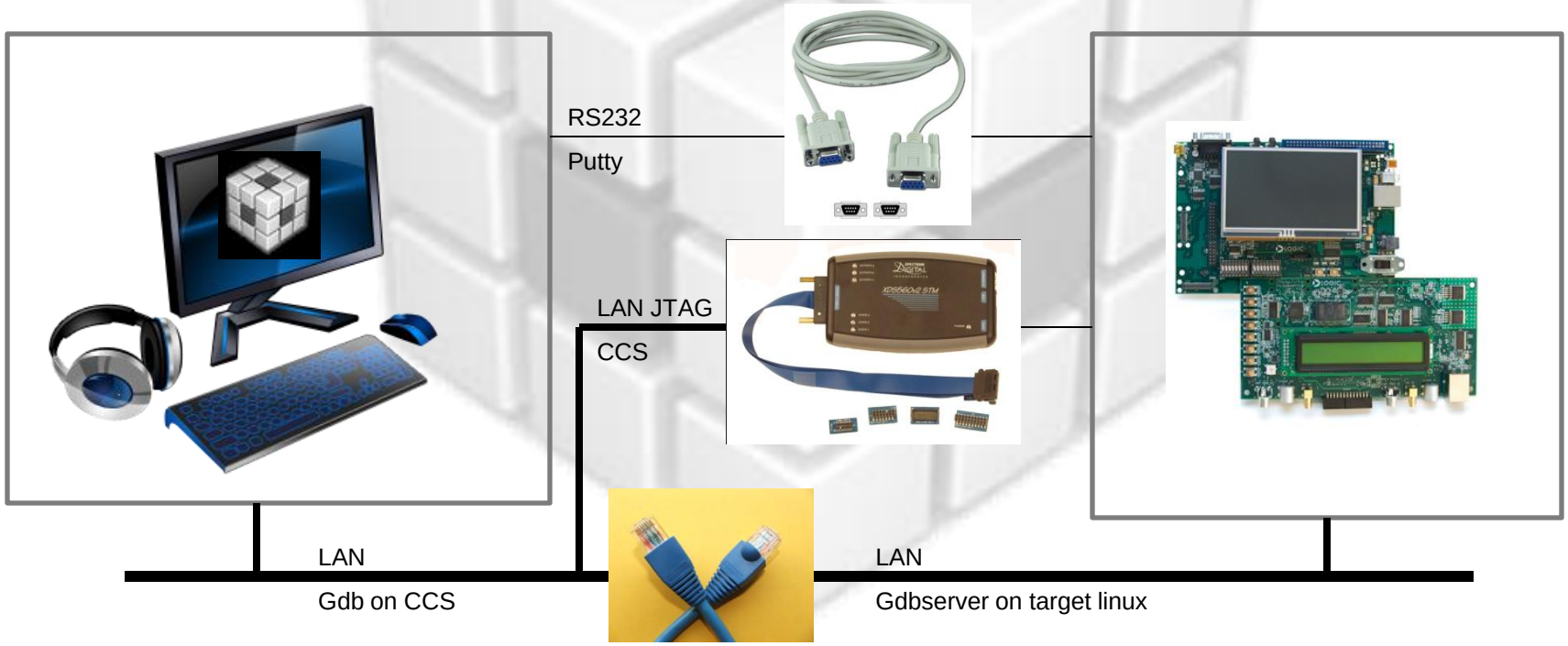
- **Simple Mode**
  - By default CCS will open in simple/basic mode
  - Simplified user interface with far fewer menu items, toolbar buttons
  - TI supplied Edit and Debug Perspectives
  - Simplified build options
- **Advanced Mode**
  - Use default Eclipse perspectives
  - Very similar to what exists 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

Easy– Common tasks

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

Linux Development

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



Upgrading to CCSv5

- **CCSv5.0**
  - Bundled with new Linux SDKs
  - Can be downloaded
    - http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5
  - Works with a CCSv4 license
- **CCSv5.1**
  - Replaces CCSv4
  - Requires a CCSv5 license
    - Users with active subscription will receive CCSv5.1 for free
    - Users with expired subscription can renew it to receive the upgrade
  - During alpha & beta you can use a CCSv4 license

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 will be much smoother**
  - Environment will be 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 will help with the learning curve
  - Improved documentation will help people get up to speed

Questions?

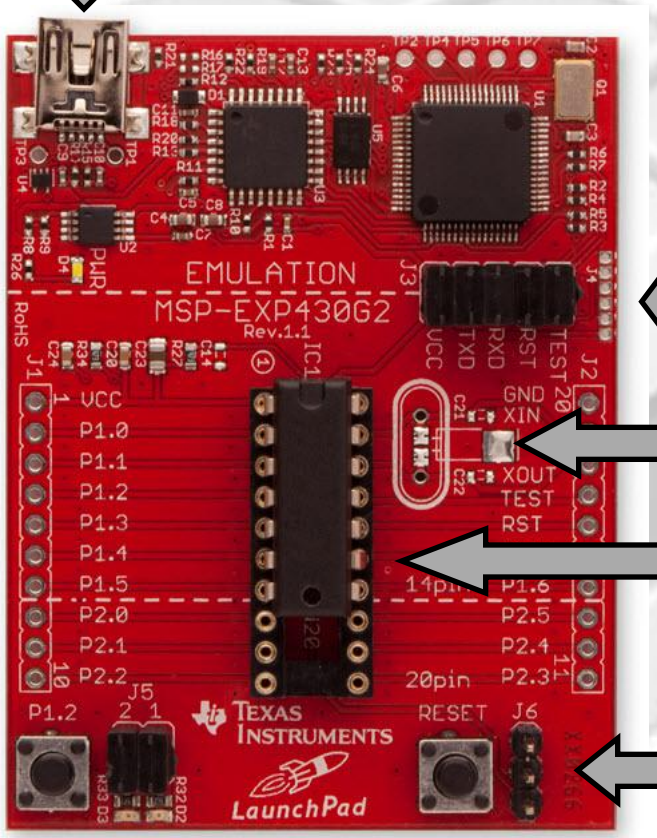
Getting Started with CCSv5 and LaunchPad

What is LaunchPad?

- **Low cost (under $5), easy-to-use development tool intended for beginners and experienced users alike for the MSP430G2xx Value Line series devices**

- **Complete development environment that features integrated USB-based emulation and all of the hardware and software necessary to develop applications**

LaunchPad: Development Board



USB Emulator Connection

Embedded Emulation

6-pin eZ430 Connector

Crystal Pads

Chip Pinouts

Part and Socket

P1.3 Button

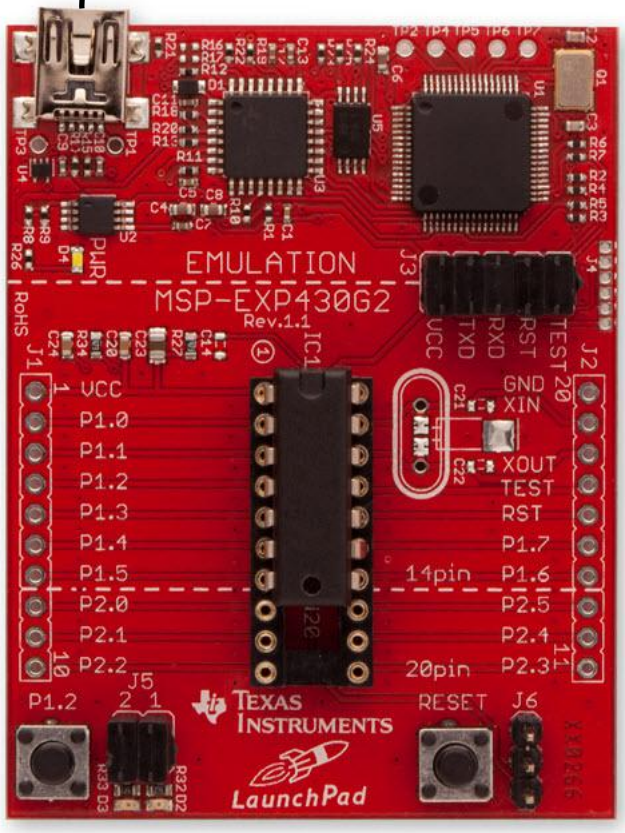
Power Connector

LEDs and Jumpers P1.0 & P1.6

Reset Button

LaunchPad: Hardware Setup

Connect mini-USB cable from PC to board



Workshop Instructions

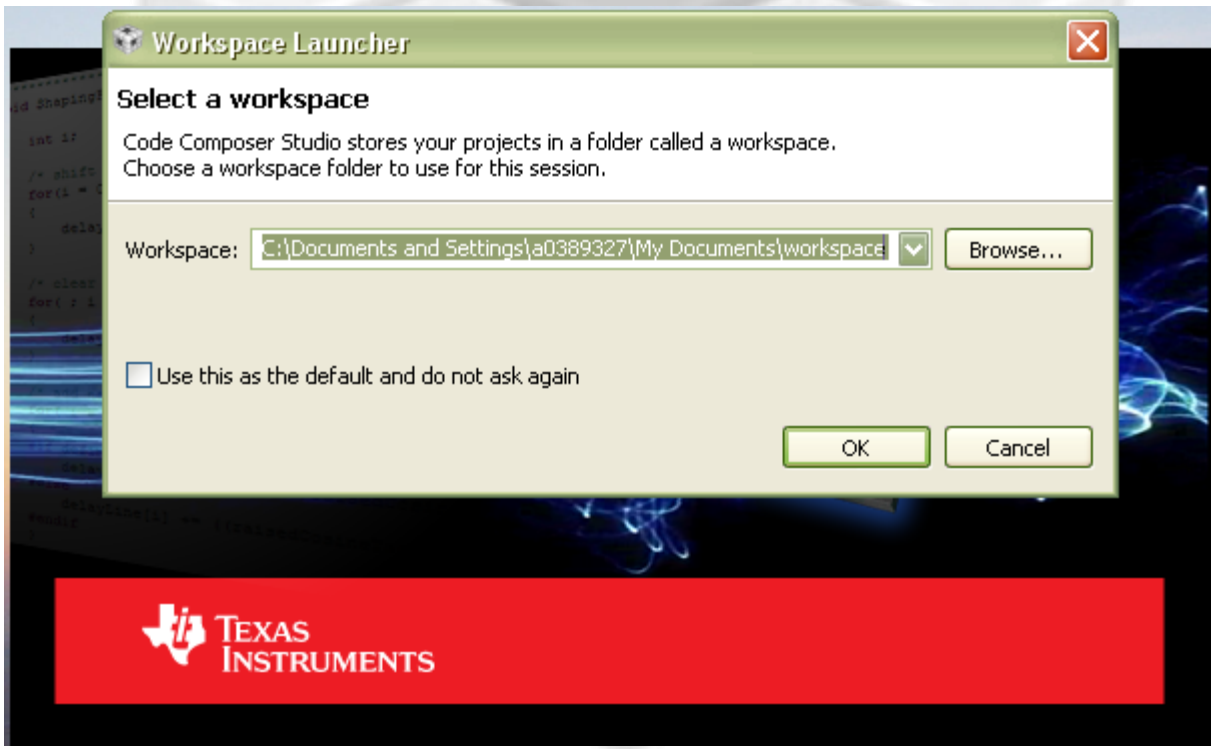
- This workshop is a mix of presentation material and activities
- Whenever you see the “Let’s Do it” picture on a slide, there are actions for you to perform

Blink LED1: Exercise Summary

- **Key Objectives**
  - Create and build a simple program to blink LED1
  - Start a debug session and load/flash the program on the LaunchPad
  - Run the program to blink LED1
- **Tools and Concepts Covered**
  - Workspaces
  - Welcome screen / Resource Explorer
  - Project Wizard
  - Template code
  - Project concepts
  - Basics of working with views
  - Debug launch
  - Debug control
  - Profile Clock

