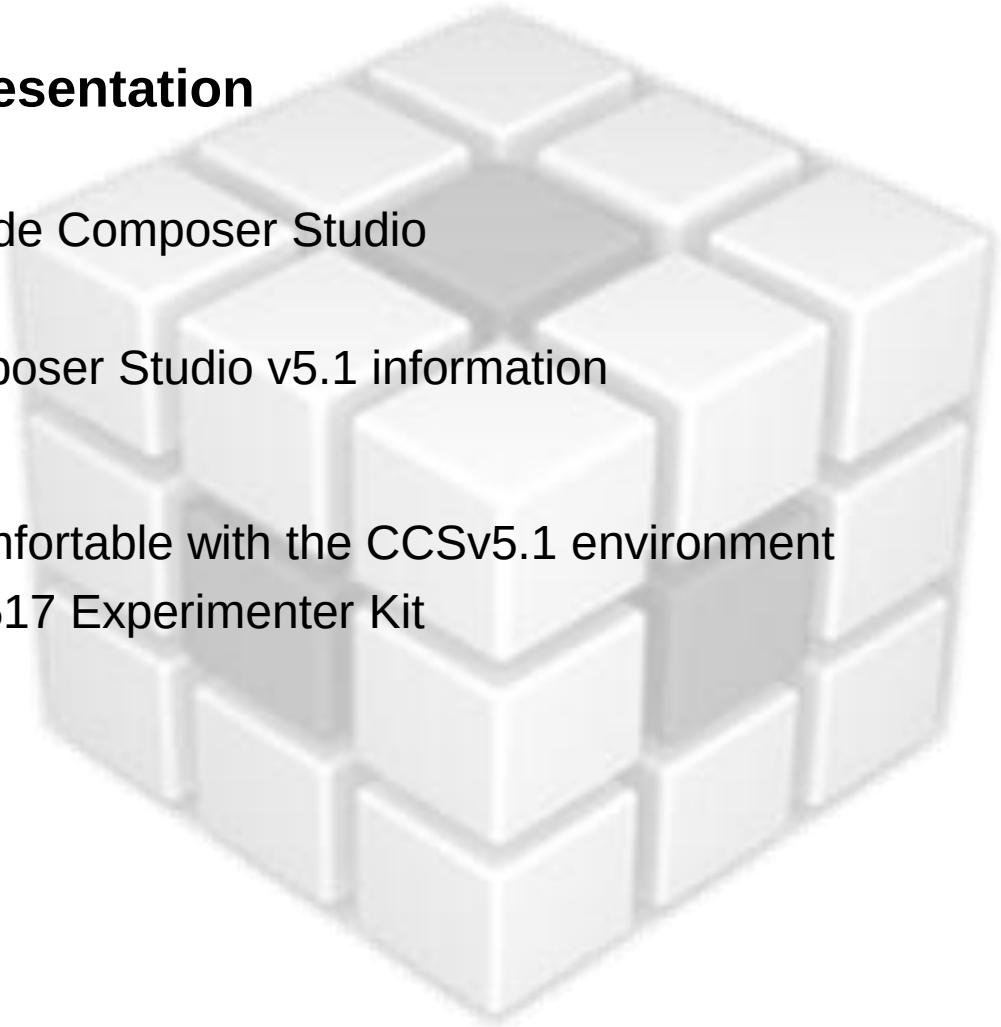
Code Composer Studio Advanced Workshop

CCS APPS

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 AM3517 Experimenter Kit

Survey

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

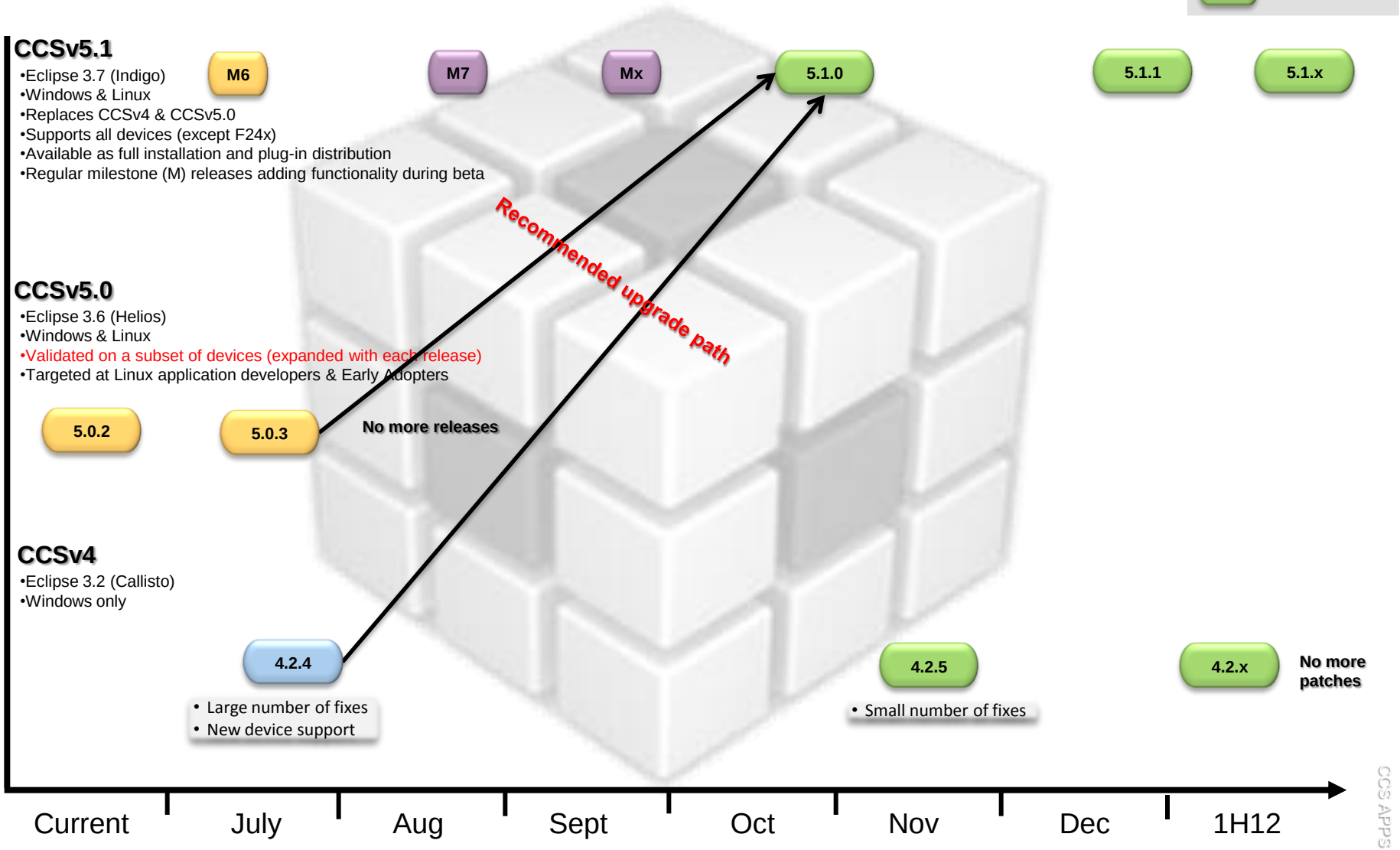
What is Code Composer Studio?

- **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

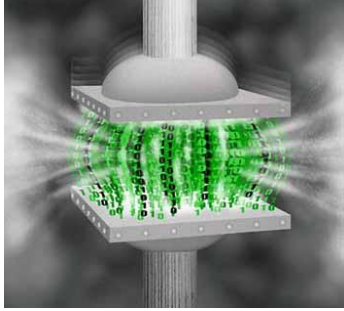
What's new in Eclipse?

- **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

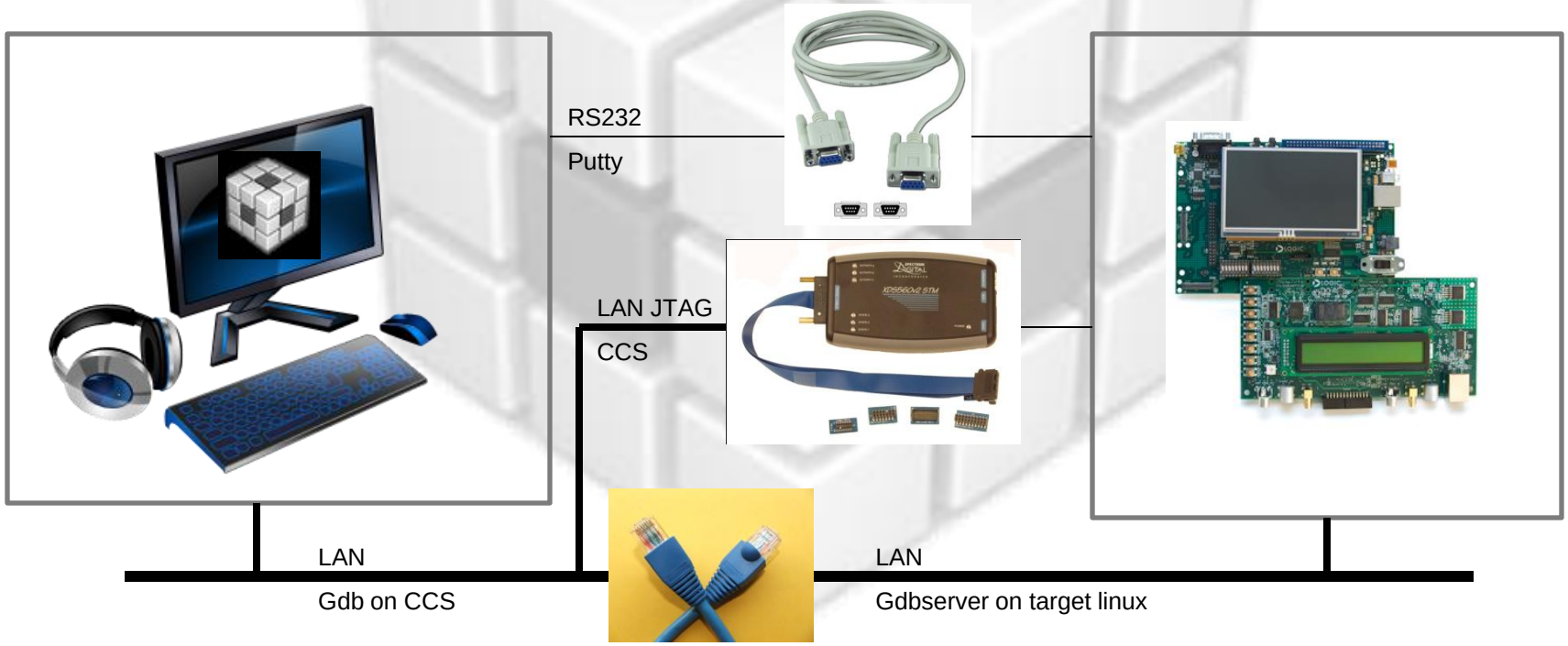
- **Simple Mode**
  - By default CCS will open in simple/basic mode
  - Simplified user interface with far fewer menu items, toolbar buttons
  - TI supplied Edit and Debug Perspectives
  - Simplified build options
- **Advanced Mode**
  - Use default Eclipse perspectives
  - Very similar to what exists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

GETTING STARTED WITH CCSV5 AND AM3517

What is the AM3517?

- **High-performance ARM Cortex-A8 microprocessor with speeds up to 600 MHz**
- **3D graphics acceleration while also supporting numerous peripherals, including DDR2, CAN, EMAC, and USB OTG PHY that are well suited for industrial applications**
- **The processor can support other applications, including:**
  - Single Board Computers
  - Home and Industrial automation
  - Digital Signage

What is the AM3517 Experimenter Kit?

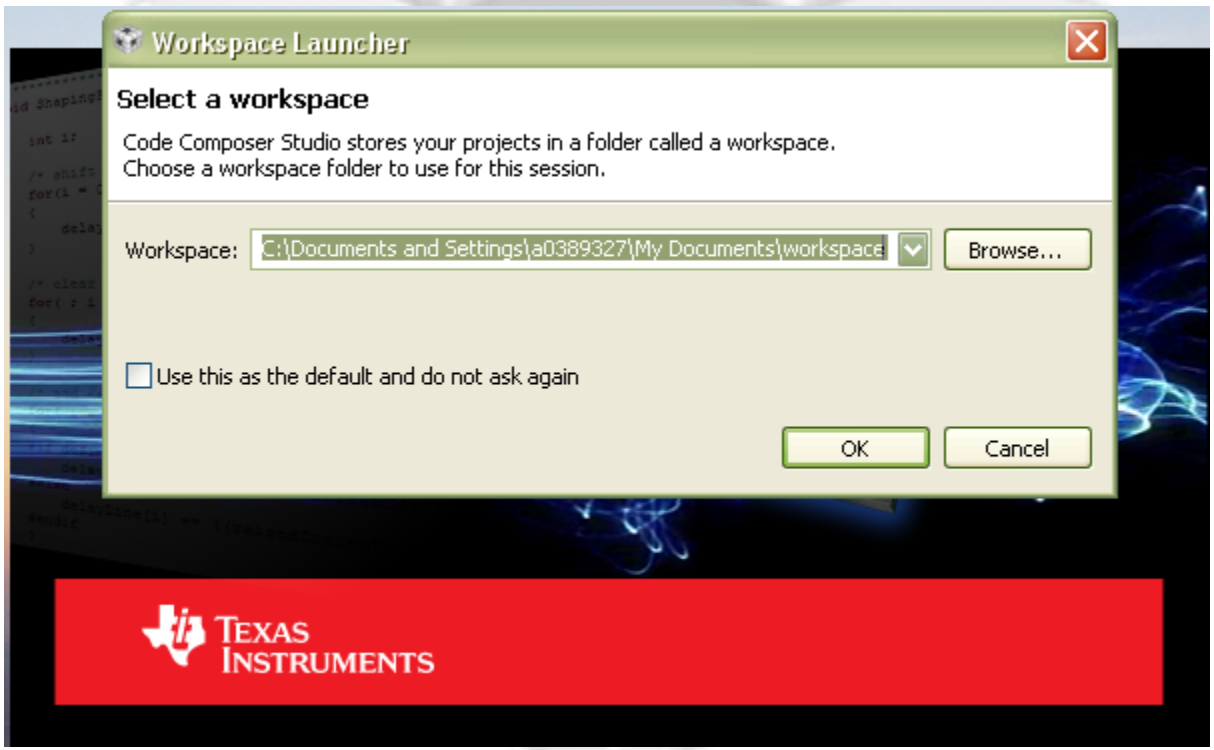
- **Low-cost application development kit from Logic PD for evaluating the functionality of AM3517 applications processor and Logic PD's System on Module (SOM)**