Workspace

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



Eclipse Concept: Workspaces

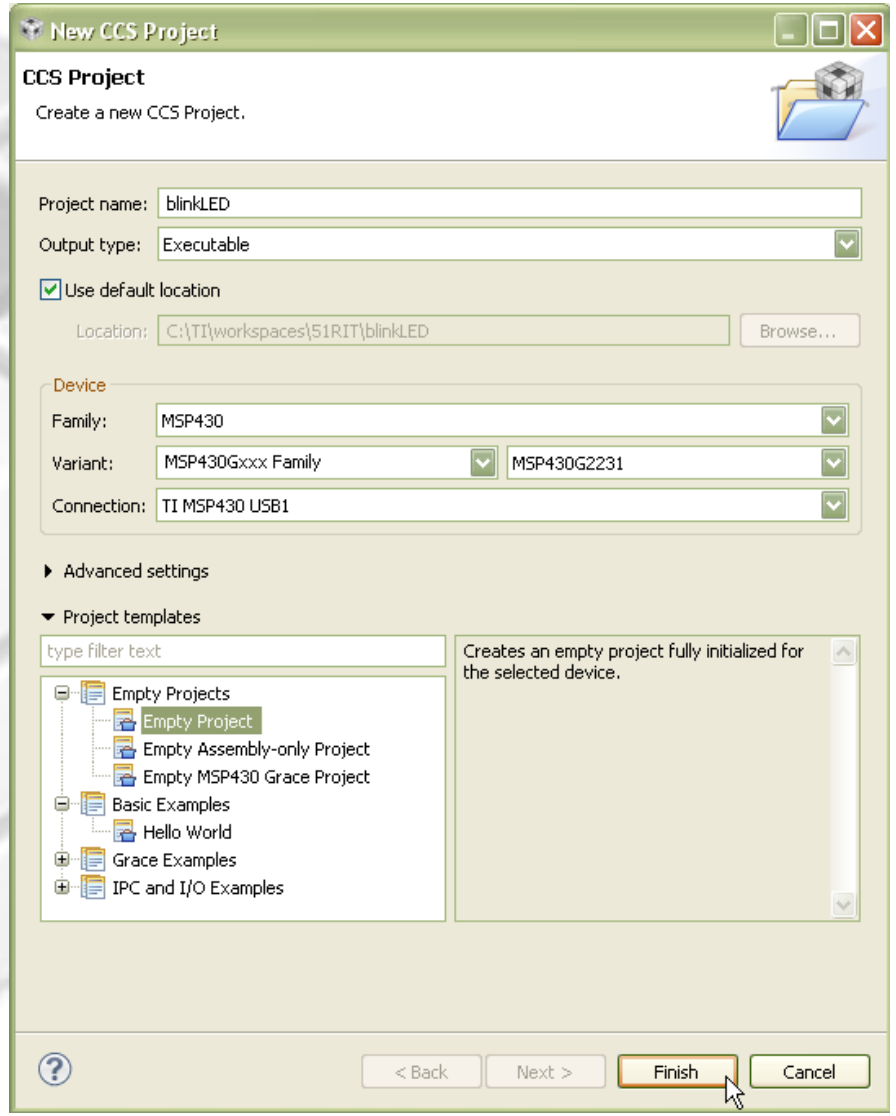
- **Main working folder for CCS**
- **Contains information to manage all the projects defined to it**
  - The default location of any new projects created
- **User preferences, custom perspectives, cached data for plug-ins, etc all stored in the workspace**
- **Workspaces are not to be confused with 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 ‘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

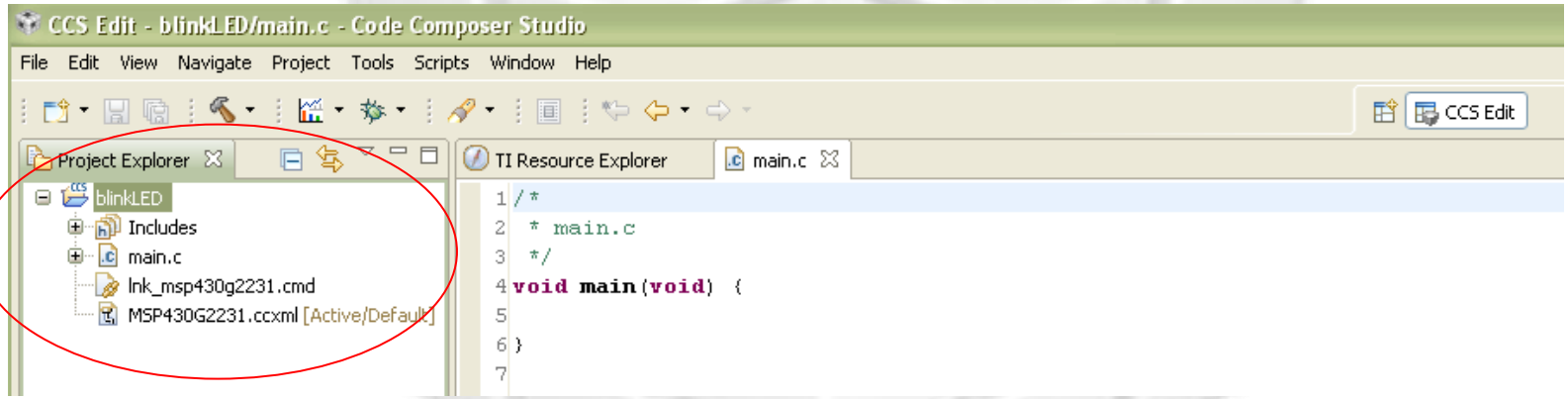
Create a New Project

- **Start the Project Wizard**
  - Select ‘New Project’ on the Welcome page
- **Fill in the fields as shown on the right**
- **Select ‘Finish’ when done**



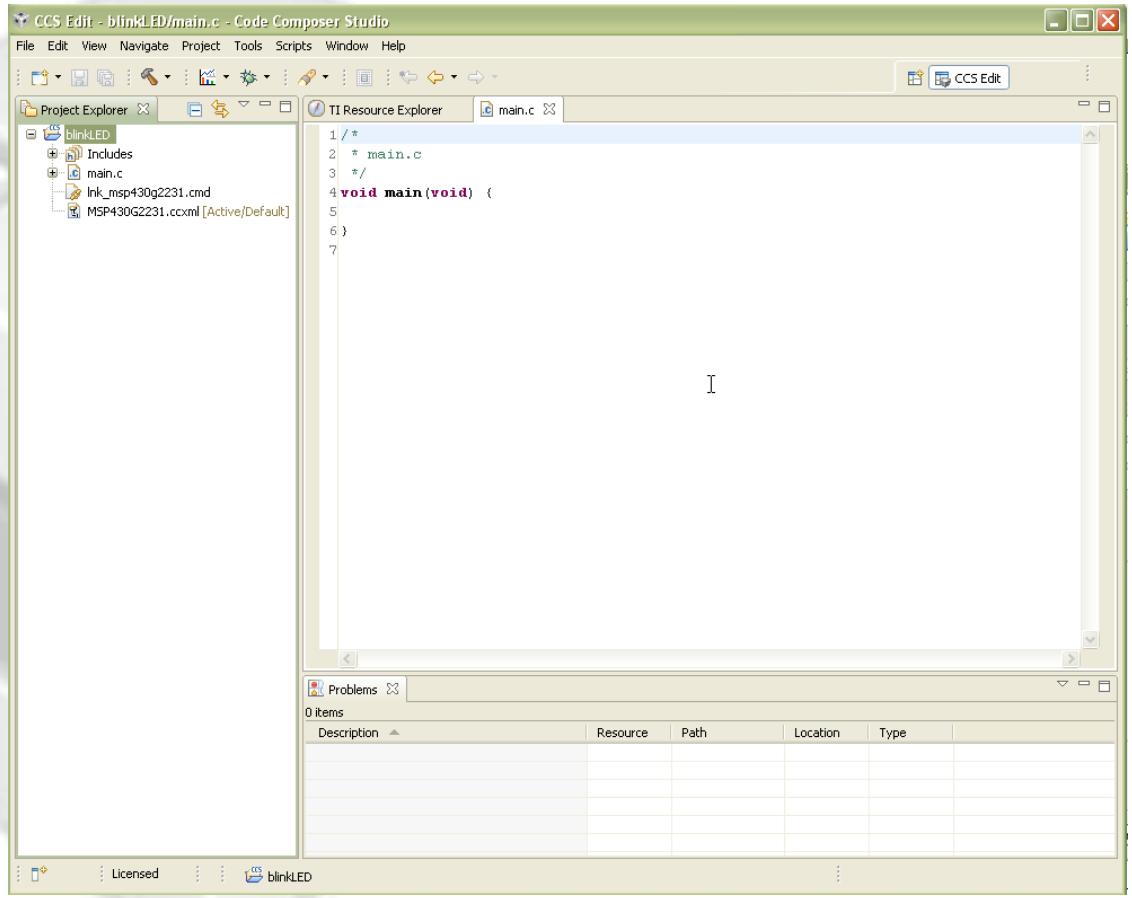
Create a New Project

- **Once the project is created, 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 Project Explorer' view**
- **Expand the project by clicking on it in the Project Explorer to see its contents**



Eclipse Concept: Workbench

- **'Workbench' refers to the main CCSv4 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**
  - Each debuggable CPU must have their own instance of the Control Window
  - Only one Control Window can be opened for each debuggable CPU
  - Information is not shared between each Control Window

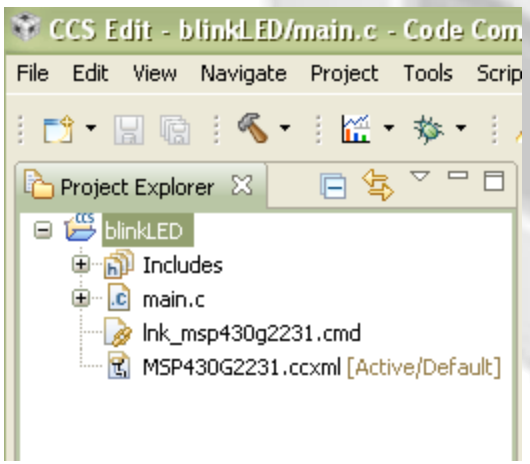
- **CCS 4/5**
  - One Workbench window can have visibility to all debuggable CPUs
  - Multiple Workbench windows can be opened ('Window->New Window')
  - Each Workbench window can differ visually (arrangement of views, toolbars and such), but refer to the same workspace and the same running instance of CCSv4
    - A project is opened from one Workbench will appear to be open in all the Workbench windows

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**
  - Newly created or imported projects are open by default
  - 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

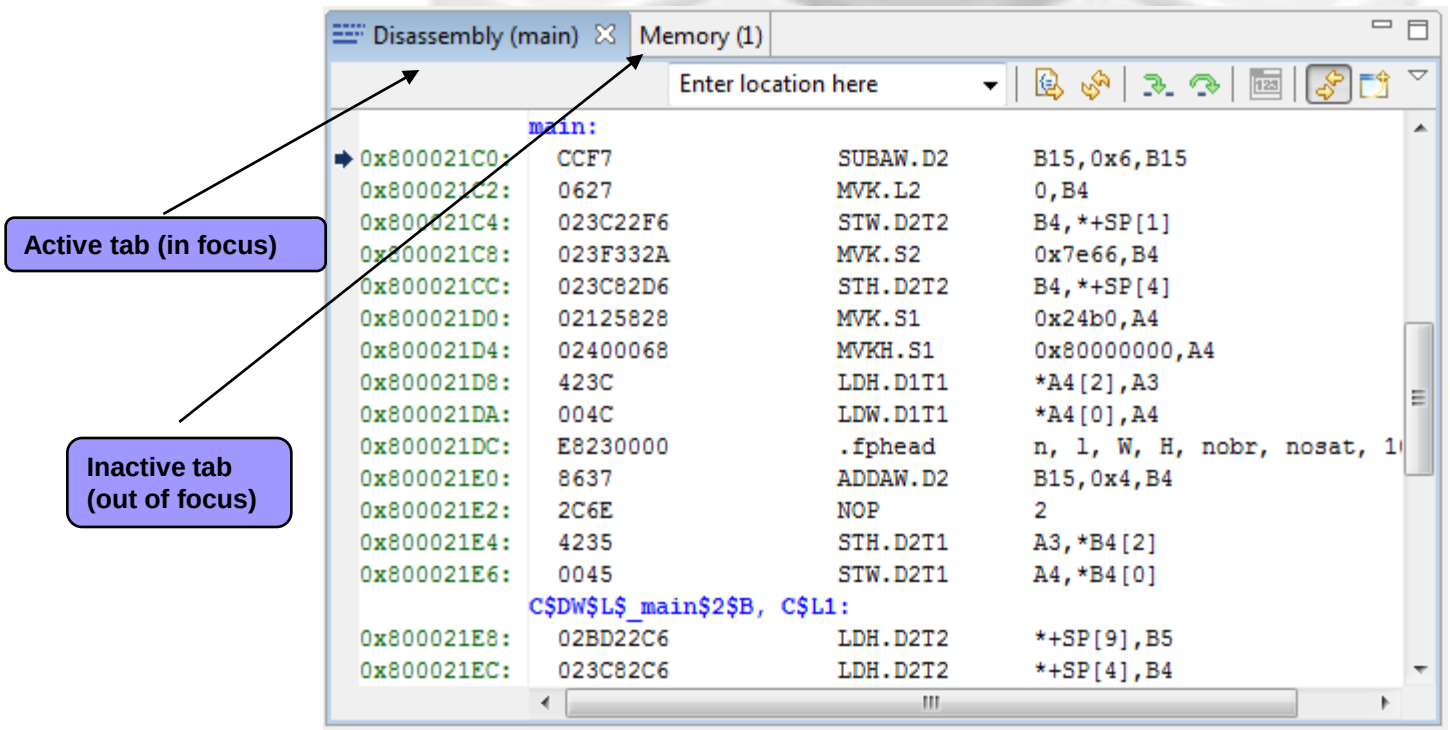
View: Project Explorer

- **Displays all projects defined in the active workspace**
- **The view is mostly a representation of the file system of the project folder**
  - Linked files are marked with a special link graphic in the icon
- **Use filters to hide various file types to reduce clutter in the view**
  - Default is to filter CCS generated project files (.*)