Workshop Instructions

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

Workspace

- **Launch CCS and select a workspace folder**
  - Add _2 onto the end of the folder name to create a new workspace



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

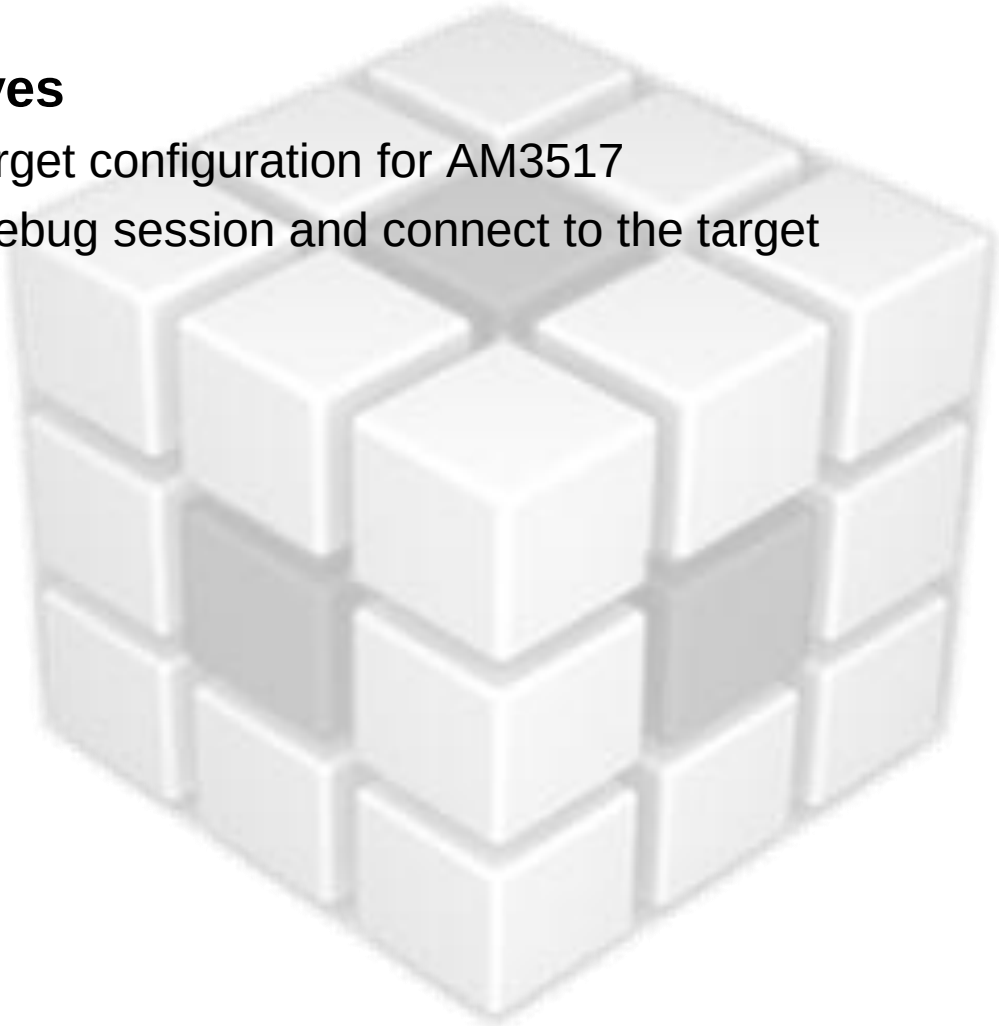
View: Resource Explorer - Welcome

- 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

WORKING WITH TARGET CONFIGURATIONS

Launch a Debug Session: Briefing

- **Key Objectives**
  - Create a target configuration for AM3517
  - Launch a debug session and connect to the target



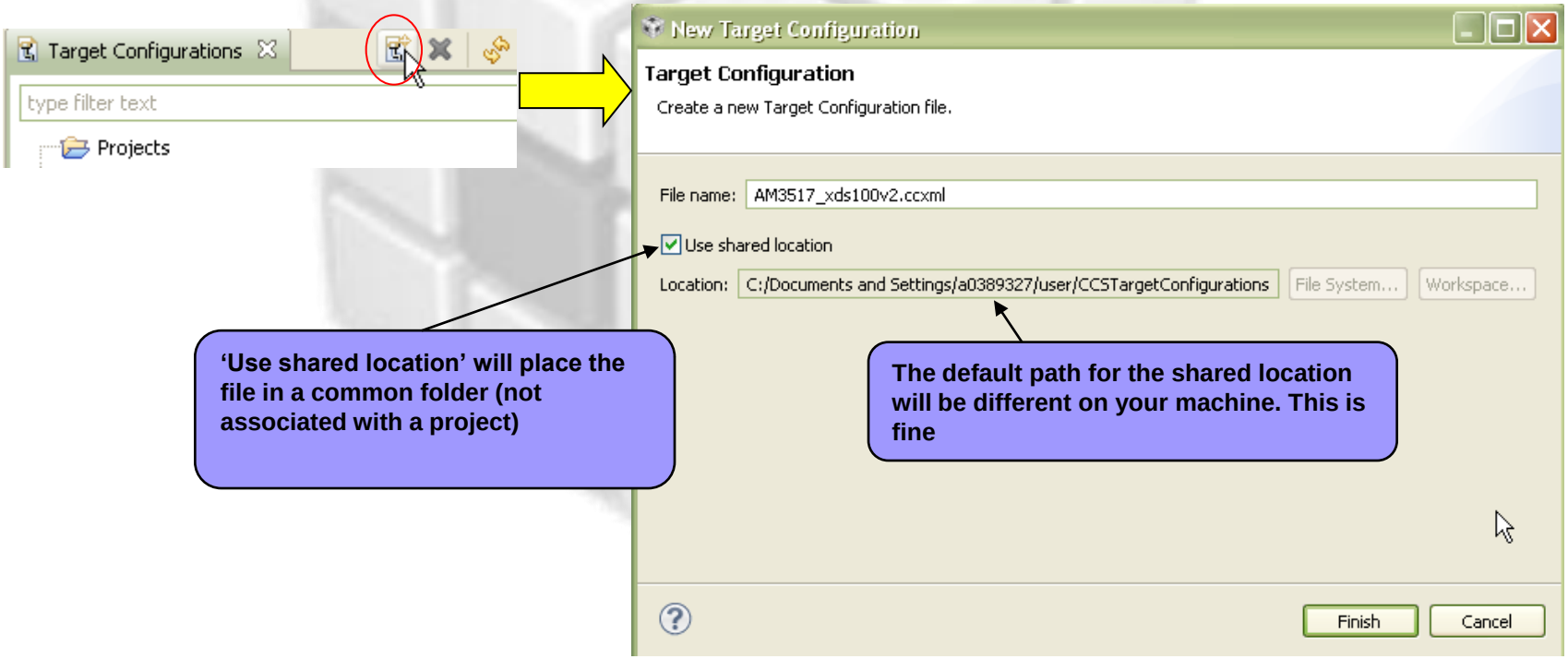
Target Configuration Files - Basics

- **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

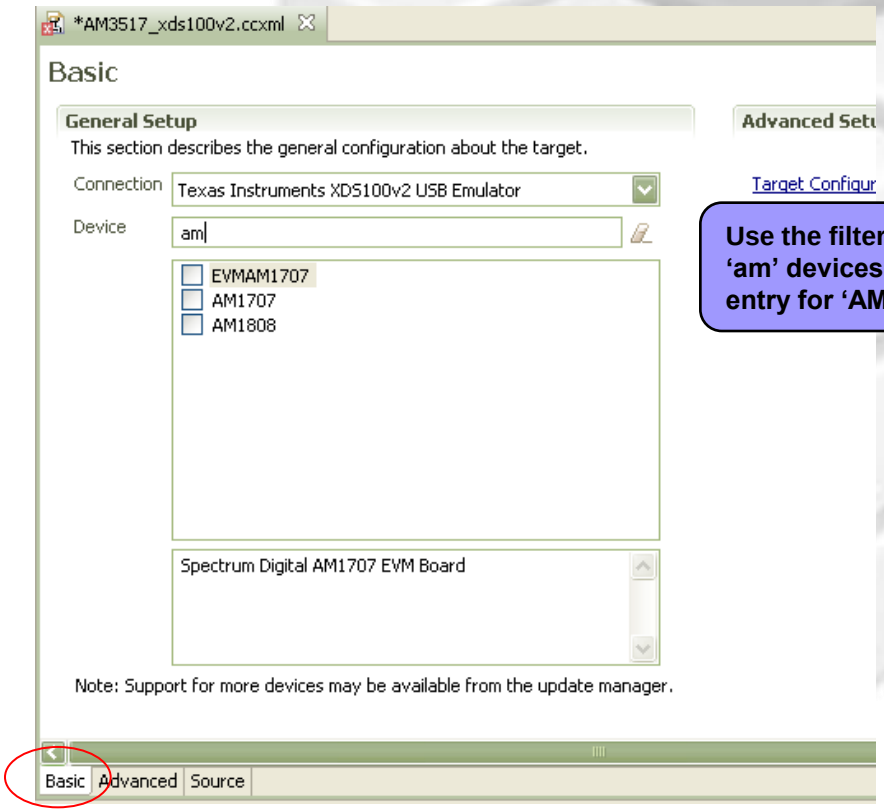
- **Open the 'Target Configuration' view**
  - View -> Target Configurations

- **Create a new configuration:**



Creating a Target Configuration

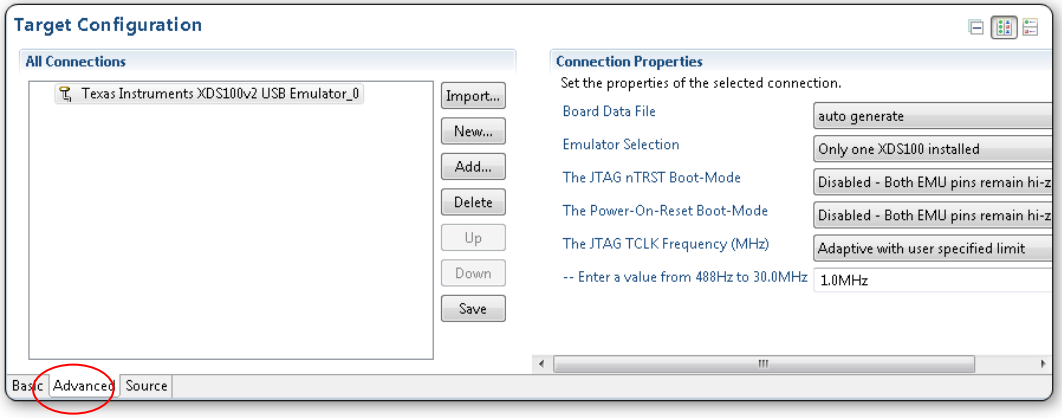
- **In the 'Basic' tab, see if there is an option for:**
  - Connection: XDS100v2
  - Device: AM3517



Use the filter field to search for all 'am' devices. Note how there is no entry for 'AM3517'

Creating a Target Configuration

- **AM3517 is NOT yet in the list of supported Devices**
  - It will be in later releases
  - Thus we will have to build one manually
- **Click on the 'Advanced' tab**
- **Click on the XDS100v2 emulator on the left**
  - Change the JTAG TCLK Frequency to be Adaptive with user specified limit of 1.0MHz



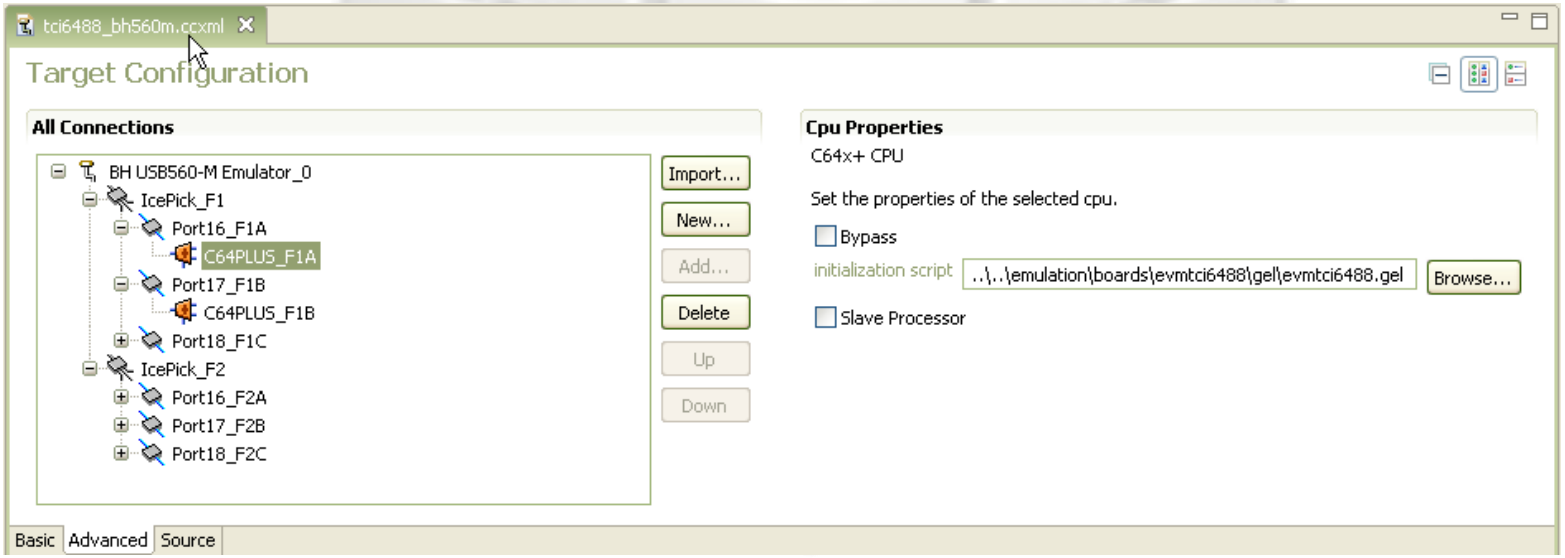
Target Configurations - Advanced

- **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

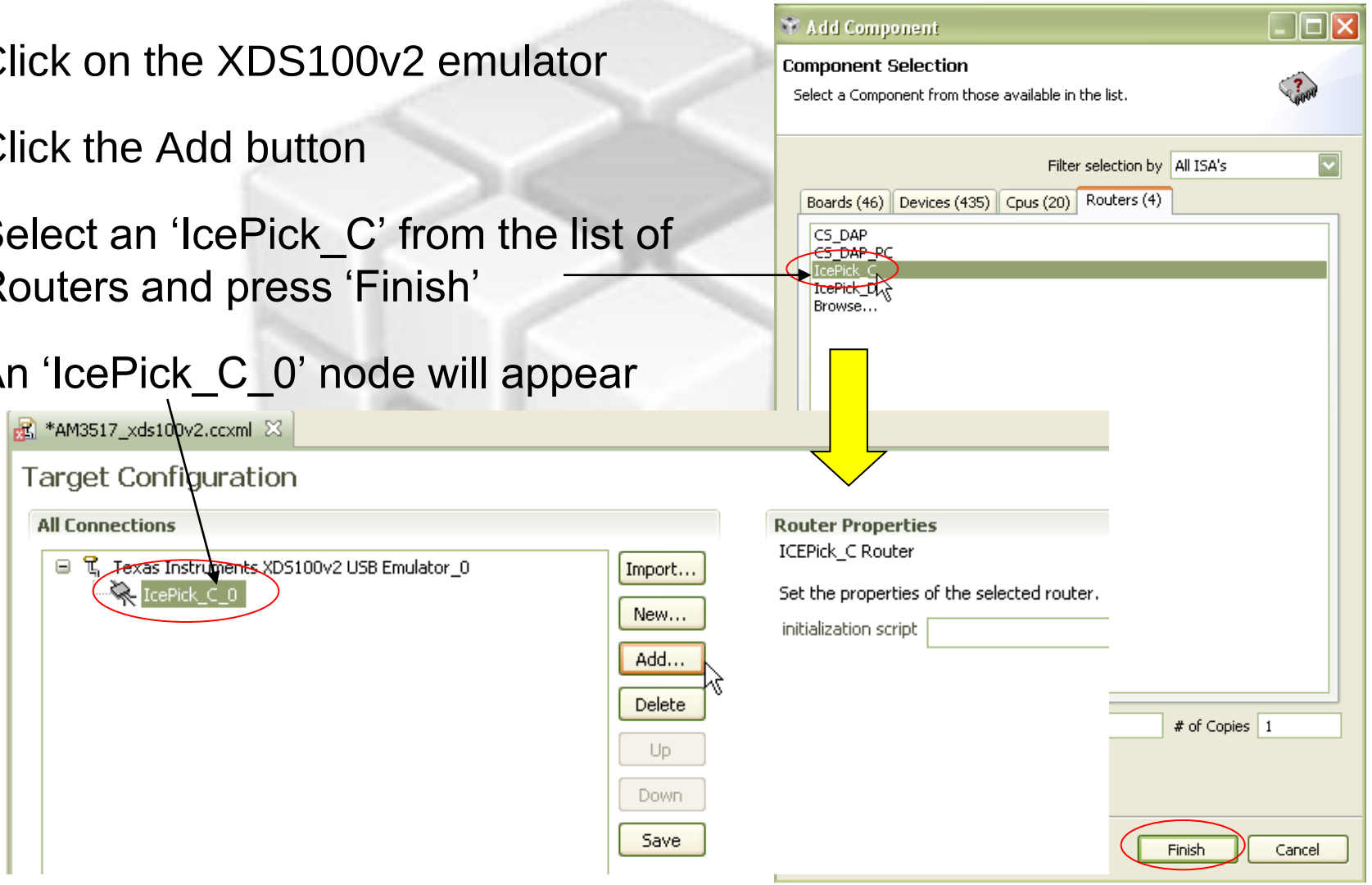
- **Adjust default properties of the target configuration:**
  - Specify initialization files (GEL startup files)
  - Specify IcePick subpath port numbers
  - Bypass CPU
  - Set JTAG TCLK Frequency
  - etc…

Similar to the CCS 3.x CCS Setup utility



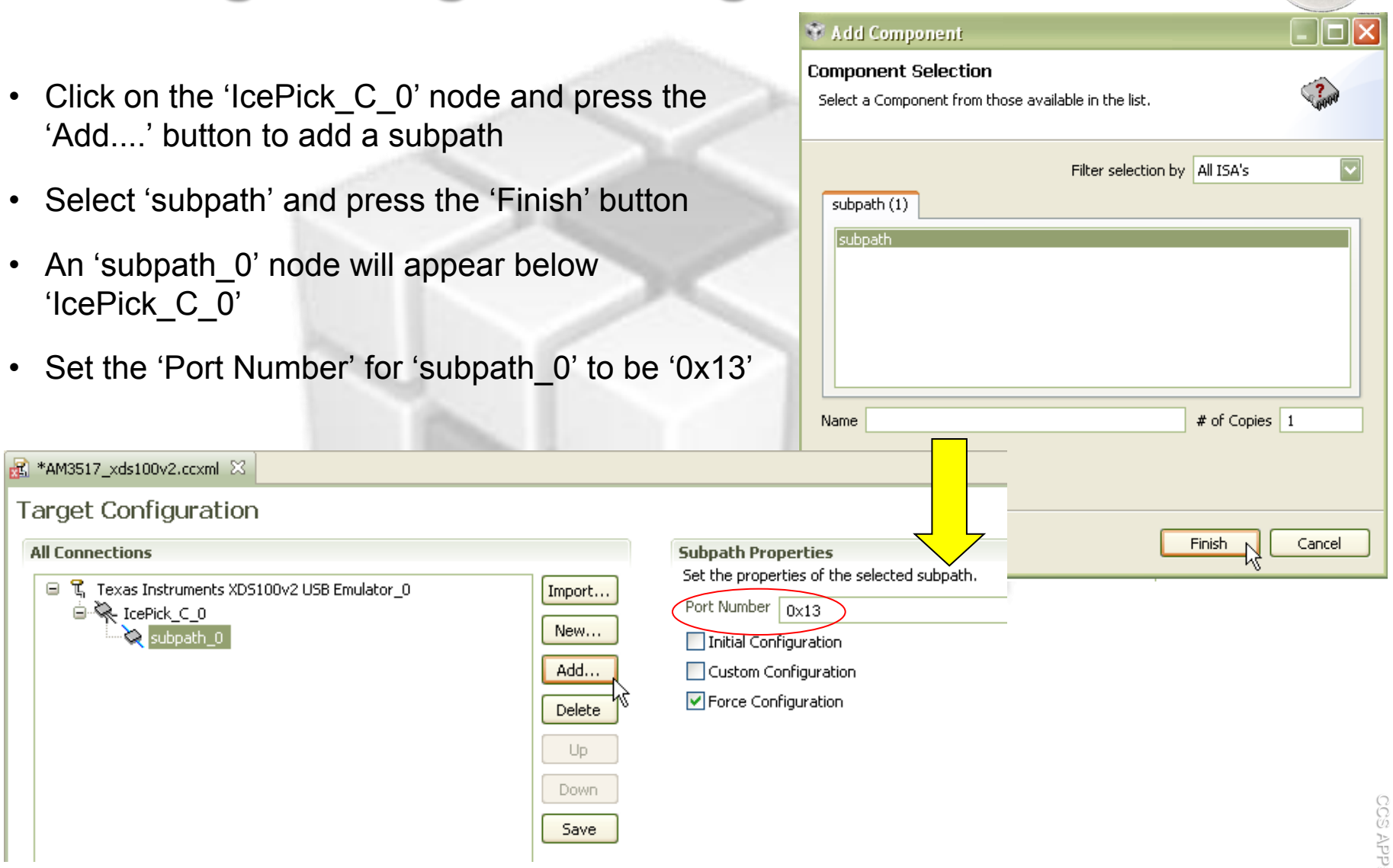
Creating a Target Configuration

- Click on the XDS100v2 emulator
- Click the Add button
- Select an ‘IcePick_C’ from the list of Routers and press ‘Finish’
- An ‘IcePick_C_0’ node will appear



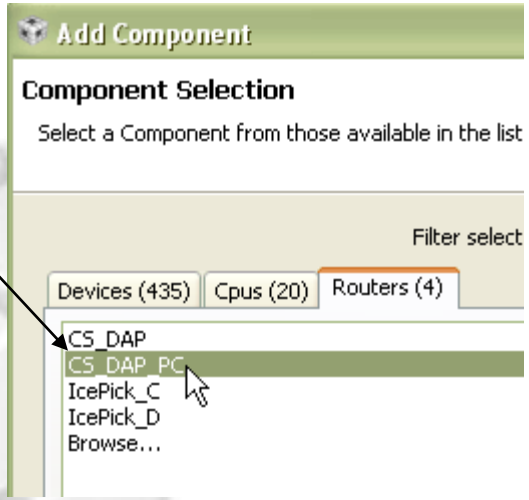
Creating a Target Configuration

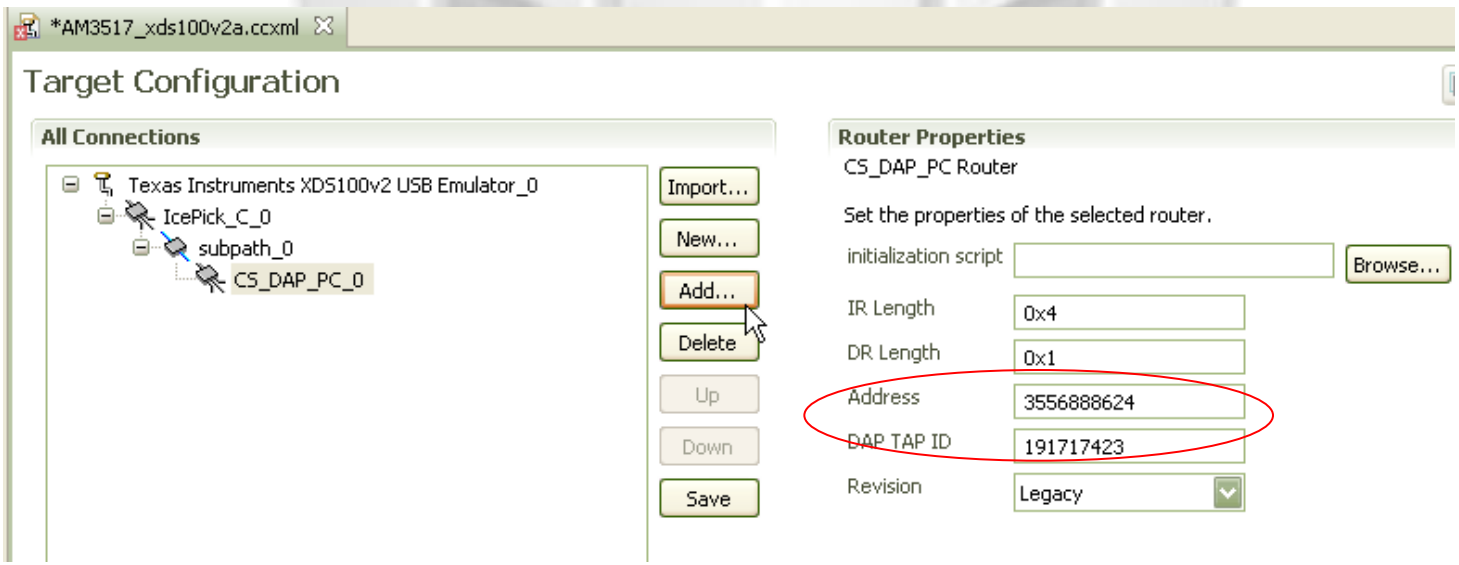
- Click on the ‘IcePick_C_0’ node and press the ‘Add....’ button to add a subpath
- Select ‘subpath’ and press the ‘Finish’ button
- An ‘subpath_0’ node will appear below ‘IcePick_C_0’
- Set the ‘Port Number’ for ‘subpath_0’ to be ‘0x13’



Creating a Target Configuration

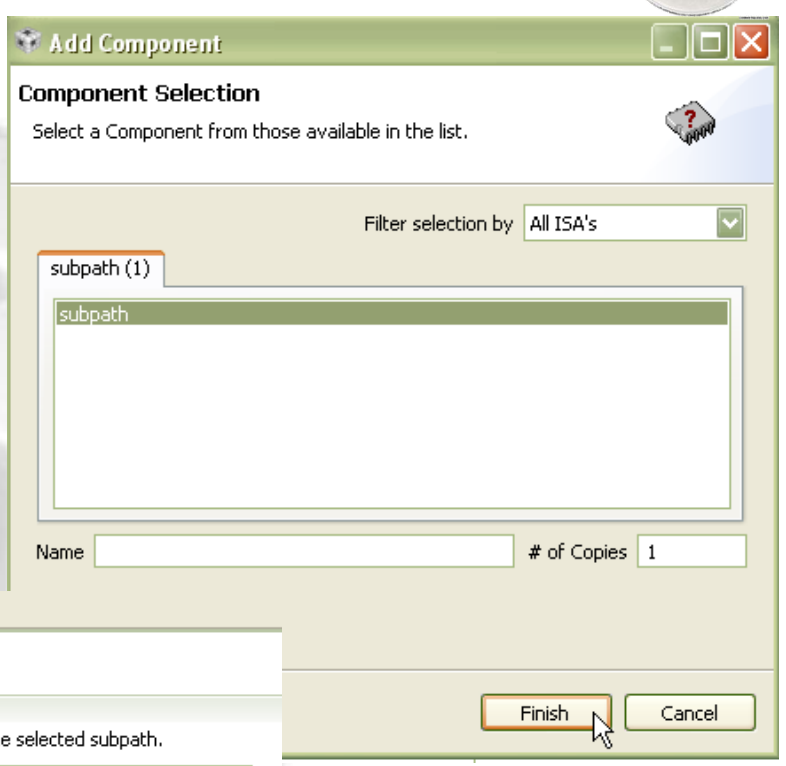
- Click on the ‘subpath_0’ node and press the ‘Add....’ button
- Add an ‘CS_DAP_PC’ from the list of Routers and press ‘Finish’
- A ‘CS_DAP_PC_0’ node will appear below ‘subpath’
- For the ‘CS_DAP_PC_0’ node, set:
  - Address = 3556888624 (0xd401d030)
  - DAP TAP ID = 191717423 (0xb6d602f)