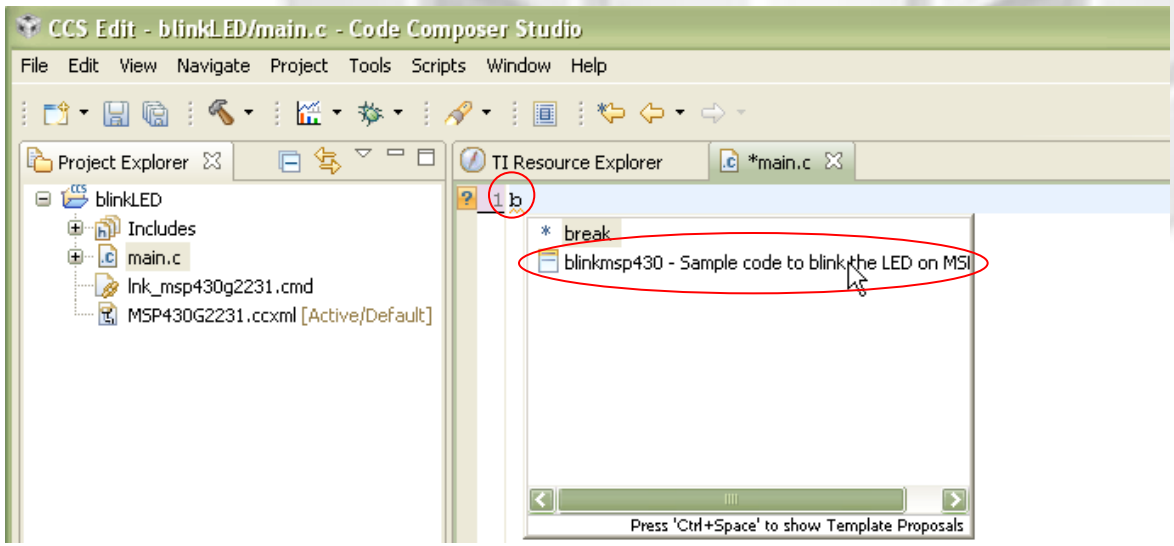
Eclipse Concept: Views

- **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



Adding Source Code to Blink LED1

- **Erase all content in the generated ‘main.c’ file**
  - Select the contents in the editor and hit delete
- **Use existing ‘blinkmsp430’ template to fill in code to blink LED1**
  - Type ‘b’ in the first line of the empty ‘main.c’ file
  - Press ‘CTRL+SPACE’ to bring up list of possible completions start with the letter ‘b’, it will include any matching code templates (or snippets)
  - Select ‘blinkmsp430’

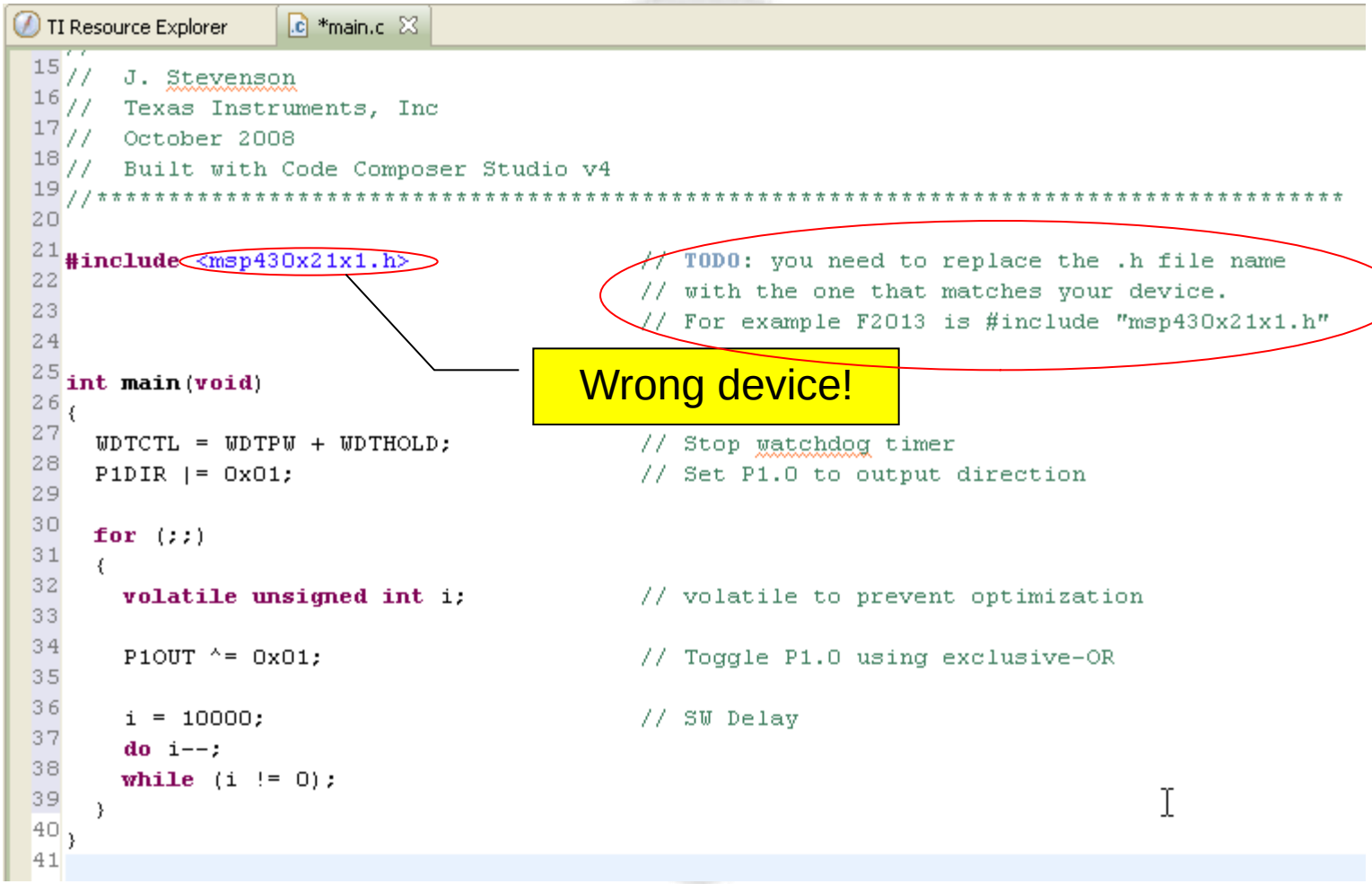


Adding Source Code to Blink LED1

TI Resource Explorer | *main.c

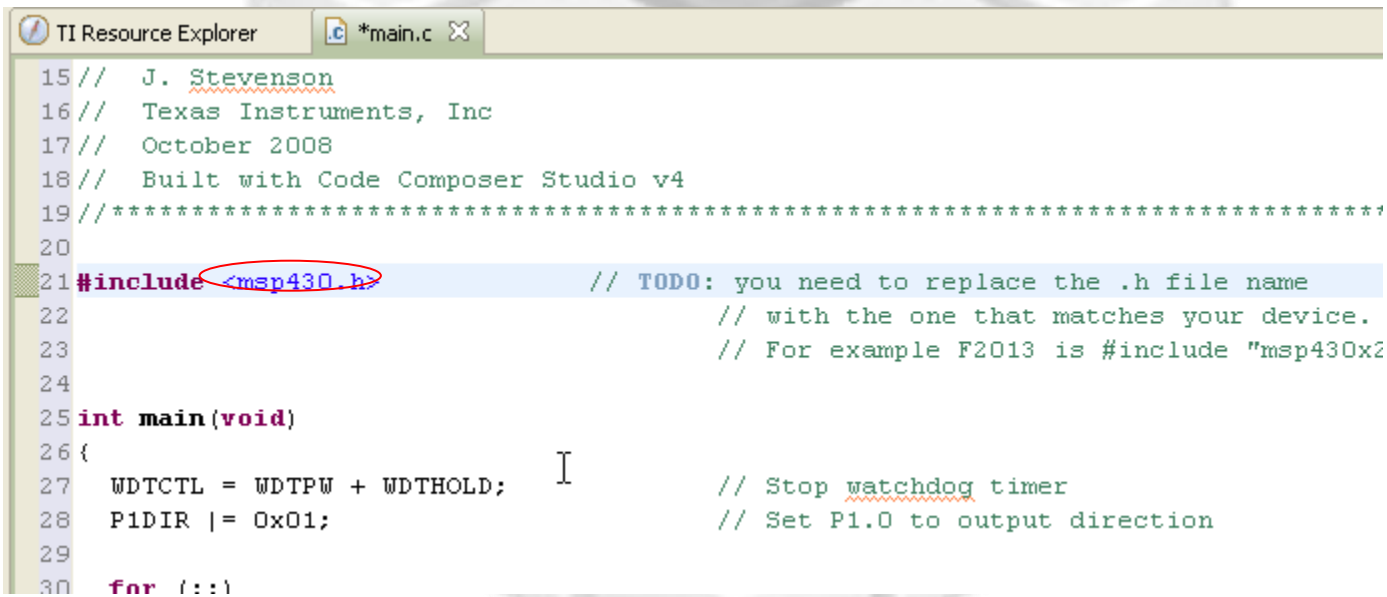
```
15 //
16 //  J. Stevenson
17 //  Texas Instruments, Inc
18 //  October 2008
19 //  Built with Code Composer Studio v4
20 //*****************************************************************************
21
22 #include <msp430x21x1.h>                  // TODO: you need to replace the .h file name
23                                           // with the one that matches your device.
24                                           // For example F2013 is #include "msp430x21x1.h"
25
26 int main(void)
27 {
28   WDTCTL = WDTPW + WDTHOLD;               // Stop watchdog timer
29   P1DIR |= 0x01;                          // Set P1.0 to output direction
30
31   for (;;)
32   {
33     volatile unsigned int i;              // volatile to prevent optimization
34
35     P1OUT ^= 0x01;                        // Toggle P1.0 using exclusive-OR
36
37     i = 10000;                            // SW Delay
38     do i--;
39     while (i != 0);
40   }
41 }
```

Adding Source Code to Blink LED1



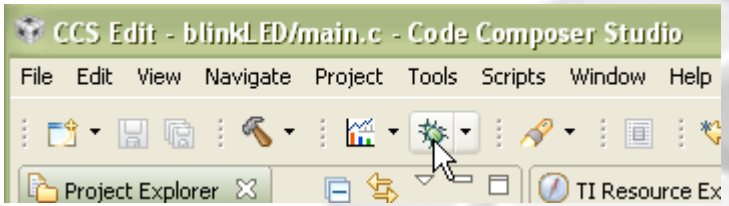
Adding Source Code to Blink LED1

- **Replace <msp430x21x1.h> with <msp430.h>**
  - 'msp430.h' will automatically pull in the correct header file for the device being used