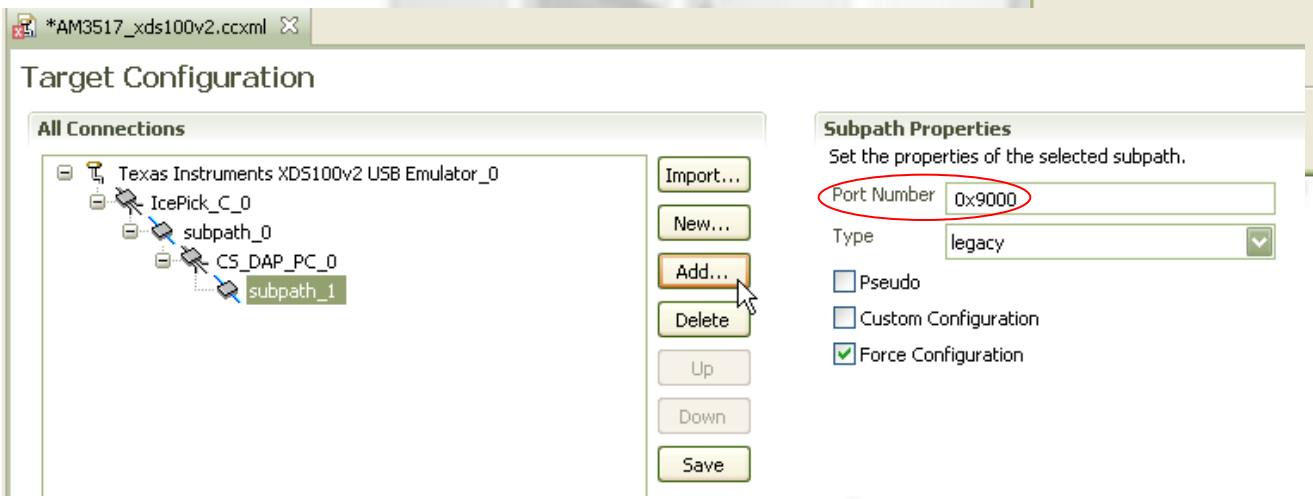




Creating a Target Configuration

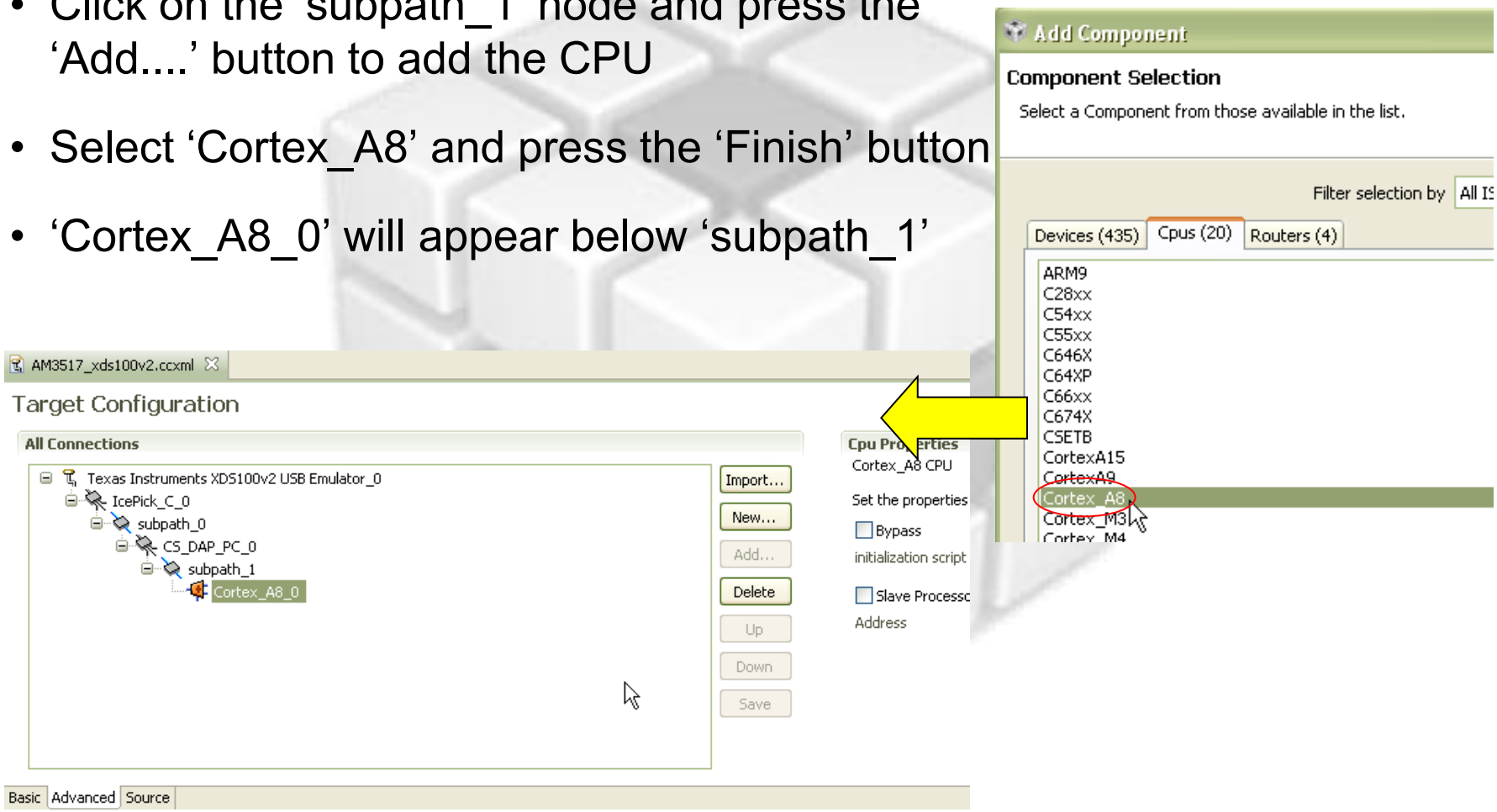
- Click on the ‘CS_DAP_PC_0’ node and press the ‘Add....’ button to add a subpath
- Select ‘subpath’ and press the ‘Finish’ button
- A ‘subpath_1’ node will appear below ‘CS_DAP_PC_0’
- Set the ‘Port Number’ for ‘subpath_1’ to be ‘0x9000’





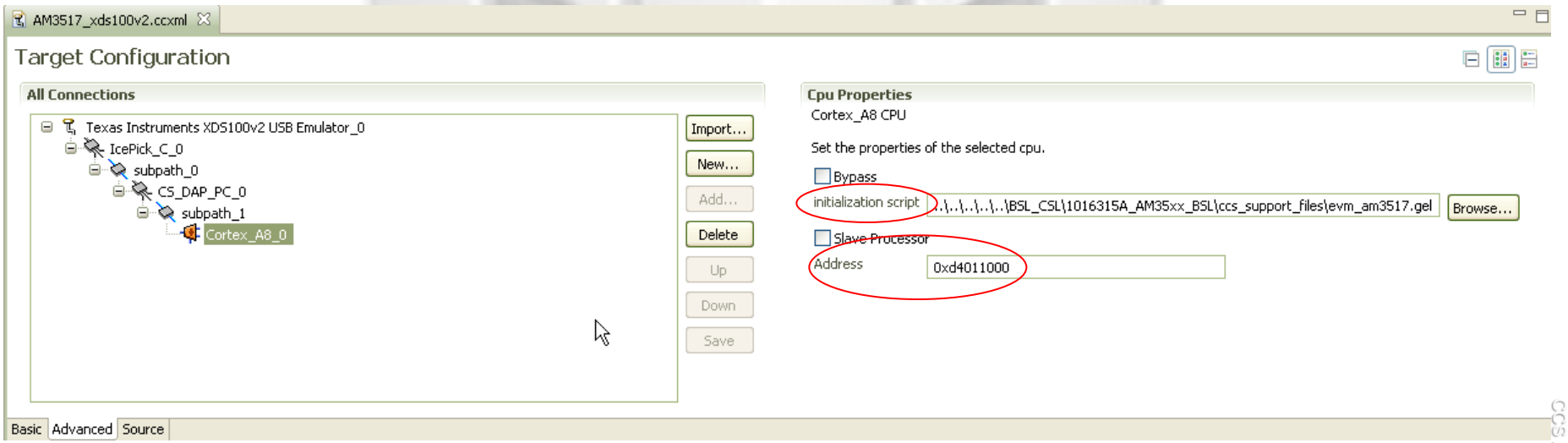
Creating a Target Configuration

- Click on the ‘subpath_1’ node and press the ‘Add....’ button to add the CPU
- Select ‘Cortex_A8’ and press the ‘Finish’ button
- ‘Cortex_A8_0’ will appear below ‘subpath_1’



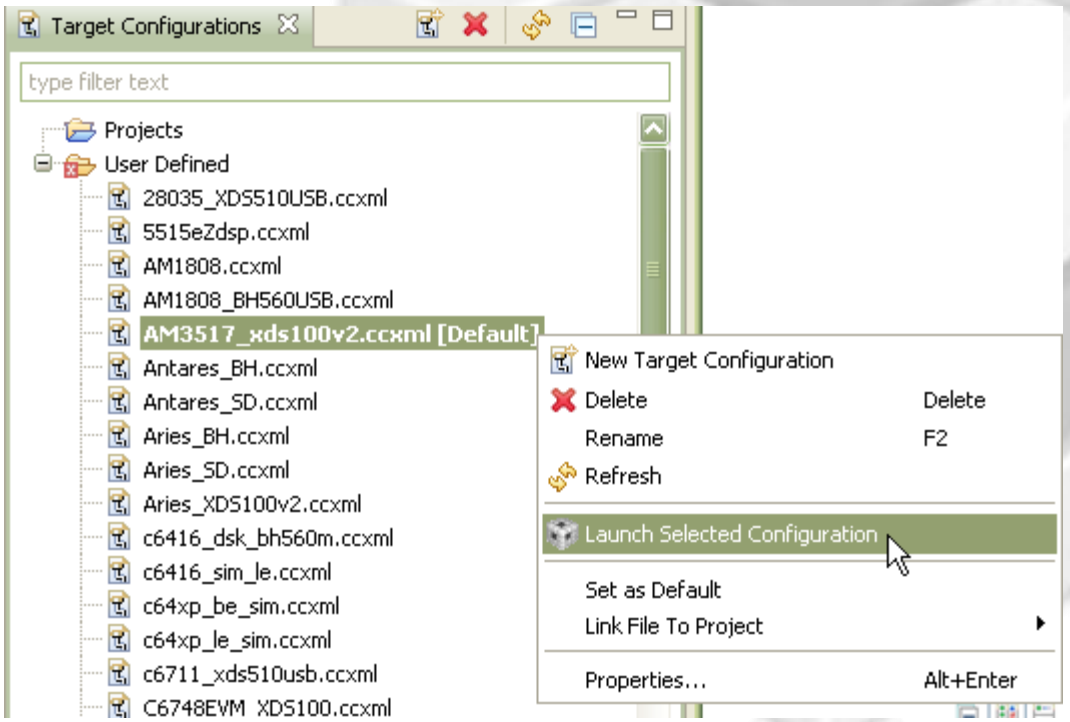
Creating a Target Configuration

- Click on the ‘Cortex_A8_0’ node
- Set the ‘initialization script’ to use the ‘evm_am3517.gel’ file
  - Browse to the location of the ‘evm_am3517.gel’ file
    - Location is: ‘C:\TI\AM3517XP\1016315A_AM35xx_BSL\ccs_support_files’
- Set ‘Address’ to ‘0xd4011000
- Click Save!



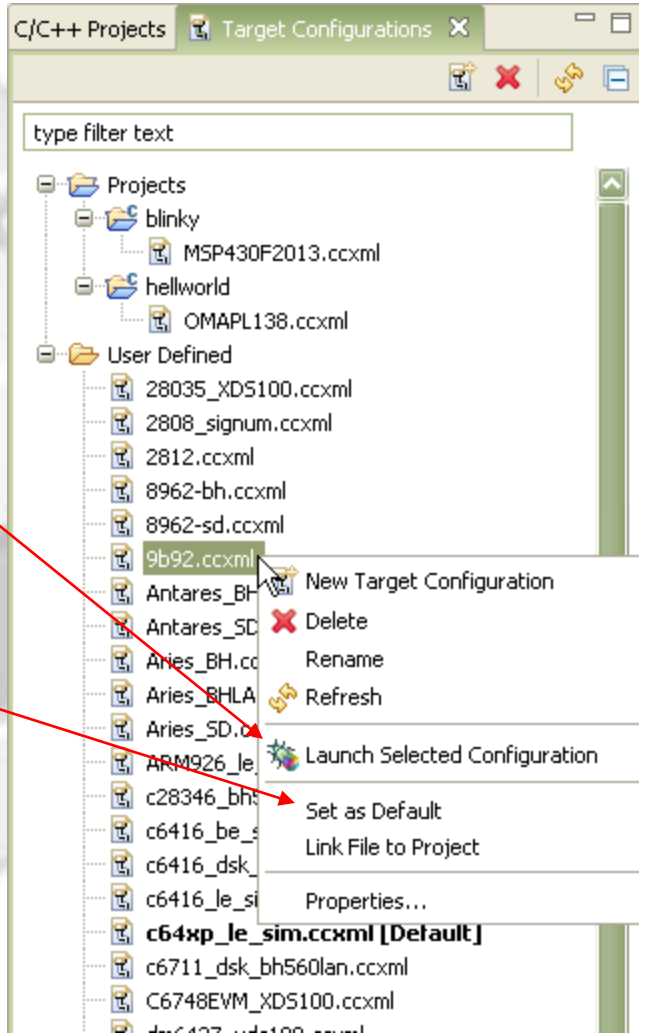
Launch a Debug Session

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



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'**
- **'Launch TI Debugger' 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

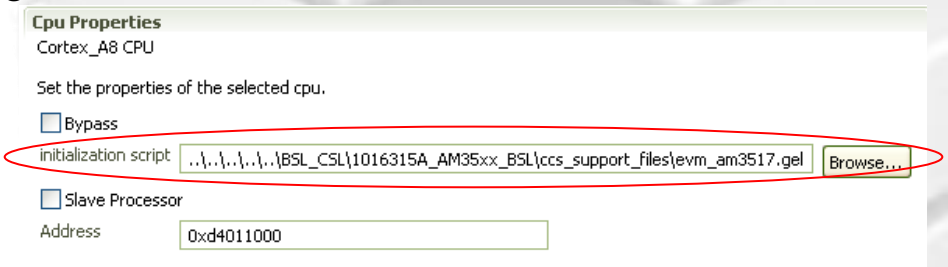


GEL (General Extension Language)

- **'C' like scripting language that extends CCS usefulness**
- **Interfaces to debuggers expression evaluator**
  - You can enter expressions in:
    - GEL file
    - Watch window
    - Conditions on breakpoints
    - Scripting Console
- **Many built-in GEL functions for common actions**
- **Create custom GEL functions for additional functionality**
  - Automate several steps within one GEL function
  - Create GEL "hotmenu" items for easy access from the 'Scripts' menu

GEL

- **Load the GEL script(s) into CCS memory**
  - Specify a default GEL file to be loaded upon launching a debug session using the Target Configuration editor's advanced section:



- **GEL callback functions are called on an event**
  - Startup(): Executed when the debugger is launched
  - OnTargetConnect(): Called when connection to the target has been established
  - OnFileLoaded(): Called after a program is loaded
  - OnPreFileLoaded(): Called before a program is loaded
  - OnReset(): Called after a target reset
  - OnRestart(): Called after a program restart
  - OnHalt(): Called after a user halt

Example GEL Script: evm_am3517.gel

```
menuitem "evm_am3517"


StartUp()

{

    // Do nothing

}


OnTargetConnect()

{

    Startup_Sequence();

}


OnFileLoaded()

{

    GEL_Reset();

    GEL_Restart();

}


OnReset()

{

     // Do nothing

}


OnRestart()

{

    // Do nothing

}
```

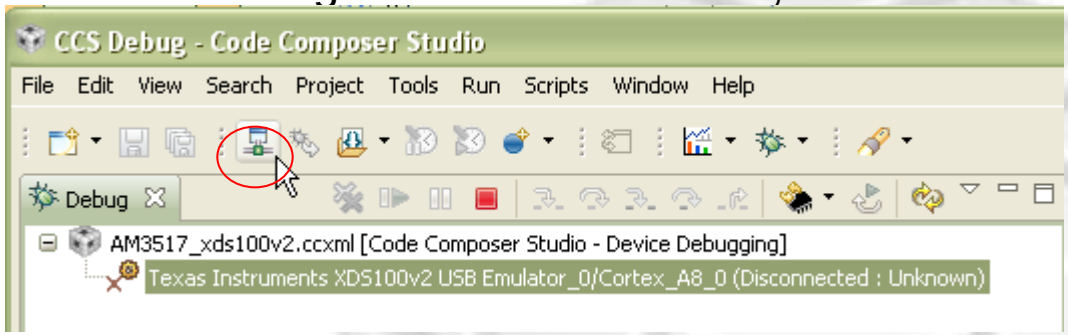
```
hotmenu Startup_Sequence()

{

    GEL_Reset();

    GEL_MapOff();

    GEL_MapReset();

    MemoryMap_Init();

    GEL_MapOn();

    GEL_TextOut("EVM AM3517 Startup Sequence
    Complete\n");

}


MemoryMap_Init()

{

    /* !! FOLLOWING MEM SPACE TO BE CONFIGURED PROPERLY
    !! */

    GEL_MapAddStr(0x00000000, 0, 0x40000000, "R|W", 0);
    /* GPMC */

    GEL_MapAddStr(0x40200000, 0, 0x4020FFFF, "R|W", 0);
    /* GPMC */


    /* L4-peripheral memory space mapping --------------
    ------------------------*/

    GEL_MapAddStr(0x48002000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - module */

    GEL_MapAddStr(0x48003000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - L4 interconnect */


...
```

GEL Limitations

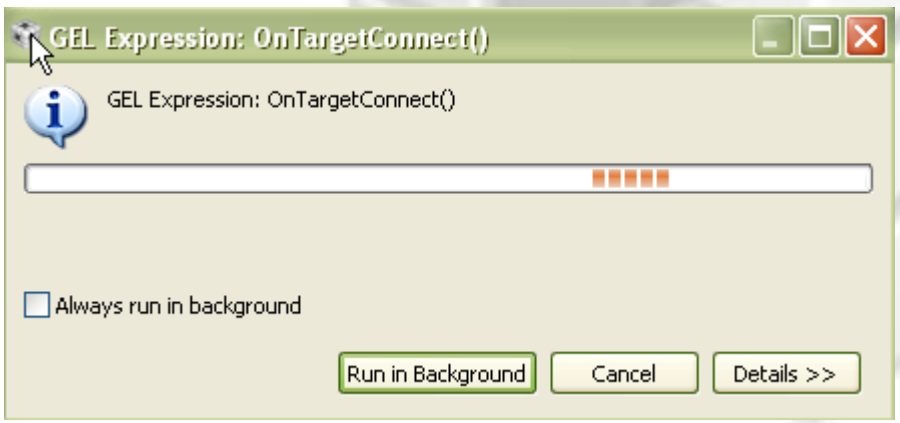
- **Can only be interfaced from within a CCS debug session**
  - GEL supports only debug actions from within a CCS debug session
  - No project management support
- **Asynchronous behavior of built-in functions**
  - Difficult to create long complex scripts
  - Built-in callback functions help but not enough
    - Cannot create custom callback functions
- **GEL was never meant for full blown automation purposes like overnight testing or batch-runs**
  - Debug Server Scripting (DSS) is a better solution for full automation

Launch a Debug Session

- Once the debug session is launched, connect to the target

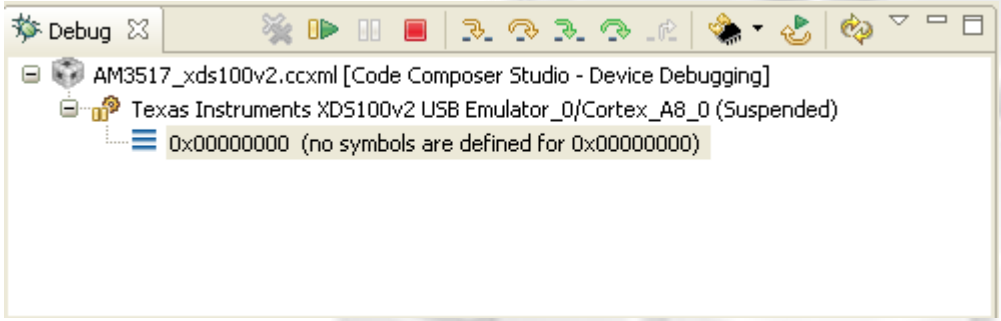


- This can take some time because of multiple actions in the ‘OnTargetConnect()’
- While connecting, the following popup may occur:

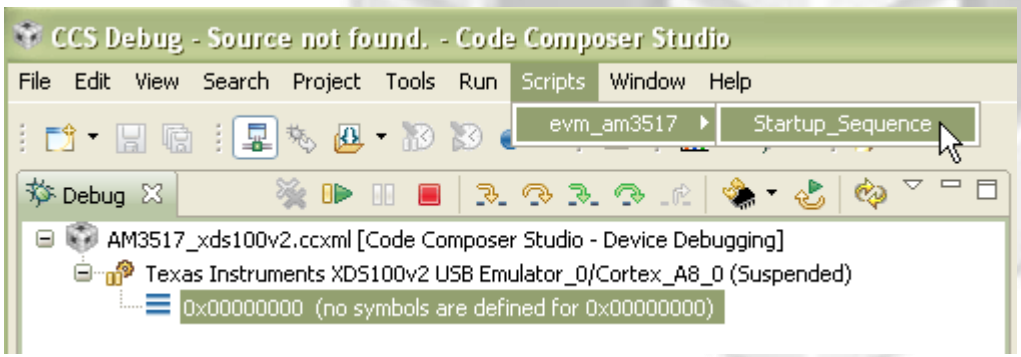


Launch a Debug Session

- Target should now be in a connected state



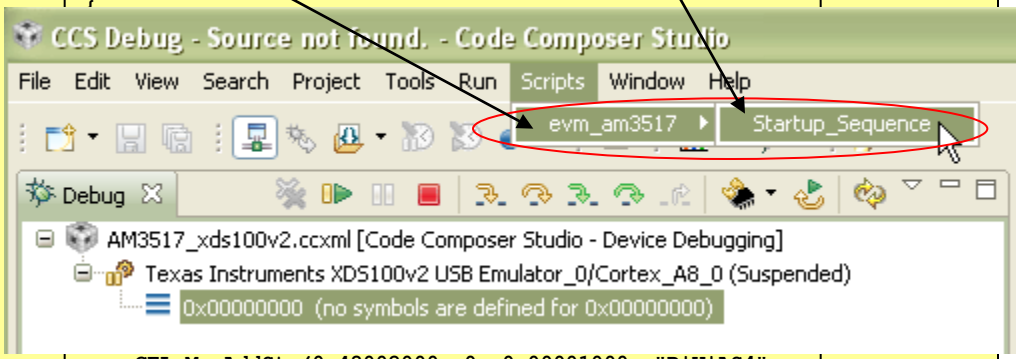
- Click on the 'Scripts'menu and notice the menu items 'evm_am3517->Startup_Sequence'



Example: Example GEL Script: evm_am3517.gel

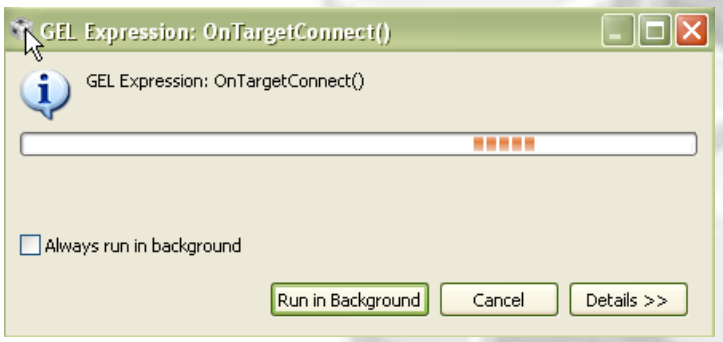
```
menuitem "evm_am3517"

StartUp()
{
    // Do nothing
}

OnTargetConnect()
{
    Startup_Sequence();
}

OnFileLoaded()
{
    GEL_Reset();
    GEL_Restart();
}

OnReset()
{
    // Do nothing
}

OnRestart()
{
    // Do nothing
}
```

```
hotmenu Startup_Sequence()
{
    GEL_Reset();
    GEL_MapOff();
    GEL_MapReset();
    MemoryMap_Init();
    GEL_MapOn();
    GEL_TextOut("EVM AM3517 Startup Sequence
    Complete\n");
}

MemoryMap_Init()
{
    GEL_MapAddStr(0x48002000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - module */
    GEL_MapAddStr(0x48003000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - L4 interconnect */

...
```



Launch a Debug Session

- Disconnect the target and reconnect by toggling the connect button 
- Note that the reconnect is pretty slow…

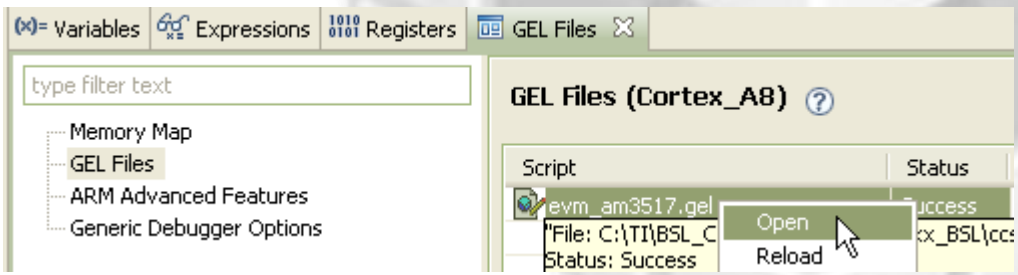


- What can we do to improve things?

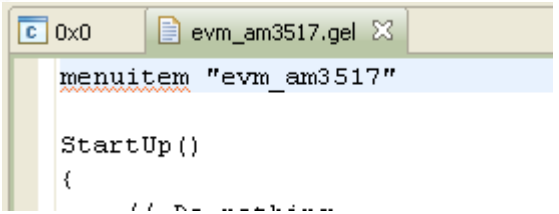
Improve Connect Time

- Let's open up the startup GEL file in CCS and take a look
- Tools -> GEL Files

- Right-click on the gel file and select 'Open' to open in the editor

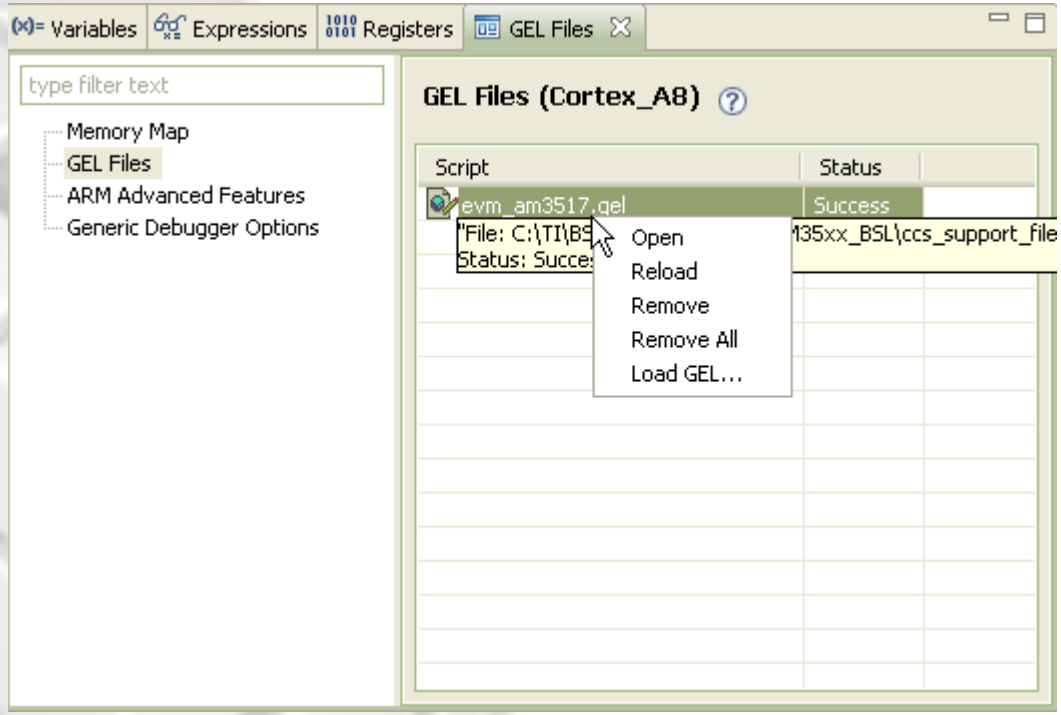


- The gel file should now appear in the editor



View: GEL Files

- Displays all loaded GEL files for the current debug session and the status of each
- 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