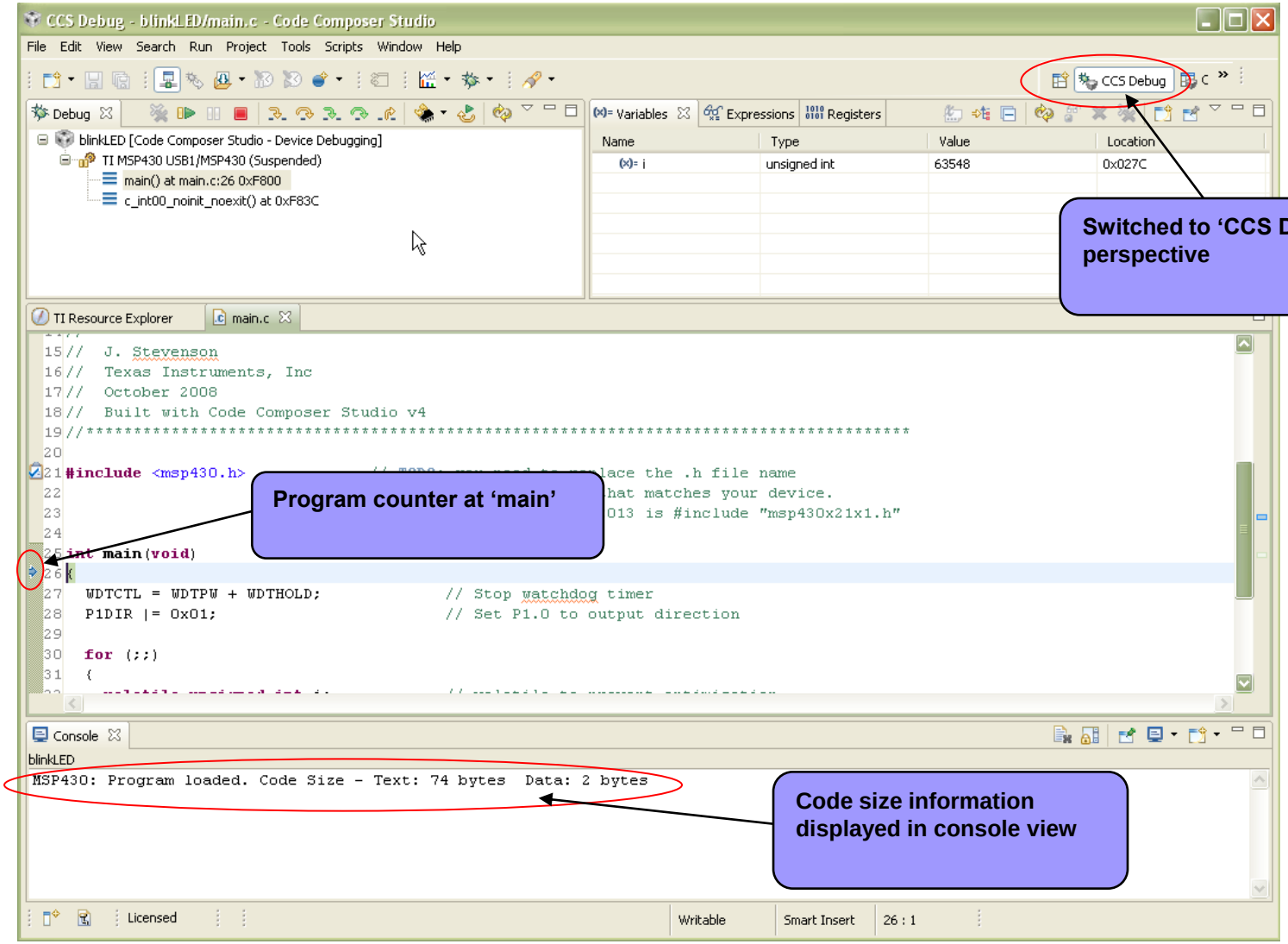
Build, Load/Flash the Program

- **Click the 'Debug' button:**



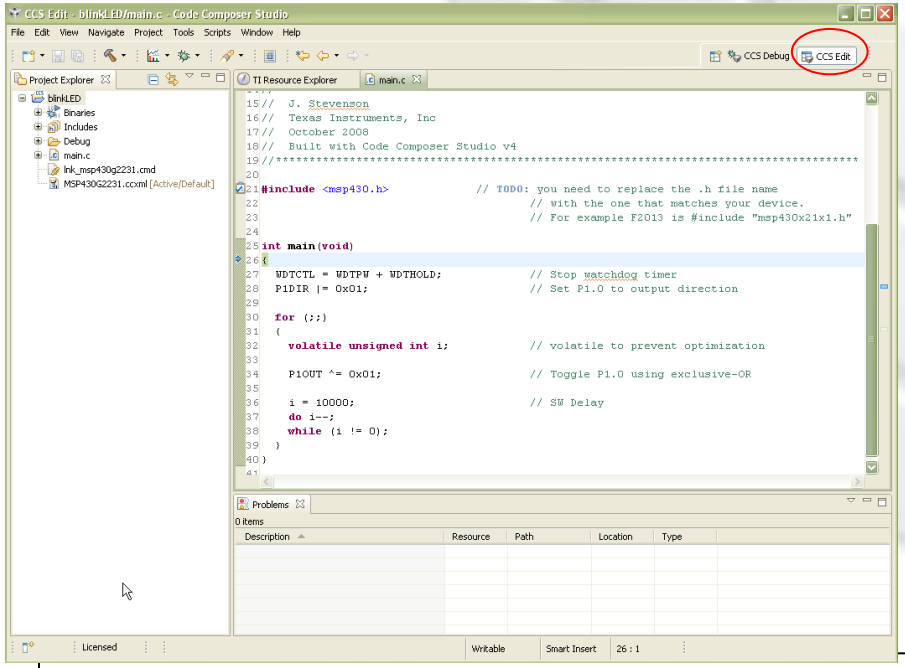
- **'Debug' button does multiple steps at once:**
  - Prompt to save source files
  - Build the project
  - Start the debugger (CCS will switch to the 'CCS Debug' perspective)
  - Connect CCS to the target
  - Load (flash) the program on the target
  - Run to 'main'
- **Don't want it to do all of the above at once? You can configure it to skip some steps (Debugger Options)**

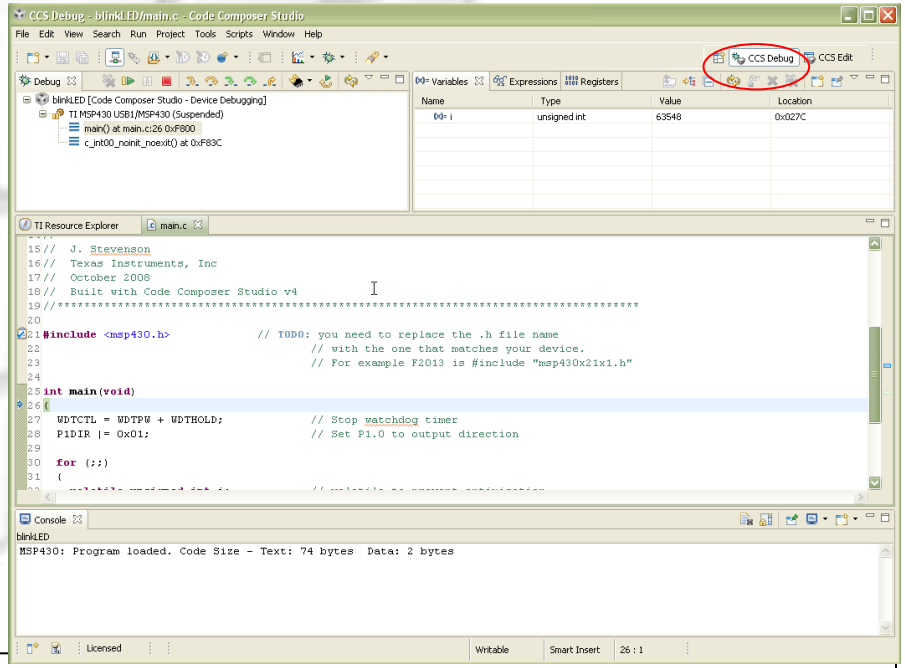
Build, Load/Flash the Program



Eclipse Concept: Perspectives

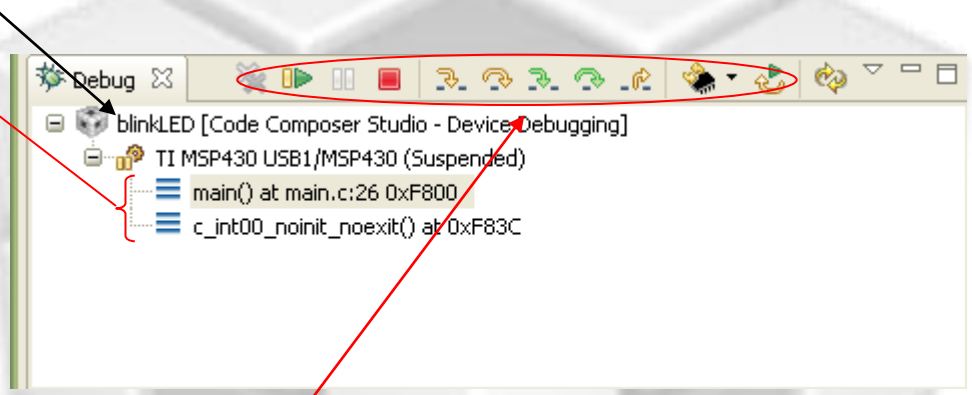
- Defines the initial set and layout of views in the Workbench window
- Similar in concept to CCSv3 'workspaces' (*.wks) except that multiple perspectives are available from the Workbench window (though only one can remain active at a time)
- Each perspective provides a set of functionality aimed at accomplishing a specific type of task (‘CCS Edit’ for project development, ‘CCS Debug’ for debugging, etc)
- Can create custom perspectives





Debug View

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



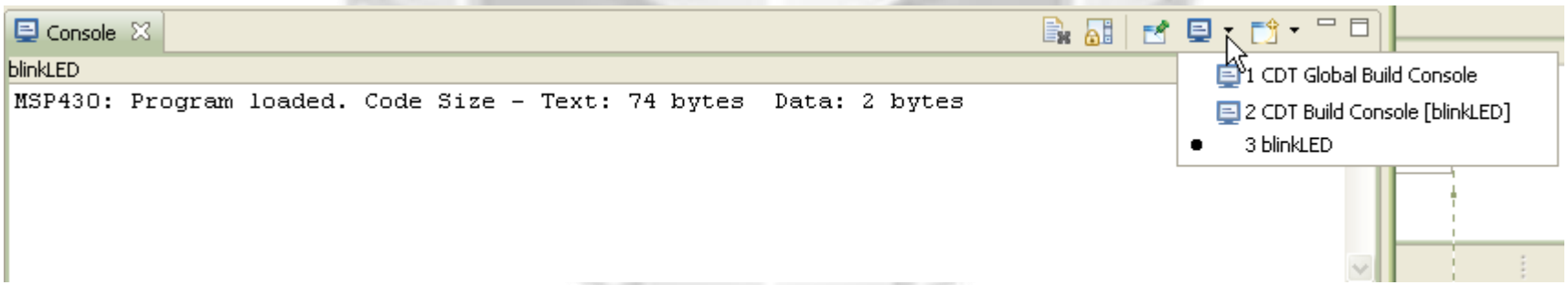
- **Buttons to 'run, halt, terminate (debug session), source and asm stepping, reset CPU, restart program**

View: 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

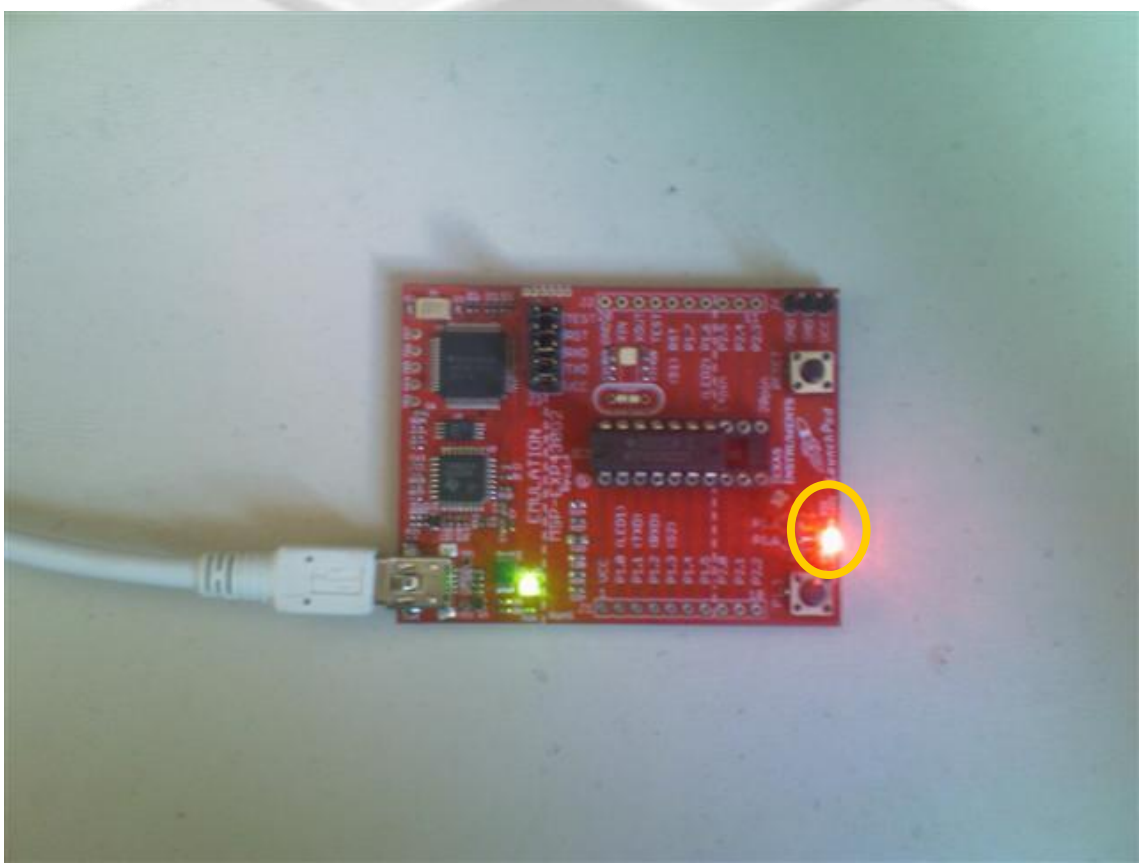
CCS 3.x had separate, dedicated views for build output, CIO output and debugger messages