Example GEL Script: evm_am3517.gel

```
menuitem "evm_am3517"

StartUp()
{
    // Do nothing
}

OnTargetConnect()
{
    Startup_Sequence();
}

OnFileLoaded()
{
    GEL_Reset();
    GEL_Restart();
}

OnReset()
{
     // Do nothing
}

OnRestart()
{
    // Do nothing
}
```

```
hotmenu Startup_Sequence()
{
    GEL_Reset();
    GEL_MapOff();
    GEL_MapReset();
    MemoryMap_Init();
    GEL_MapOn();
    GEL_TextOut("EVM AM3517 Startup Sequence
    Complete\n");
}


MemoryMap_Init()
{
    /* !! FOLLOWING MEM SPACE TO BE CONFIGURED PROPERLY
    !! */
    GEL_MapAddStr(0x00000000, 0, 0x40000000, "R|W", 0);
    /* GPMC */
    GEL_MapAddStr(0x40200000, 0, 0x4020FFFF, "R|W", 0);
    /* GPMC */


    /* L4-peripheral memory space mapping --------------
    ------------------------*/
    GEL_MapAddStr(0x48002000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - module */
    GEL_MapAddStr(0x48003000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - L4 interconnect */


...
```

Example GEL Script: evm_am3517.gel

```
menuitem "evm_am3517"

StartUp()
{
    // Do nothing
}

OnTargetConnect()
{
    Startup_Sequence();
}

OnFileLoaded()
{
    GEL_Reset();
    GEL_Restart();
}

OnReset()
{
    // Do nothing
}

OnRestart()
{
    // Do nothing
}
```

OnTargetConnect() calls Startup_Sequence() every time

```
hotmenu Startup_Sequence()
{
    GEL_Reset();
    GEL_MapOff();
    GEL_MapReset();
    MemoryMap_Init();
    GEL_MapOn();
    GEL_TextOut("EVM AM3517 Startup Sequence
    Complete\n");
}

MemoryMap_Init()
{
    /* !! FOLLOWING MEM SPACE TO BE CONFIGURED PROPERLY
    !! */
    GEL_MapAddStr(0x00000000, 0, 0x40000000, "R|W", 0);
    /* GPMC */
    GEL_MapAddStr(0x40200000, 0, 0x4020FFFF, "R|W", 0);
    /* GPMC */

    /* L4-peripheral memory space mapping --------------
    ------------------------*/
    GEL_MapAddStr(0x48002000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - module */
    GEL_MapAddStr(0x48003000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - L4 interconnect */

...
```

GEL_Reset() is the only action in Startup_Sequence() that needs to have target access

No reason have the rest of these actions in Startup_Sequence() need to occur every time the target is connected

Improved GEL Script: evm_am3517.gel

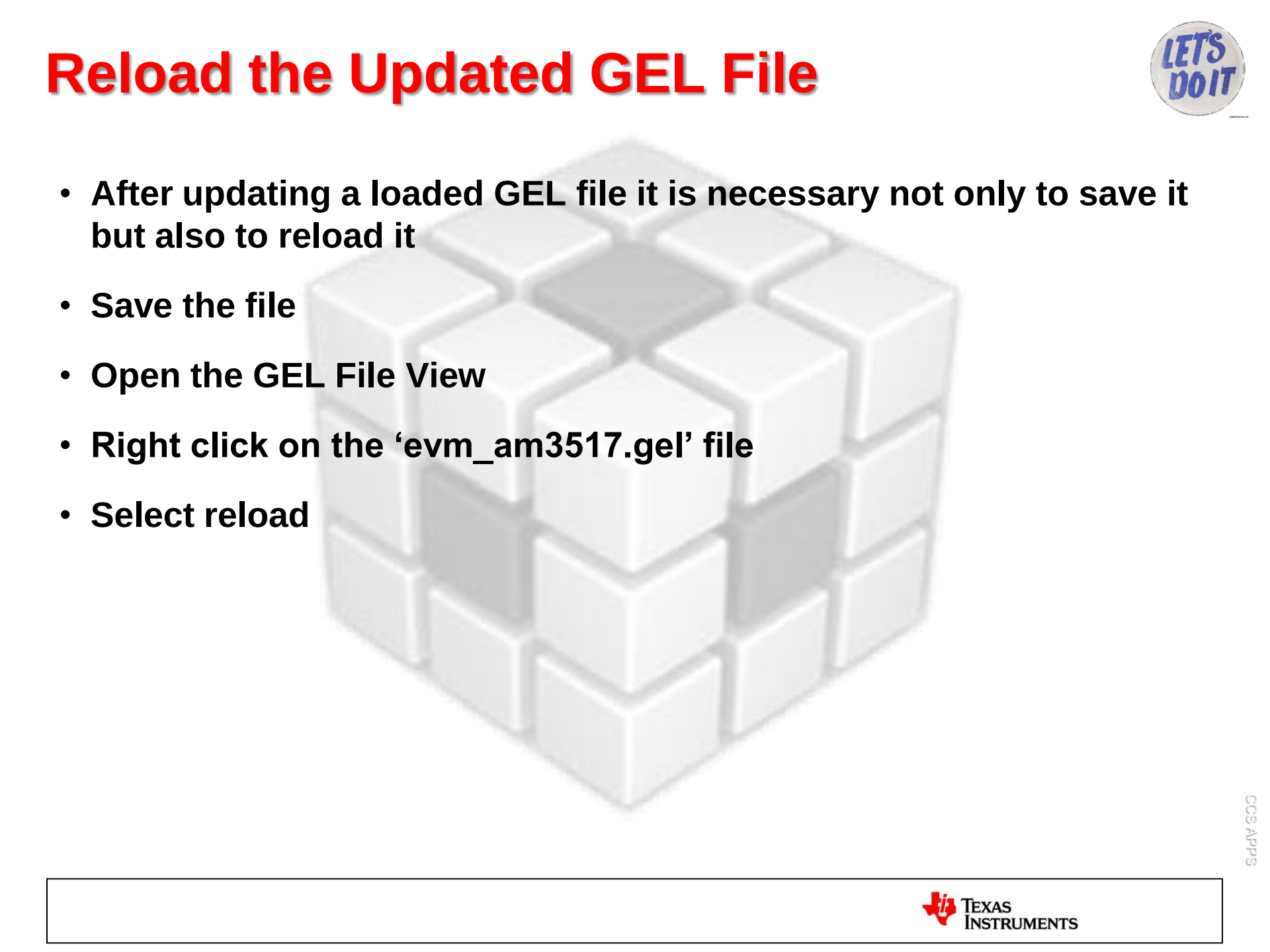

```
menuitem "evm_am3517"

StartUp()
{
    Startup_Sequence();
}

OnTargetConnect()
{
    GEL_Reset();
}

OnFileLoaded()
{
    GEL_Reset();
    GEL_Restart();
}

OnReset()
{
    // Do nothing
}

OnRestart()
{
    // Do nothing
}
```

Startup_Sequence() call moved into Startup()

```
hotmenu Startup_Sequence()
{
    GEL_MapOff();
    GEL_MapReset();
    MemoryMap_Init();
    GEL_MapOn();
    GEL_TextOut("EVM AM3517 Startup Sequence
    Complete\n");
}


MemoryMap_Init()
{
    /* !! FOLLOWING MEM SPACE TO BE CONFIGURED PROPERLY
    !! */
    GEL_MapAddStr(0x00000000, 0, 0x40000000, "R|W", 0);
    /* GPMC */
    GEL_MapAddStr(0x40200000, 0, 0x4020FFFF, "R|W", 0);
    /* GPMC */


    /* L4-peripheral memory space mapping --------------
    ------------------------*/
    GEL_MapAddStr(0x48002000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - module */
    GEL_MapAddStr(0x48003000, 0, 0x00001000, "R|W|AS4",
    0);  /*  system control - L4 interconnect */


...
```

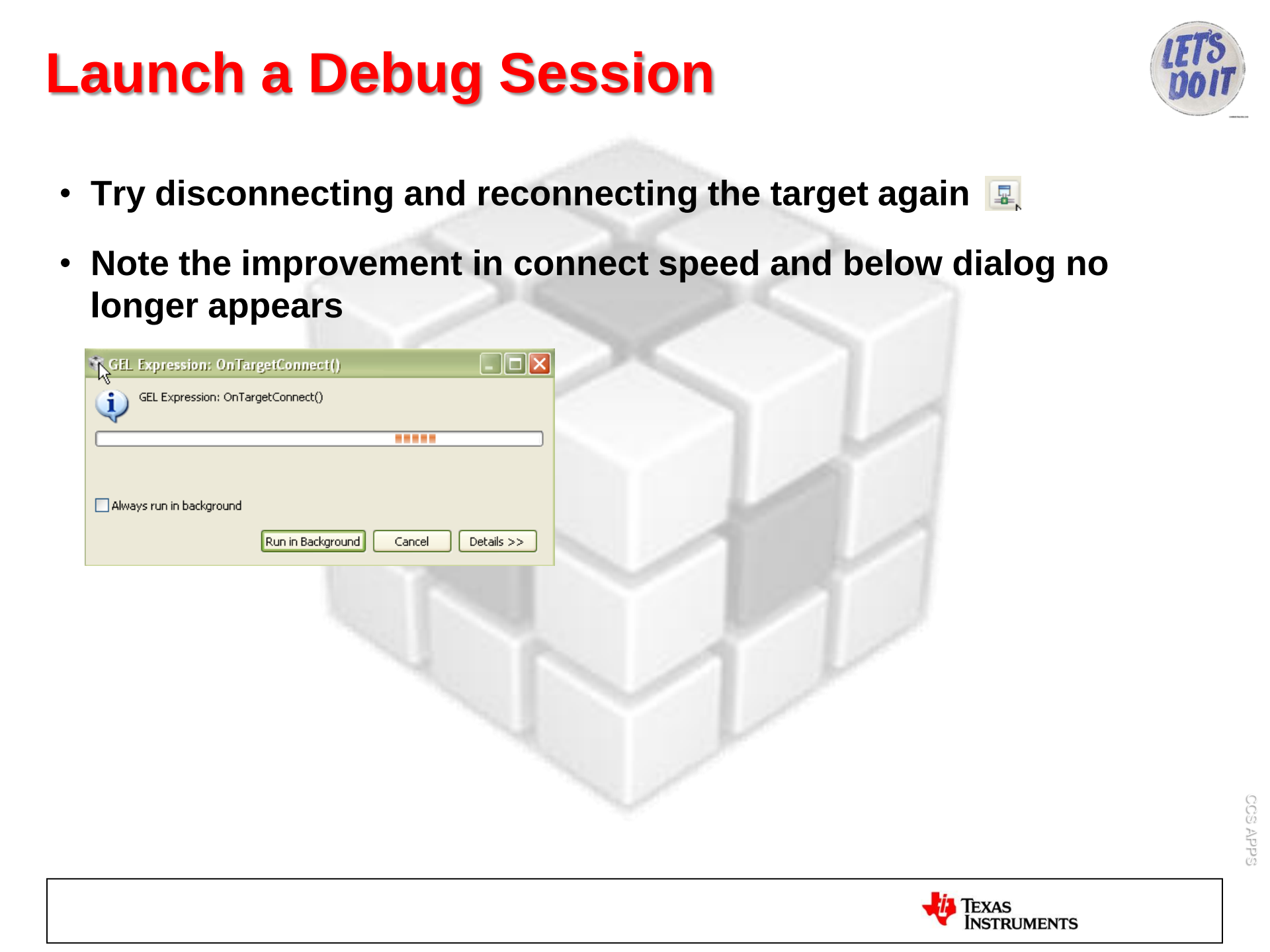
GEL_Reset() moved to OnTargetConnect()

Reload the Updated GEL File



- After updating a loaded GEL file it is necessary not only to save it but also to reload it
- Save the file
- Open the GEL File View
- Right click on the ‘evm_am3517.gel’ file
- Select reload

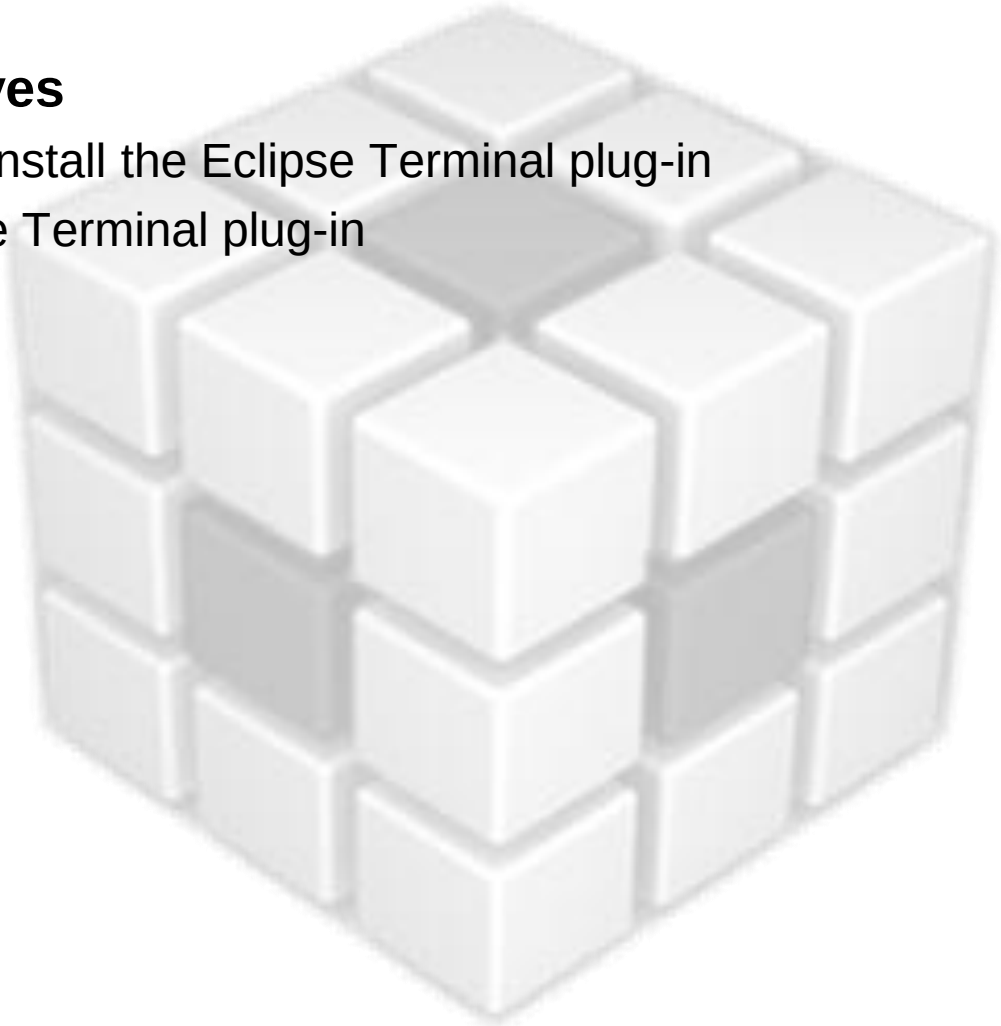
Launch a Debug Session



- Try disconnecting and reconnecting the target again
- Note the improvement in connect speed and below dialog no longer appears

ECLIPSE PLUG-INS

Install an Eclipse Plug-in: Briefing



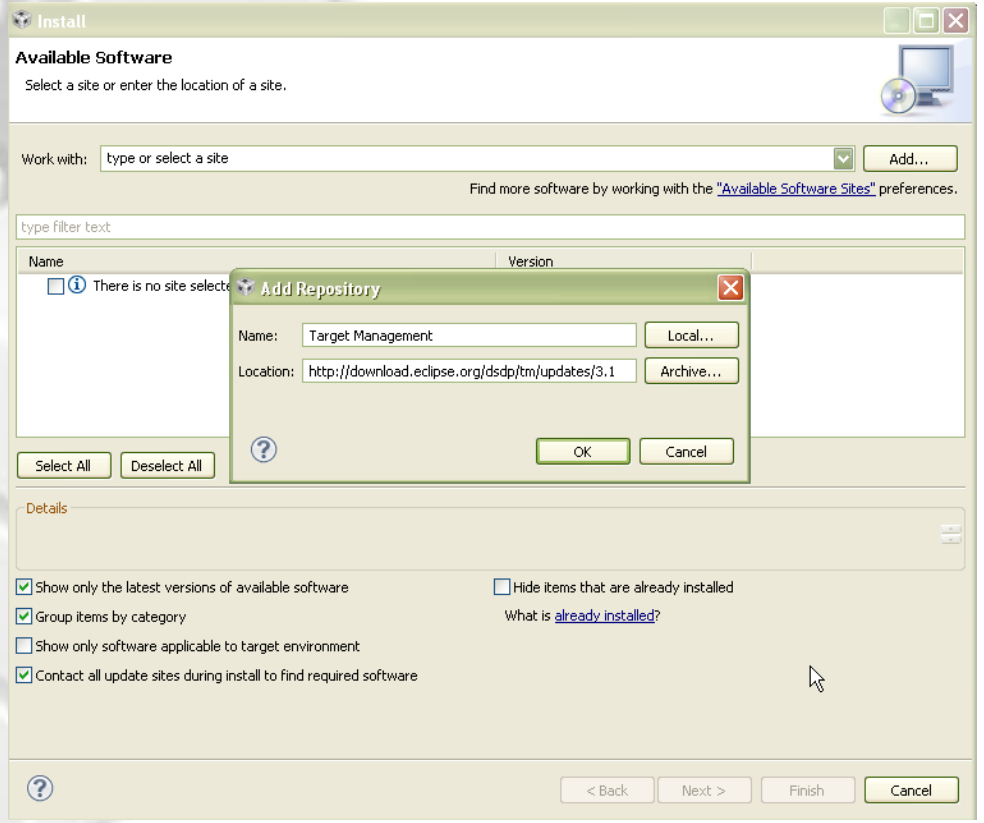
- **Key Objectives**
  - Download/Install the Eclipse Terminal plug-in
  - Validate the Terminal plug-in

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



Eclipse Plug-ins – Terminal Plug-in

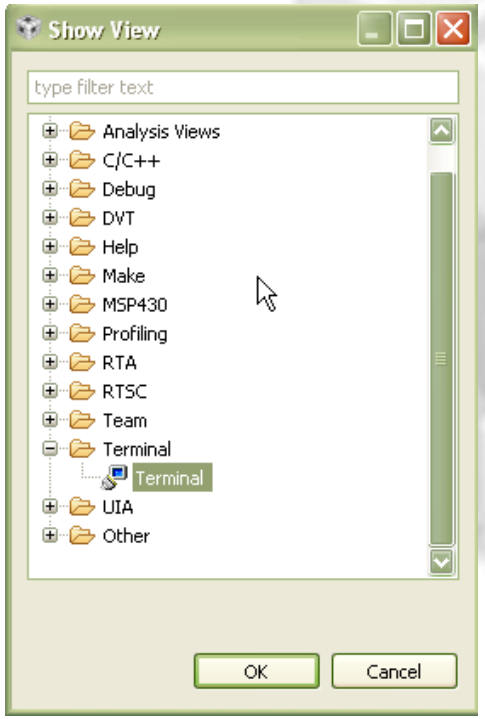
- **Provides the 'Terminal' view in the CCSv5 IDE**
- **Can connect to a remote target via a serial port or over TCP/IP using the TELNET or SSH protocol**
- **Here are the steps to install the plug-ins using the remote site:**
  - Select Help -> Install New Software
  - Add the Target Management repository
    - http://download.eclipse.org/dsdp/tm/updates/3.1
  - Select the "Target Management Terminal" plugin
  - Follow the install wizard to finish the plugin installation and
  - Restart CCS
  - Select Help -> Install New Software
  - Add the RXTX repository
    - http://rxtx.qbang.org/eclipse/
  - Select the latest version of the RXTX plugin
  - Follow the install wizard to finish the plugin installation and
  - Restart CCS

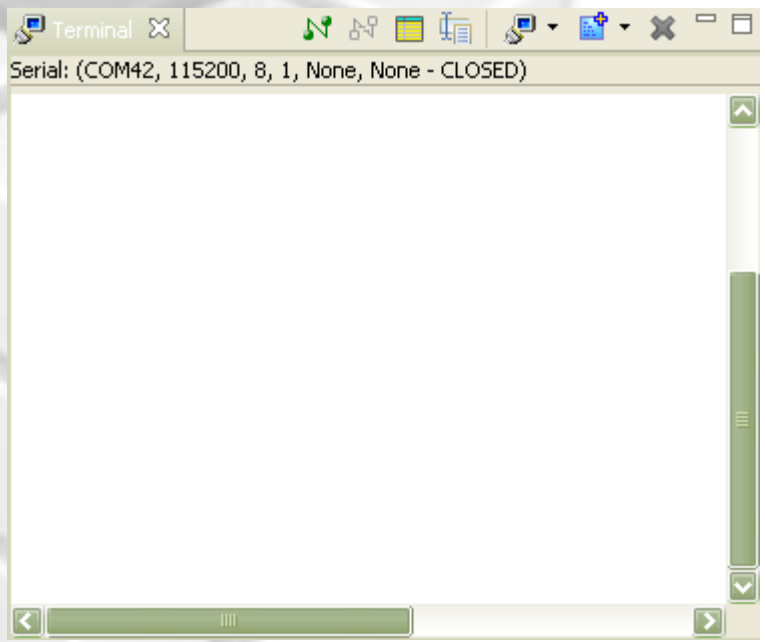
Eclipse Plug-ins – Terminal Plug-in

- **Steps to install the plug-ins with local files**
  - Target Management Terminal
    - Go to C:\TI\Plugins\TM-terminal-3.3
    - Copy and paste the contents of the eclipse folder into '<INSTALL DIR>\ccsv5\eclipse\dropins'
    - Do not copy the eclipse folder itself, just the contents of it
  - RXTX for Terminal
    - Go to C:\TI\Plugins\RXTX-SDK
    - Copy and paste the contents of the eclipse folder into '<INSTALL DIR>\ccsv5\ccs_base\eclipse\dropins'
    - Do not copy the eclipse folder itself, just the contents of it
    - Will prompt you to merge, say yes
- **Shutdown and restart CCS**

Eclipse Plug-ins – Terminal Plug-in

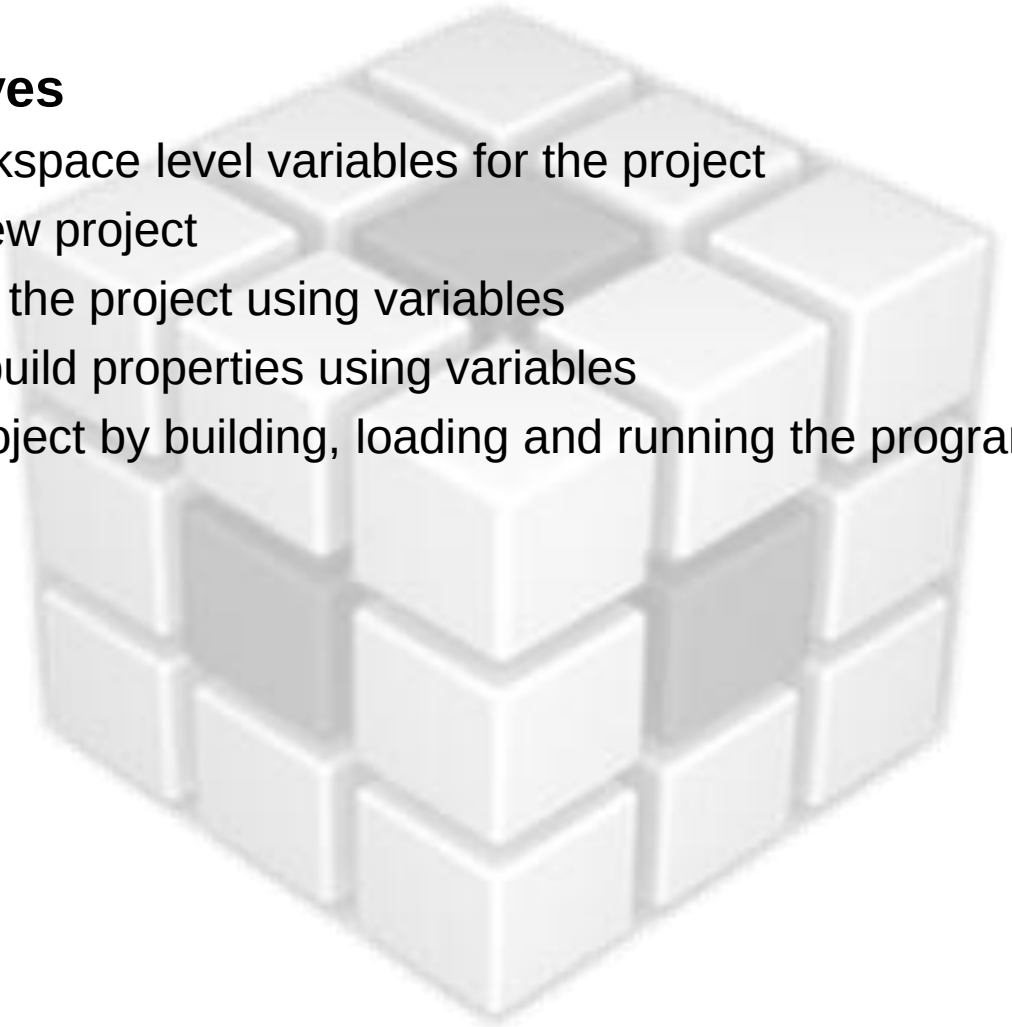
- **Check to see if the plugin has been installed by opening the 'Terminal' view**
  - View -> Other.. -> Terminal -> Terminal





SHARING PROJECTS

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 “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

Sharing Projects – Simple Projects

- **<u>USE CASE</u>: Wish to share (give) a project folder and all needed source files to build the project**
  - All source files are inside the project folder.
- **Easy to share projects with no linked files:**
  - The entire project folder can be distributed to another “as-is”
  - The user who receives the project can import it into their workspace using ‘Project -> Import Existing CCE/CCS Project’ and selecting the copied folder
  - Works well for simple projects that only reference files inside the project folder

Sharing Projects – Linked File Projects (1)

- **<u>USE CASE</u>: Wish to share (give) a project folder and all needed source files to build the project**
  - Some (or all) of the source files are linked to the project
- **Use the CCS Export Project to create an archive (zip) file that contains the project folder and all project linked source files**
  - Note that this technique will fail if Linked Resource Path variables are used!
- **Exporting your project: These instructions cover how to archive your project on the source computer**
  - File -> Export
  - Expand "General" and select "Archive File". Click "Next"
  - The dialog that appears allows you to select which projects you wish to export to the archive. When you select a project it will show you on the right hand side all of the items it is going to export. Check the box beside the project to indicate that you want to archive it.
  - Specify a name for your archive and select either zip or tar, then click "Finish"
  - You now have a zip file on your computer containing the project. Not only did it copy in the files that are physically located in your project directory but it also copied in your linked/referenced resources

Sharing Projects – Linked File Projects (2)

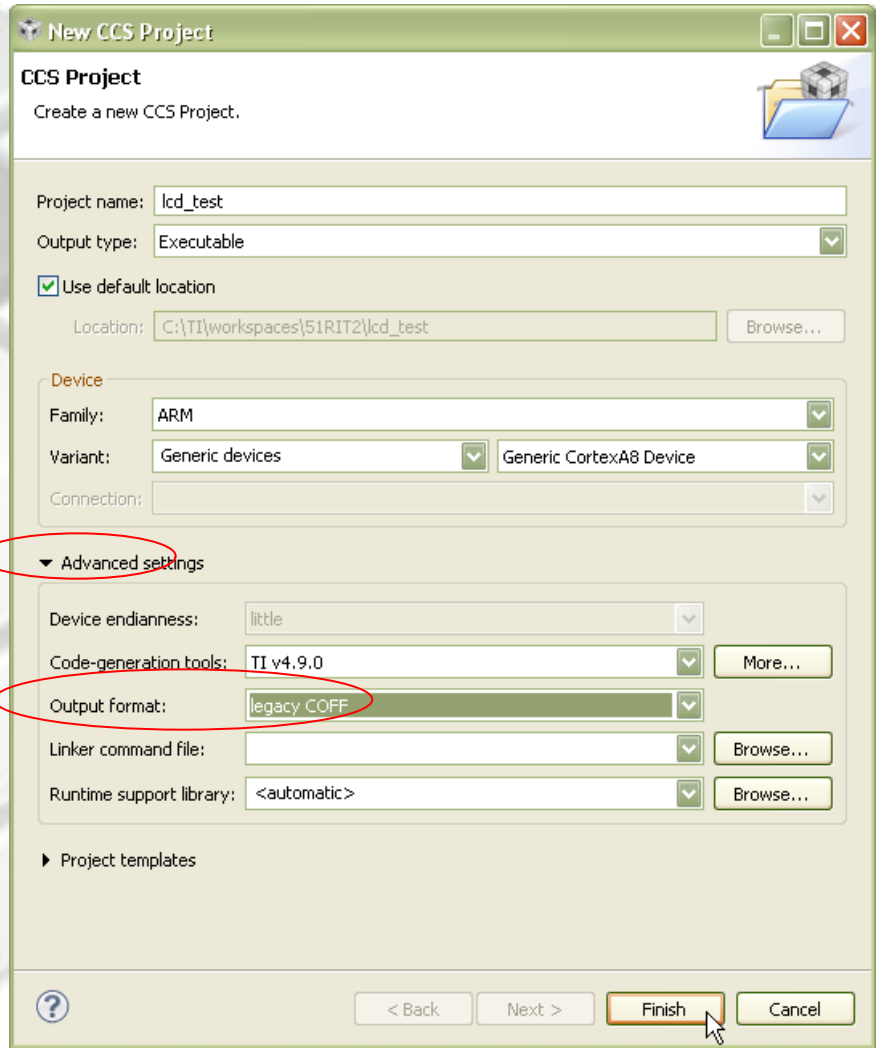
- **Importing the project on another machine: These instructions cover how to import the project from the archive. What happens is that it will import the project into your workspace**
  - Project -> Import Existing CCS/CCE Eclipse Project
  - Change the radio button selection at the top to "Select archive file"
  - Browse to and select your archive file
  - It will list all of the projects found in the archive. The are all selected by default. Select the ones you want and click "Finish"
  - The project is now in your workspace
- **For linked/referenced resources, it will copy those files from the archive and then place them at the same path where they were located on the original computer**