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



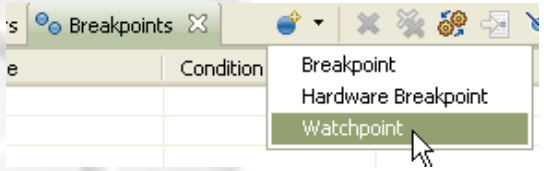
Blink LED1

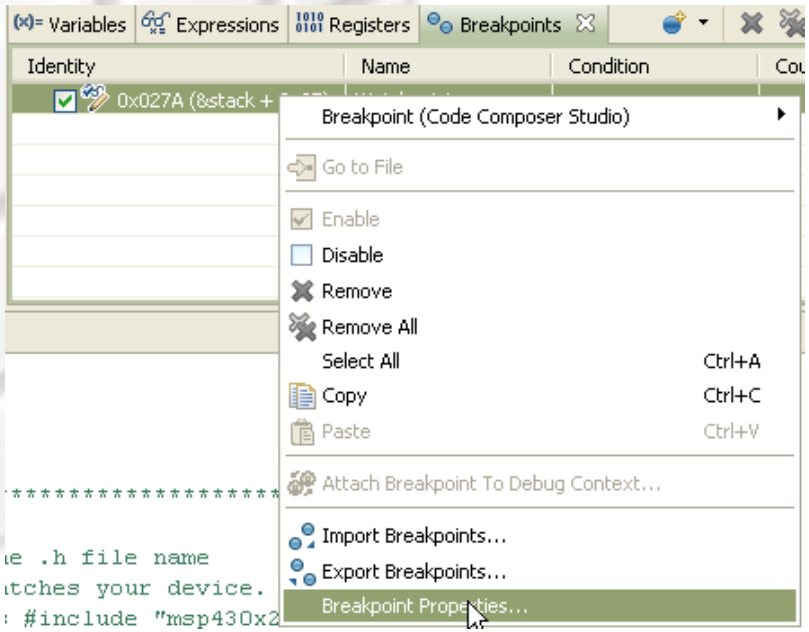
- **Press the ‘run’ button to run the program**
  - LED1 on the LaunchPad should now be blinking



Debugging: Using Watchpoints

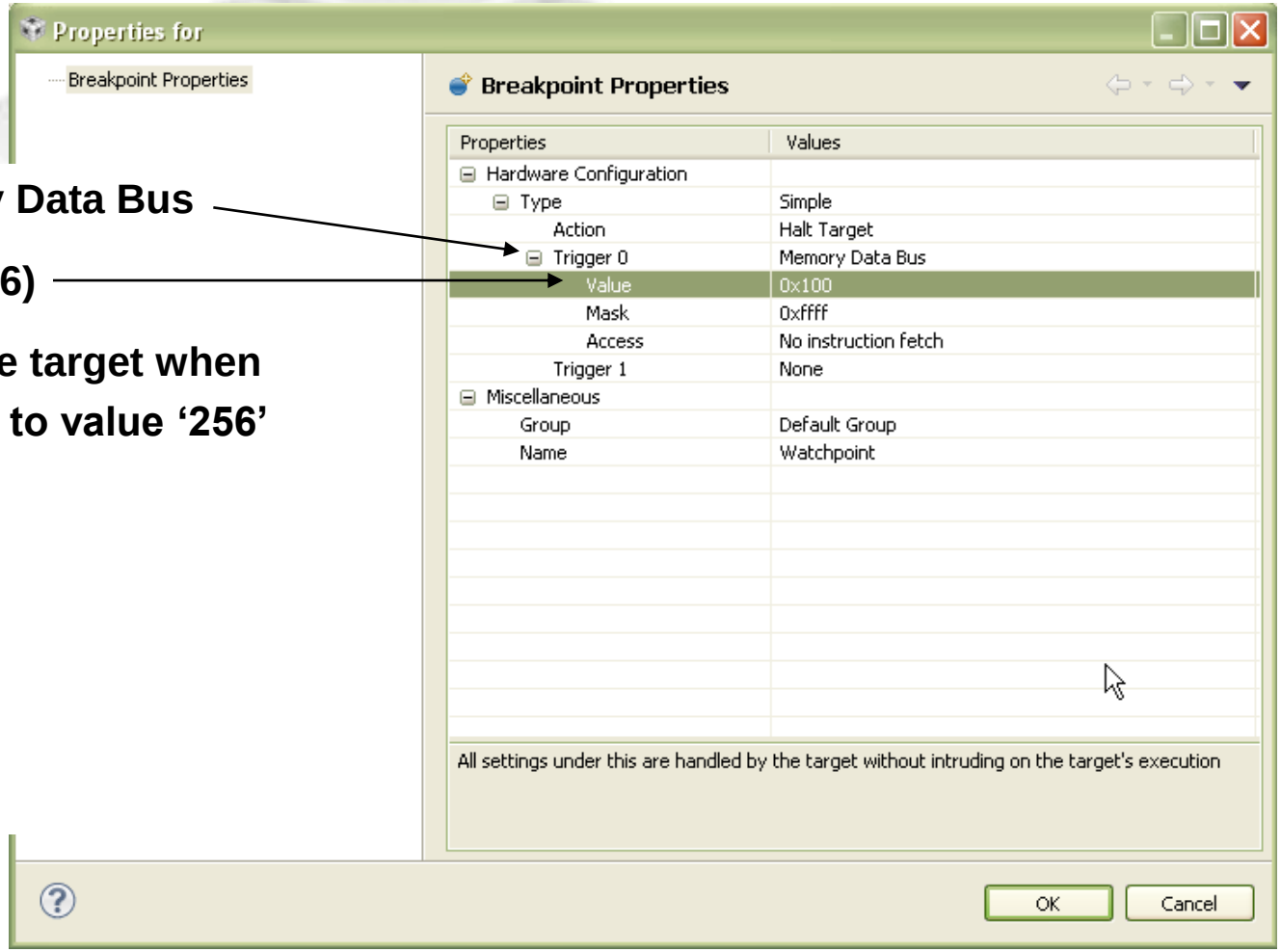
- **Press the halt\suspend button to halt the running program**
  - The code should stop somewhere inside the for loop
- **Open the 'Breakpoints' view**
  - 'View -> Breakpoints'
- **Create a new Watchpoint and specify the variable 'i' as the location**
- **Right-click on the watchpoint and select properties**





Debugging: Using Watchpoints

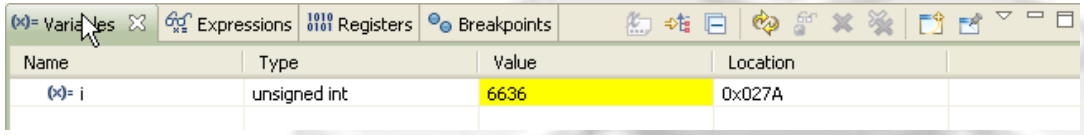
- **Set the Properties as shown below:**



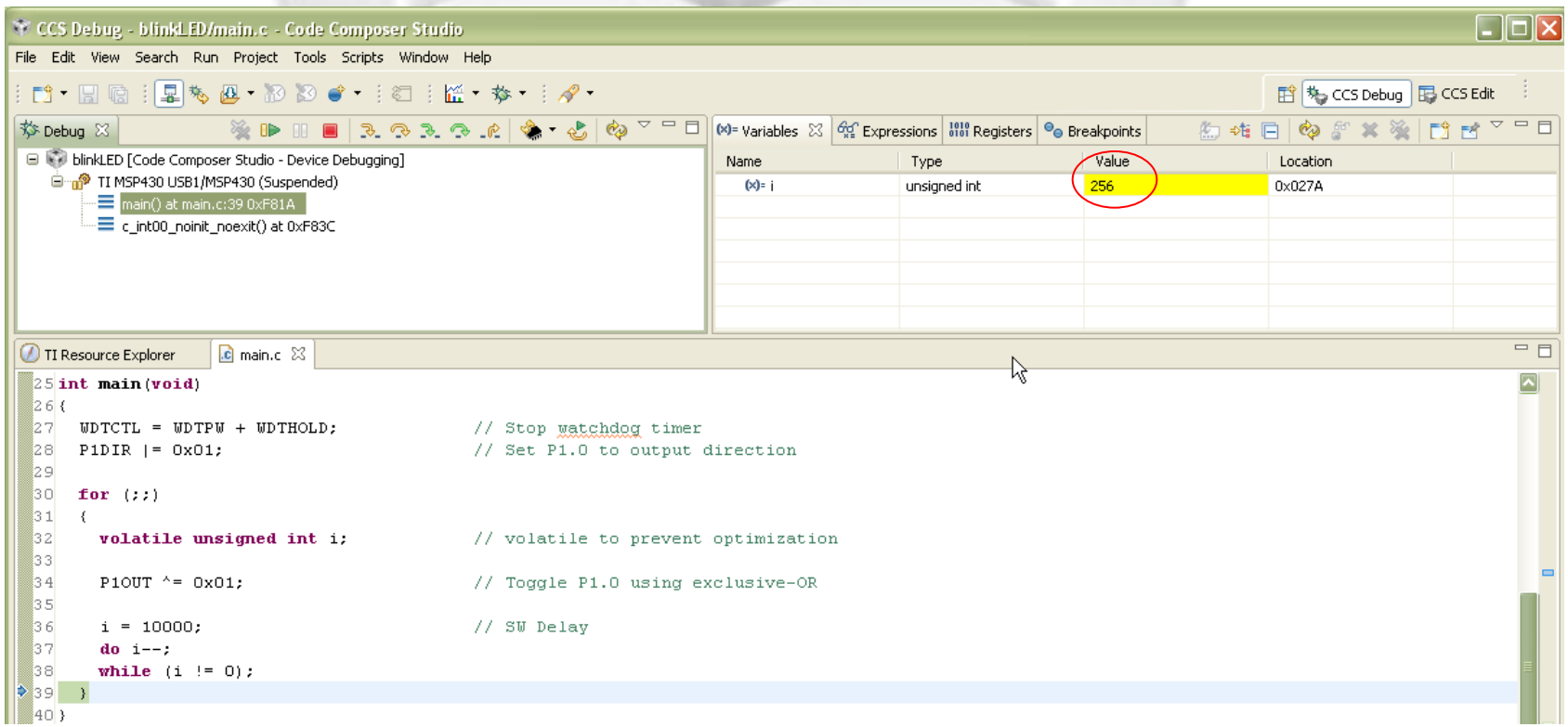
- **Trigger: Memory Data Bus**
- **Value: 0x100 (256)**
- **Basically halt the target when variable ‘i’ is set to value ‘256’**
- **Click OK**

Debugging: Using Watchpoints

- Click on the ‘Variables’ view tab to bring it to the front

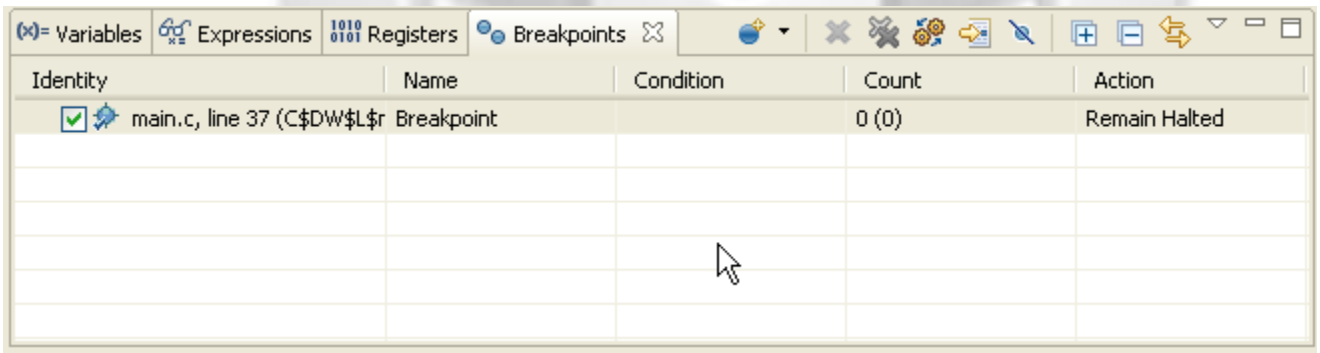


- Run the target again. Execution will automatically halt when ‘i’ is at ‘256’



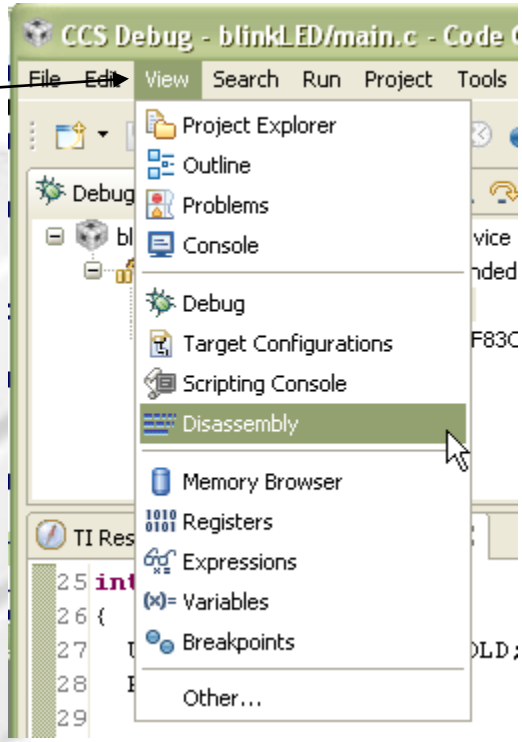
View: Breakpoints

- **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)



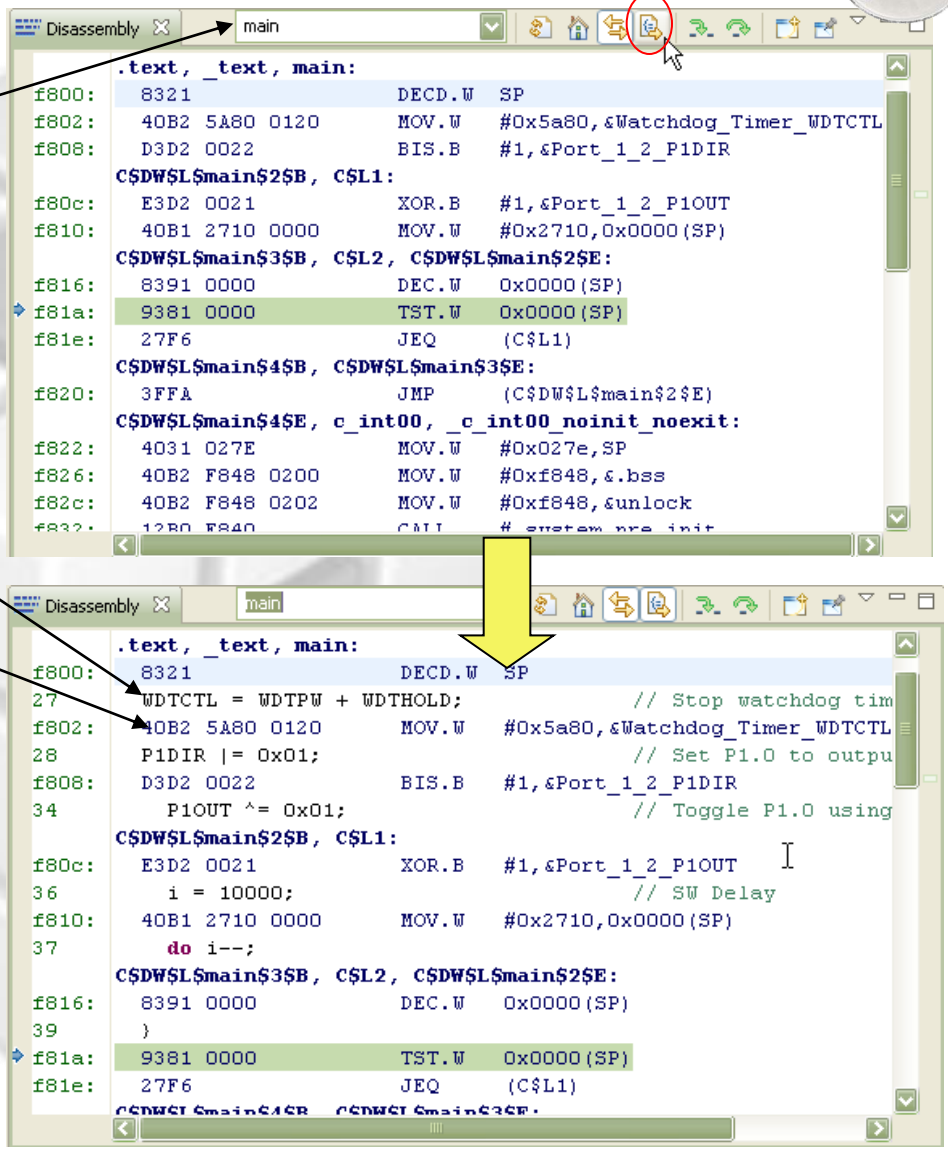
More Debugging

- **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 MSP430 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 interleaved source with the disassembly



Debugger Options

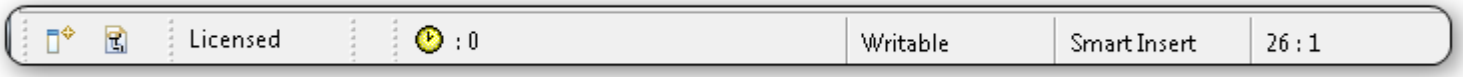
- **Many debugging features can be enabled/disabled from the Debugger Options**
- **During a debug session in the 'CCS Debug' perspective:**
  - ' Tools->Debugger Options->Generic Debugger Options'
- **Configure a variety of debug options like enable/disable:**
  - auto-run to main
  - auto-connect to 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**

Remove the Watchpoint

- **Clear breakpoints**
  - We need to clear the watch point or it will get set for our next lab as well as we are using the same target
  - Go to the Breakpoints view
  - Click on the button to “Remove all Breakpoints”, say yes to the prompt
- **Restart execution**
  - Go to the Debug View
  - Click on the Restart button 
  - The program counter should be back at main

Counting Cycles

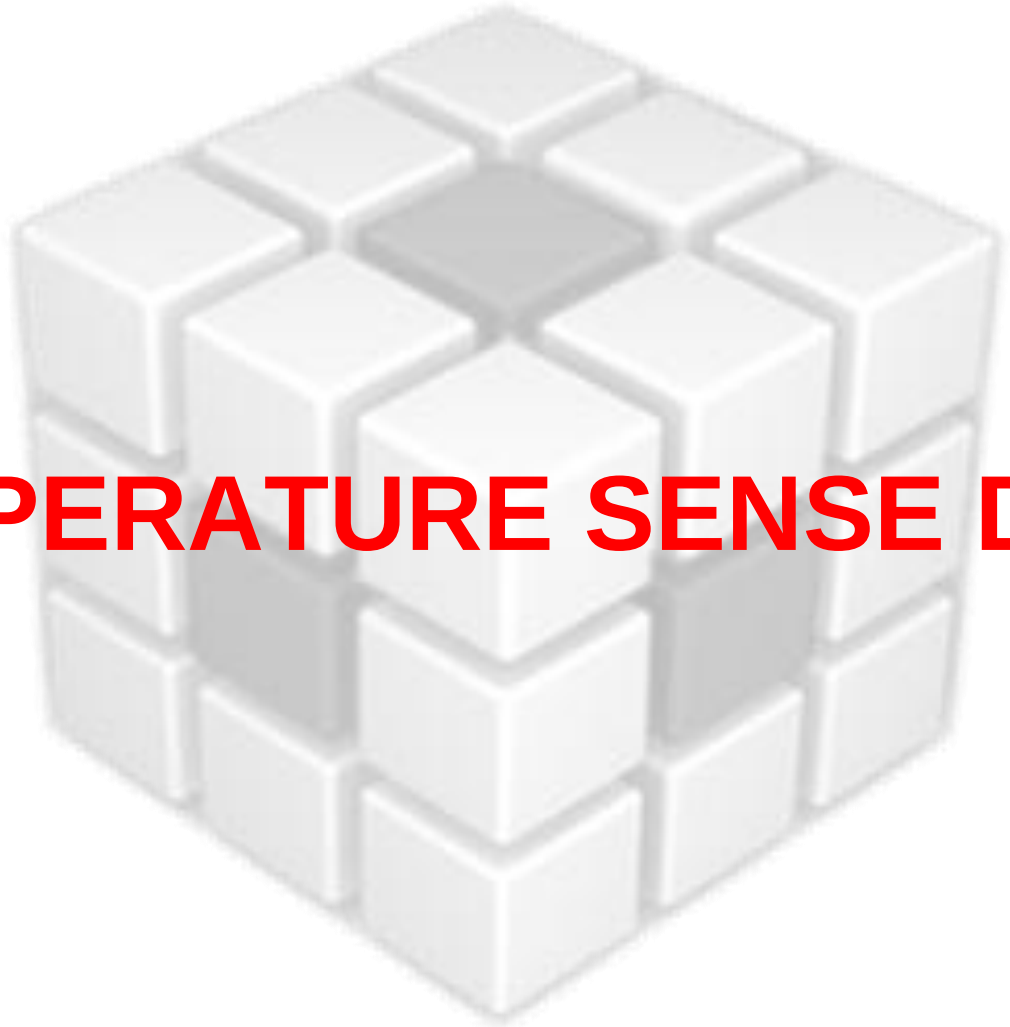
- **The profile clock**
  - Available on most devices and can be used to count cycles
  - On some targets it can be used to count other events as well
- **Enable the Clock**
  - Run -> Clock -> Enable
  - The clock will now be displayed on the status bar



- **Place a breakpoint and run to it**
  - Double click in the selection margin on line 39 (“}” for the for loop) to add a breakpoint
  - Click the run button to run to this point
  - Clock should now show 24 cycles 

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



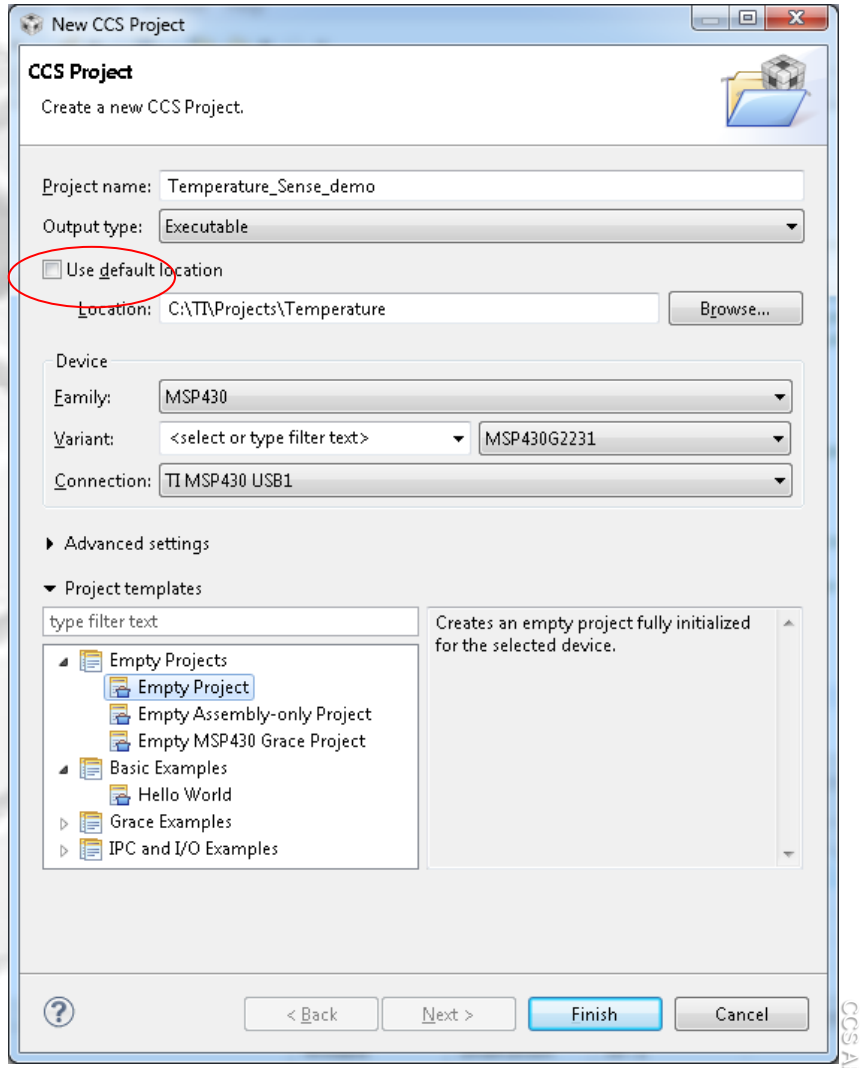
TEMPERATURE SENSE DEMO

Temperature Sense Demo: Briefing

- **Key Objectives**
  - Create and build the Temperature Sense Demo
  - Start a debug session and load/flash the program on the LaunchPad
  - Run the program to start Temperature Sense Demo
- **Tools and Concepts Covered**
  - Adding source files
  - “Focus” concept
  - Loading Symbols
  - Changing Build Options
  - Changing Compiler Version

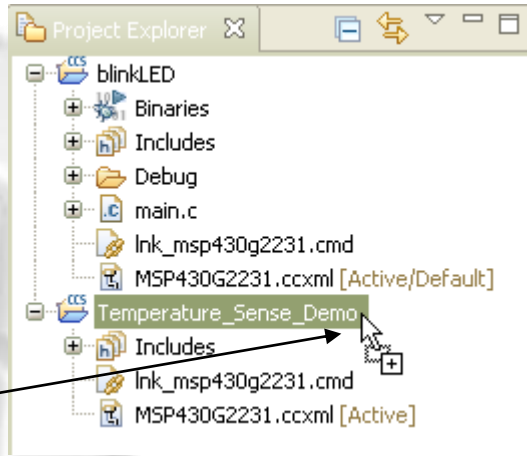
Create a New Project

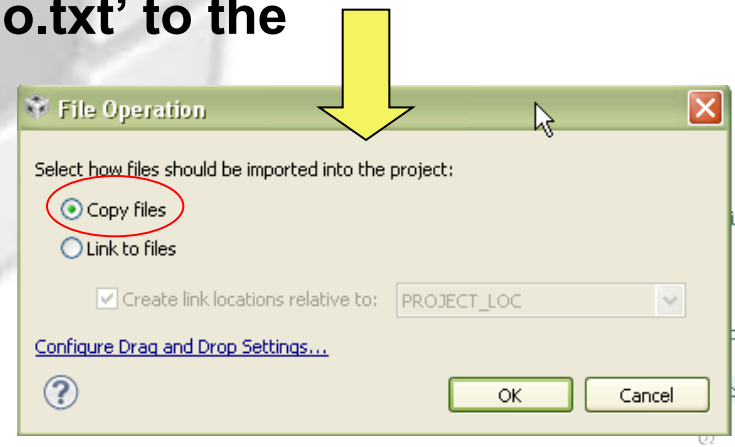
- **Launch ‘New CCS Project’ Wizard**
  - In ‘CCS Edit’ perspective, Project -> ‘New CCS Project’
- **Project Location**
  - Note that this time we will be creating this project outside the workspace
  - Uncheck the box to use default location
  - Specify **C:\TI\Projects\Temperature**
- **Select ‘Finish’ when done**



Add Temperature Sensor Source Code

- **Remove the generated ‘main.c’ file from the project**
  - Select it in the project explorer and press delete
- **Open Windows Explorer and browse to:**
  - C:\TI\LaunchPad\Temperature\Sensor
- **Drag and drop ‘Temperature_Sense_Demo.txt’ to the ‘Temperature_Sense_Demo’ project**
  - Make sure the file is dragged to the ‘Temperature_Sense_Demo’ project
- **In the dialog popup, select the option to ‘Copy Files’ and hit ‘OK’**

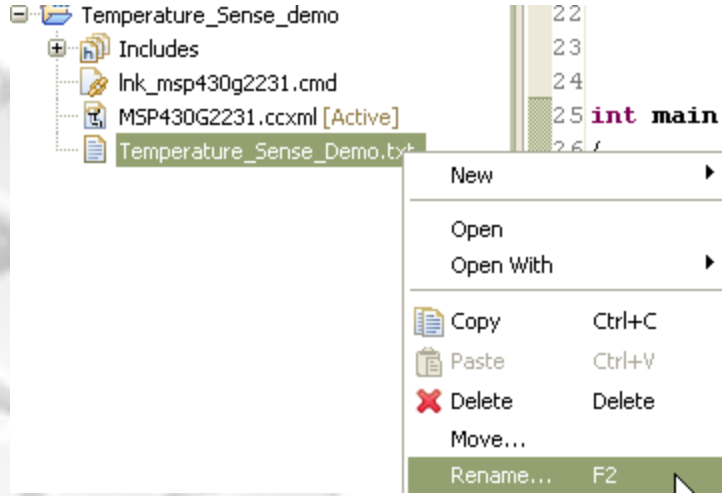


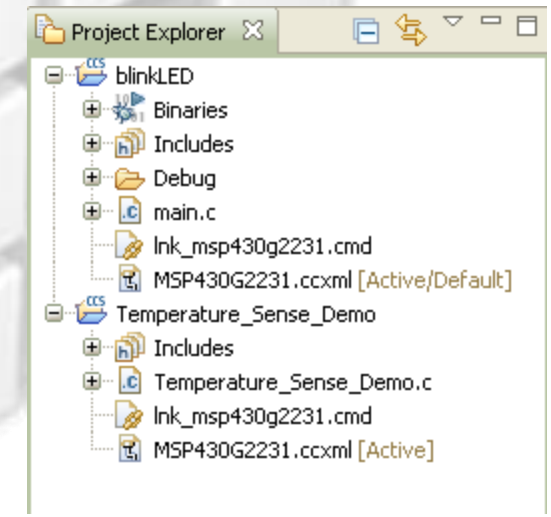


Add Temperature Sensor Source Code

- **Right-click on the ‘Temperature_Sense_Demo.txt’ file in the Project Explorer and select the option to ‘Rename..’ the file**
  - Rename the file so that the ‘.txt’ extension is renamed to ‘.c’

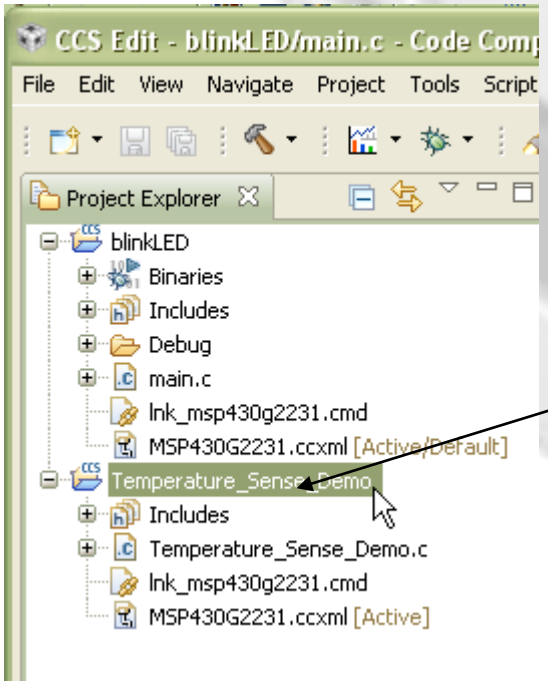
- **Project is ready to build**





Eclipse Concept: Focus

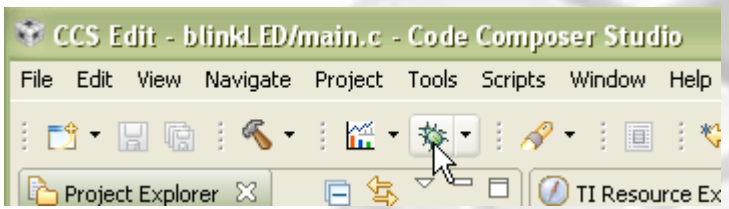
- **Focus refers to the highlighted portion of the workbench**
  - Can be an editor, a view, a project, etc.
- **This concept is important since several operations inside Eclipse are tied to the element in focus**
  - Project build errors, console, menu and toolbar options, etc.



'Temperature_Sense_Demo' project is in 'Focus' since it has been selected. So pressing the 'Debug' button will build the project and start the debugger for the 'Temperatrue_Sense_Demo' project

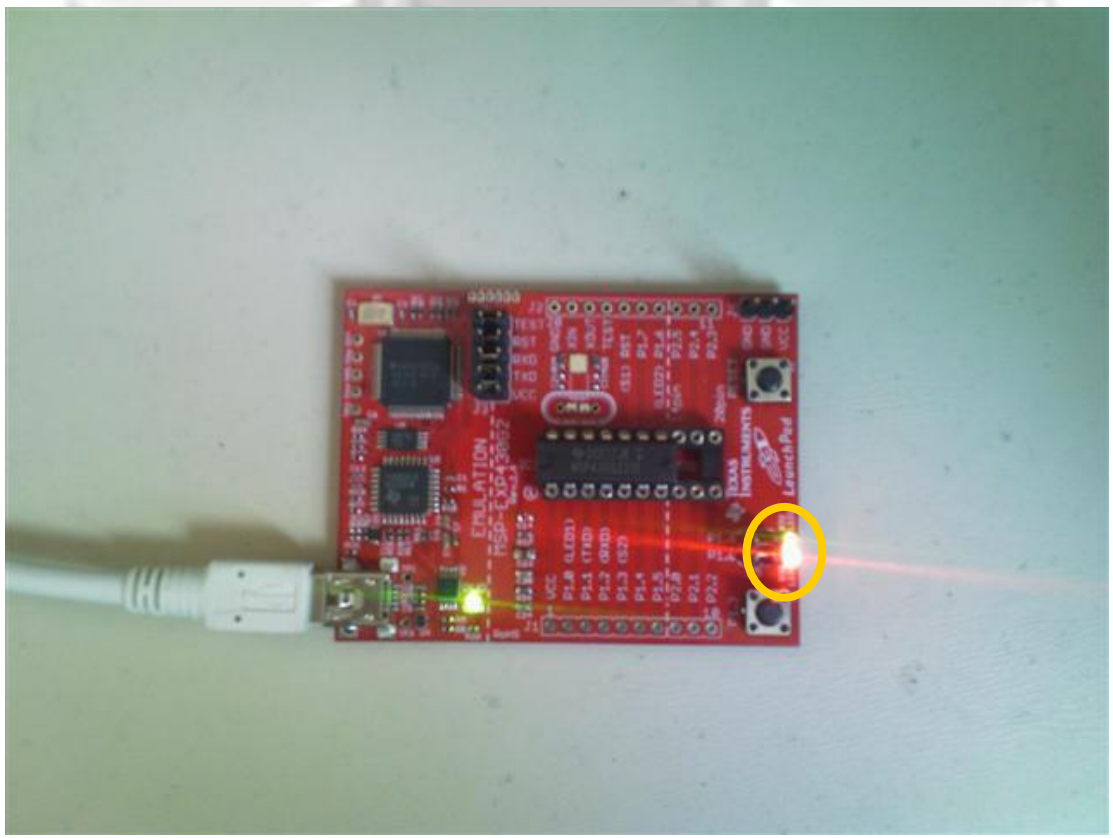
Build and Load/Flash the Program

- **Make sure the 'Temperature_Sense_Demo' project is in 'Focus'. Then use the 'Debug' button**



Temperature Sense Demo

- **Press the 'run' button to run the program**
  - LED1 (red) and LED2 (green) on the LaunchPad should now alternate blinking

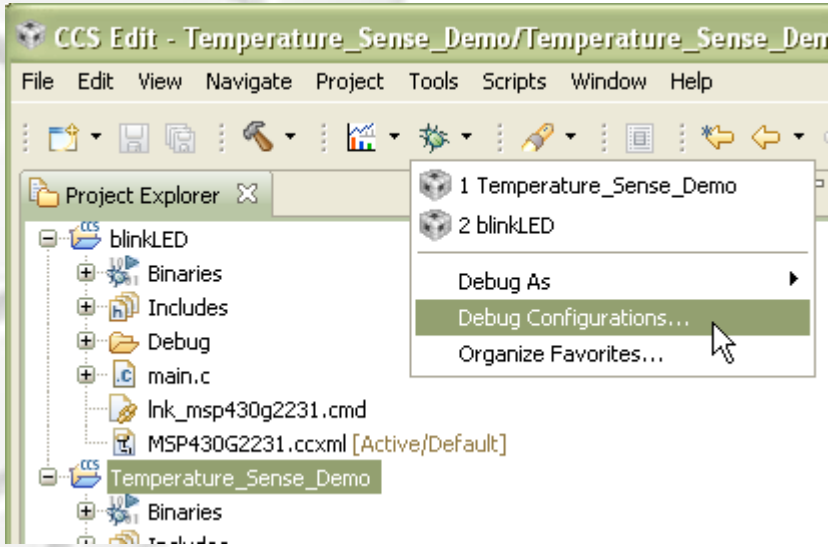


Temperature Sense Demo: Debugging

- **Press the halt\suspend button to halt the running program**
  - The code should stop in the PreApplicationMode() function.
- **Step-into the code once and it will enter the timer ISR for toggling the LEDs (ta1_isr)**
- **Step-over a few more times and notice that the red and green LEDs alternate on and off**
- **When you are done terminate the debug session**

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..'**

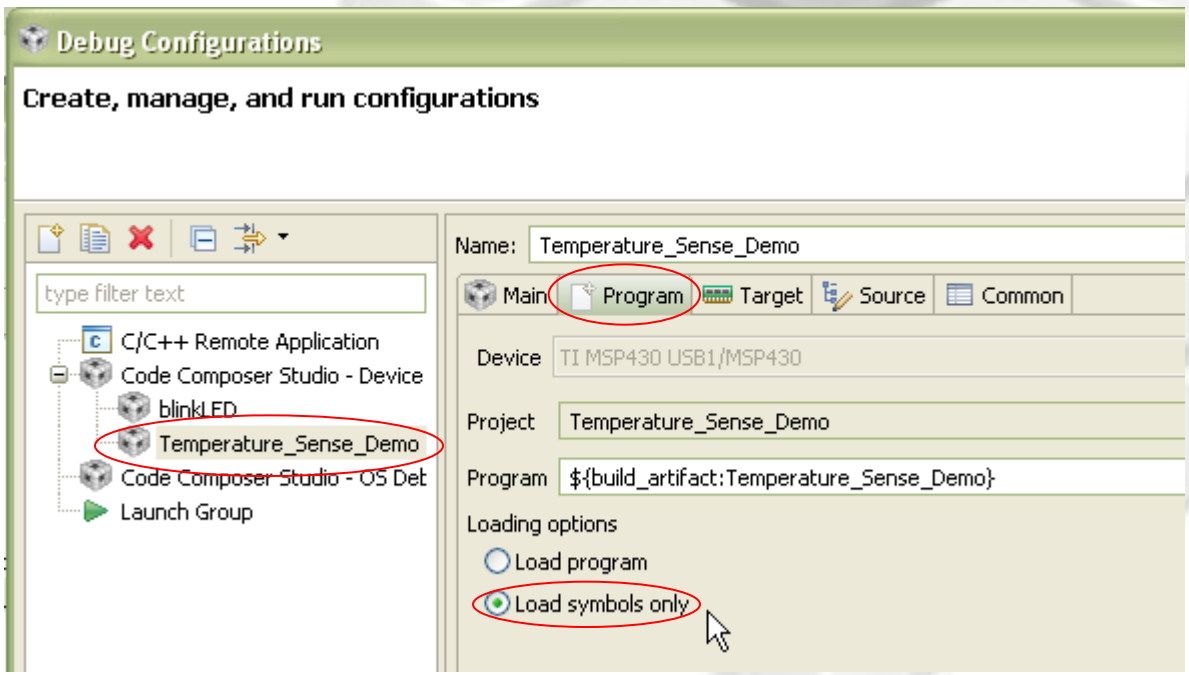


eclipse

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

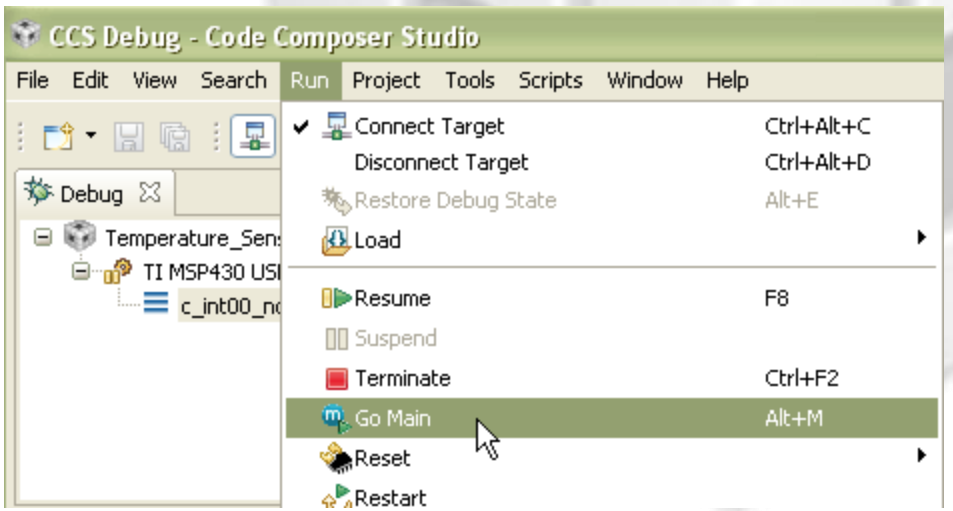
- Select ‘Temperature_Sense_Demo’ in the left panel and the ‘Program’ tab in the right panel
- Under ‘Loading options’, select ‘Load symbols only’



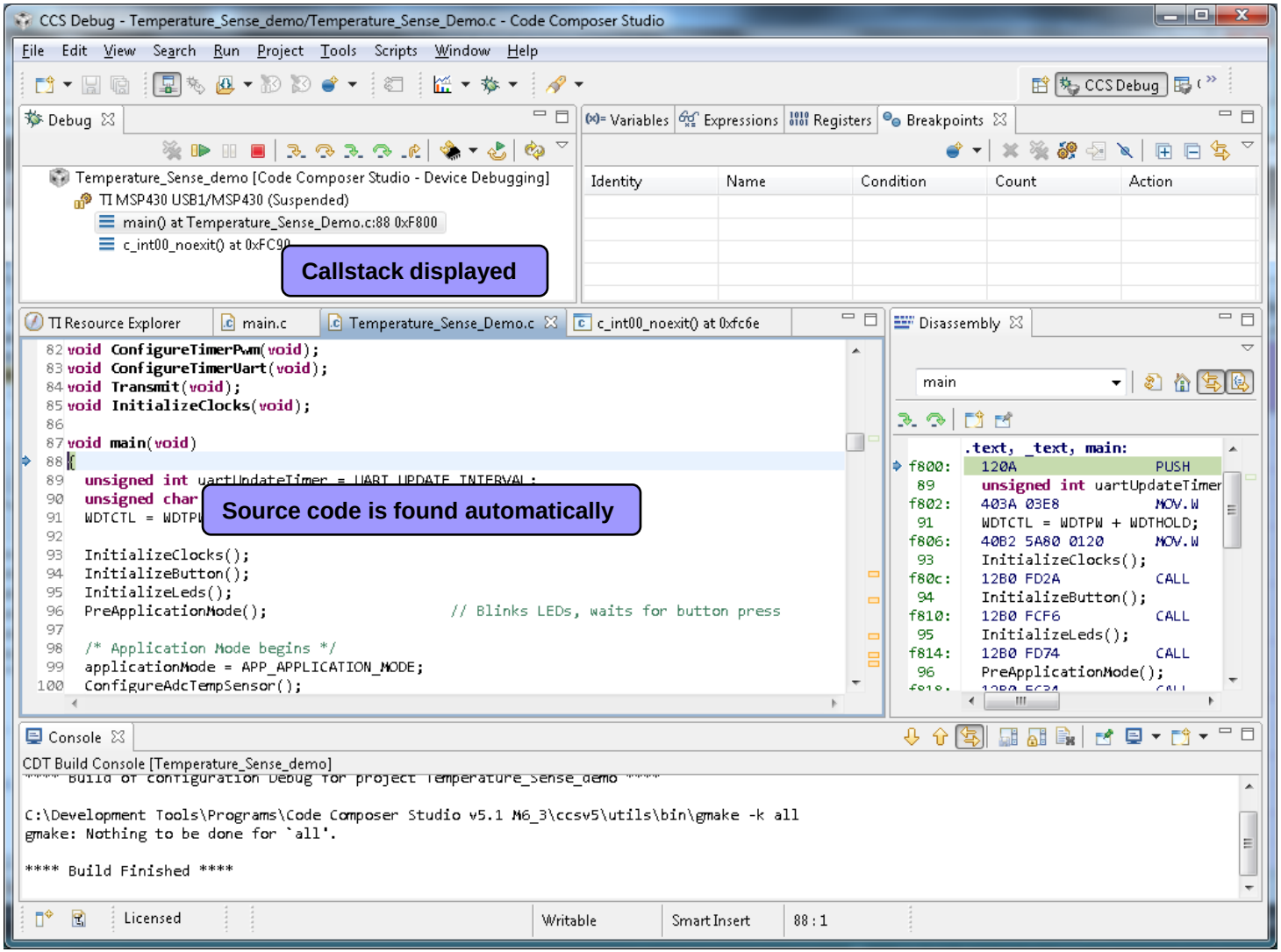
- Then select ‘Apply’ and then ‘Debug’

Loading Symbols for Flashed 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/flashed 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 Flashed Program



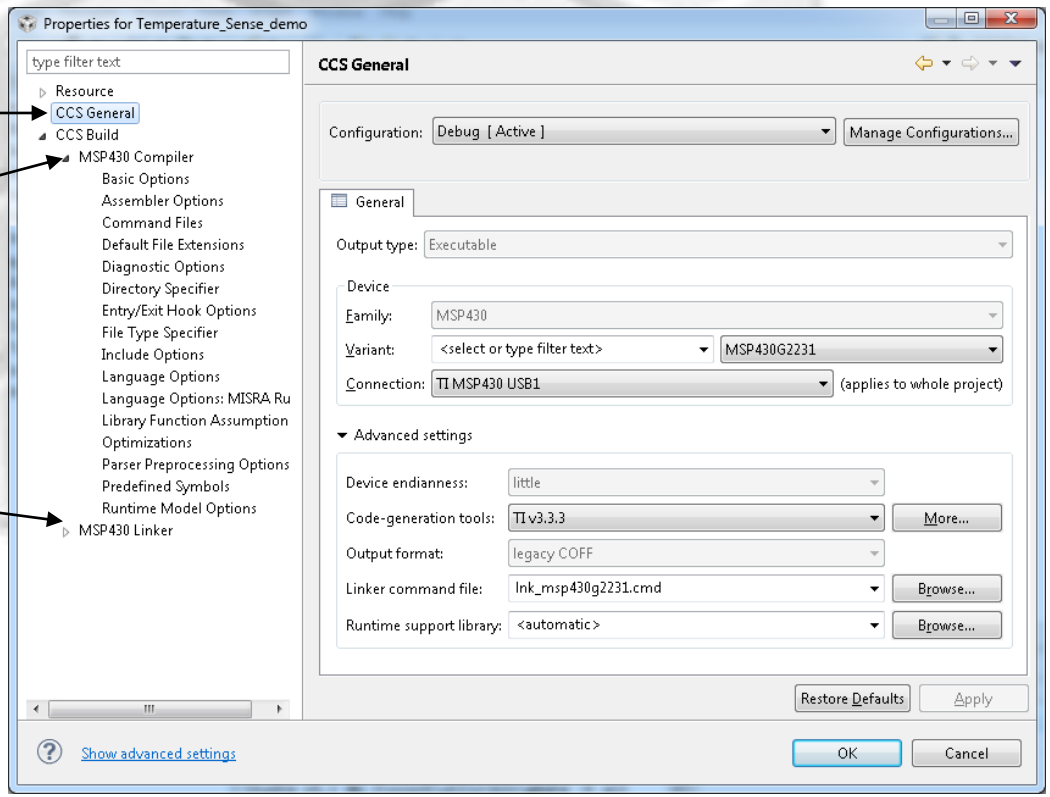
Changing Build Options

- Terminate the debug session
- Right click on the ‘Temperature_Sense_demo” project and select ‘Properties’

Device and high level settings

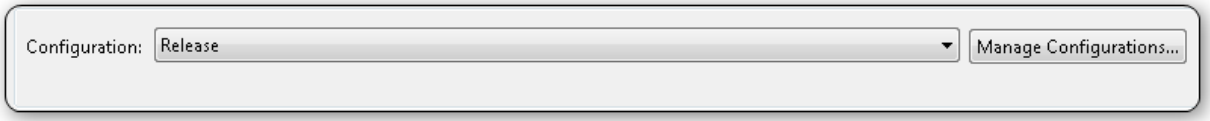
Compiler Options

Linker Options

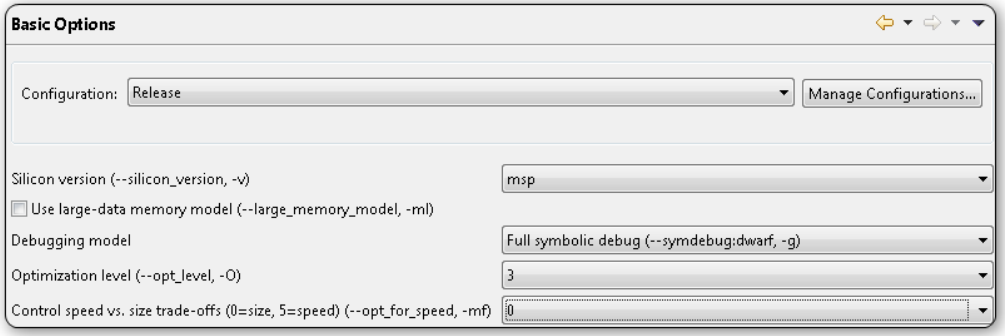


Changing Build Options

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



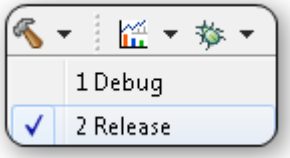
- **Change the optimization settings**
  - Go to the Basic Option Group
  - Change the optimization level to 3
  - Change speed vs size to 0 (optimize for size)



- **Click OK**

Changing Build Options

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

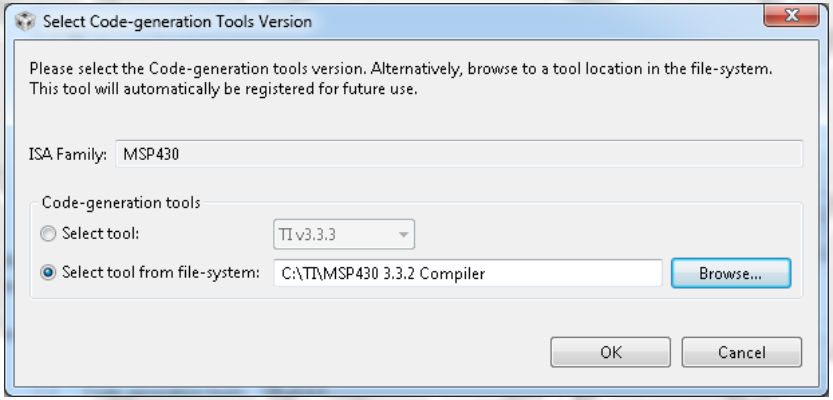


Compiler Versions

- **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 CCS General
- Click on the More button beside the Compiler version “TI 3.3.3”
- 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 v3.3.2 is now specified
- Build your project

LaunchPad

- **Want to learn more about LaunchPad?**
  - Check out: http://processors.wiki.ti.com/index.php/MSP430_LaunchPad_%28MSP-EXP430G2%29

Questions?