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 local copy of the source files
  - Wish to check the project folder/files into source control
- **Need to make the project portable to make sure the project is easily shared (people may have their copy of the source file tree in a different location)**
- **Portable projects should avoid any absolute paths**
- **Ideal portable projects should be usable without having to modify any of the project files before use**
  - This is ideal for projects maintained in source control

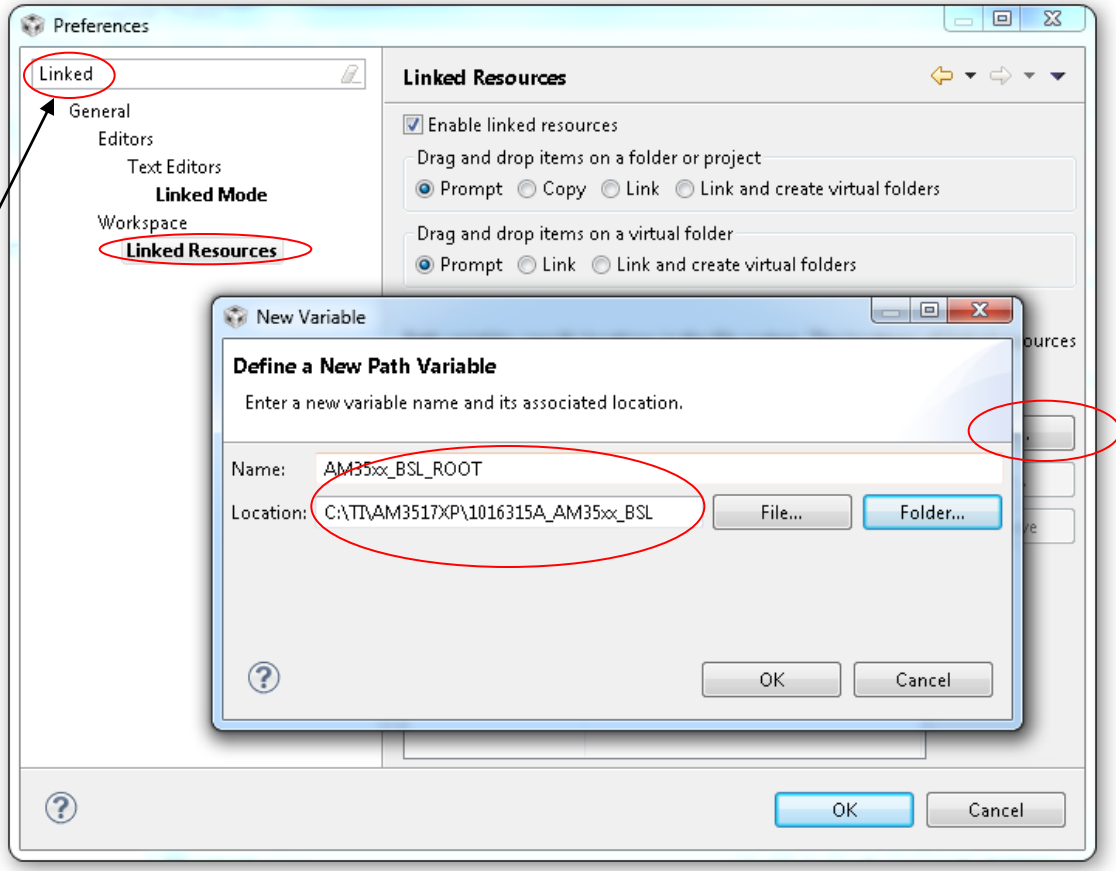
LCD Test: Create a New Project

- **Launch ‘New CCS Project’ Wizard**
  - Select ‘New Project’ from the Welcome page
- **Fill in the fields as shown in the right**
  - Note the change to ‘Output format” from the default ‘ELF’ to ‘legacy COFF’
- **Select ‘Finish’ when done**
- **Generated project will appear in the Project Explorer view**
- **Remove the generated ‘main.c’ file from the project**



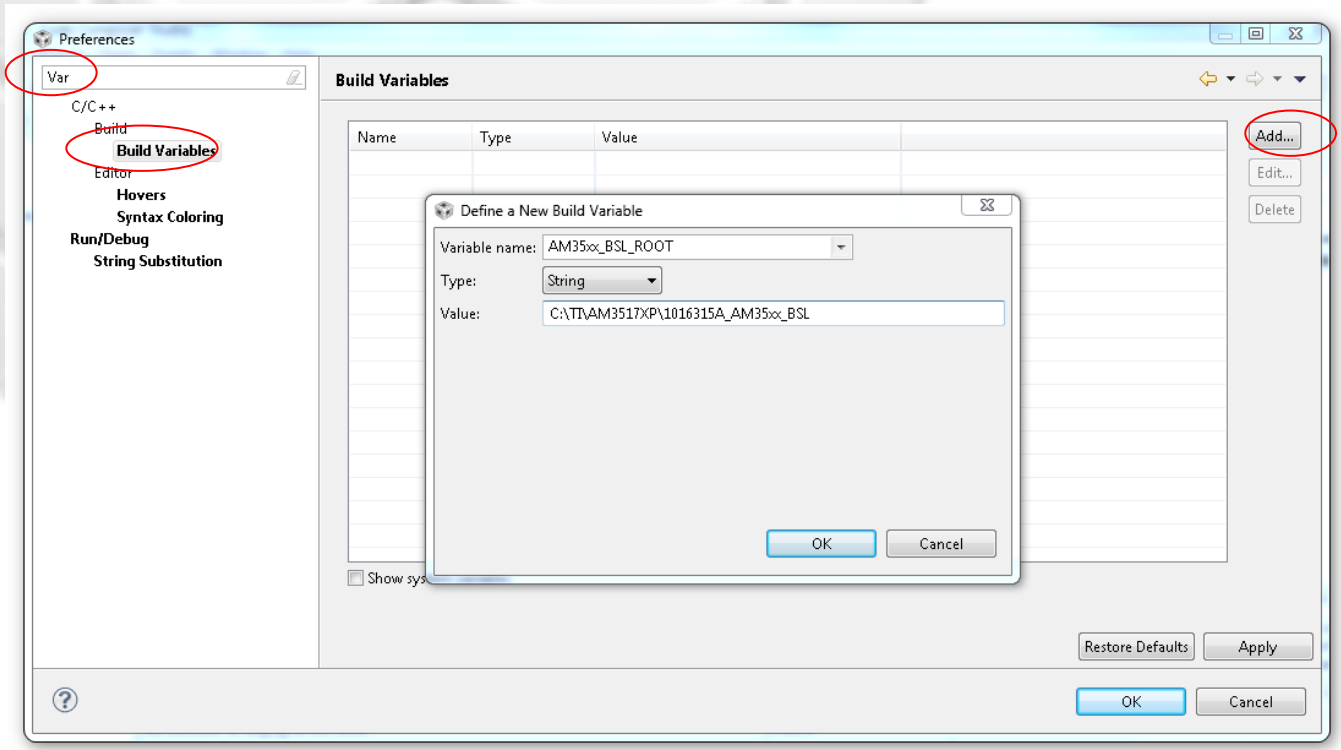
LCD Test: Create a Linked Resource Path Variable

- **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 AM35xx BSL directory**
- **Hit 'OK' when finished**



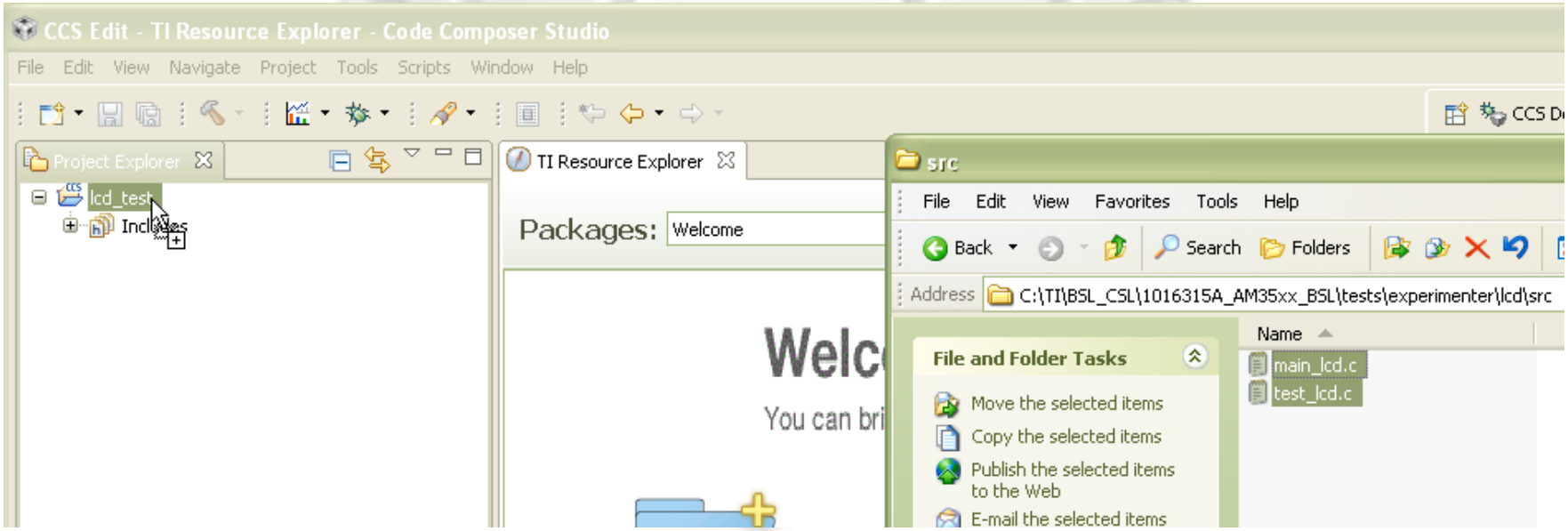
LCD Test: Create a Build Variable

- **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 AM35xx BSL directory**
- **Hit 'OK' when done**



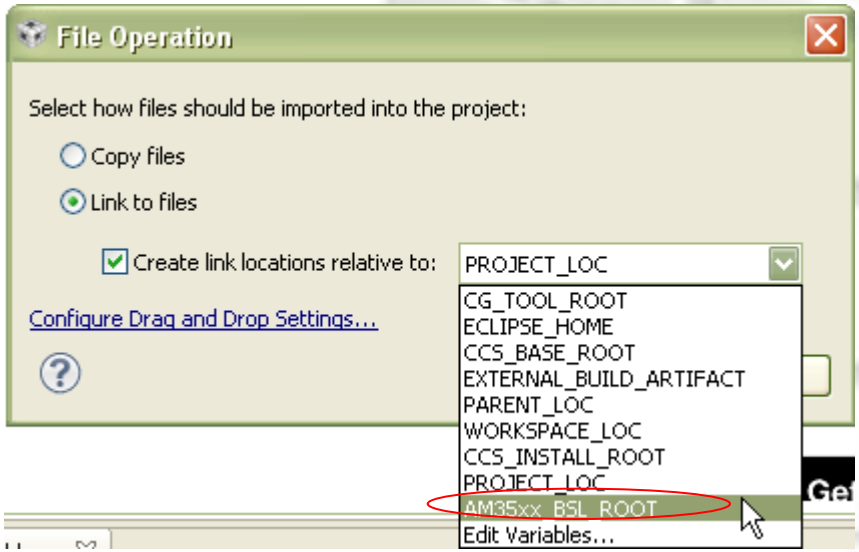
LCD Test: Link Source Files to Project

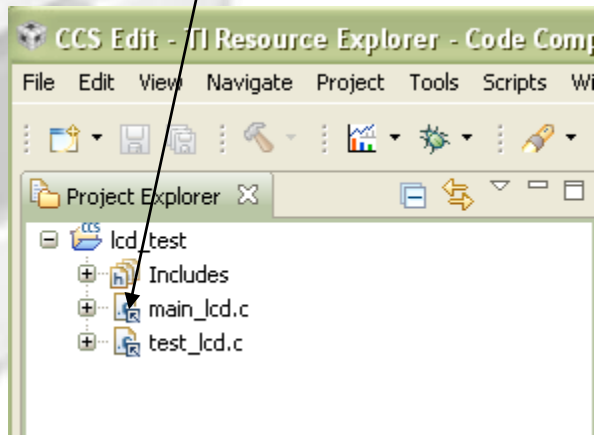
- **Open Windows Explorer and browse to:**
  - C:\TI\AM3517XP\1016315A_AM35xx_BSL\tests\experimenter\lcd\src
  - Drag and drop the two source files ‘main_lcd.c’ and ‘test_lcd.c’ into the new project in the CCS Project Explorer view



LCD Test: 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 (AM35xx_BSL_ROOT)
  - Hit ‘OK’
- **Files will now appear in the Project Explorer with the ‘link’ icon**



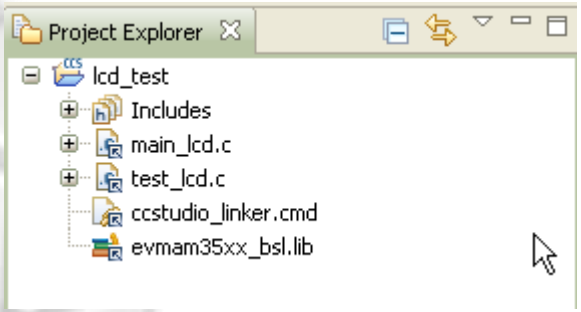


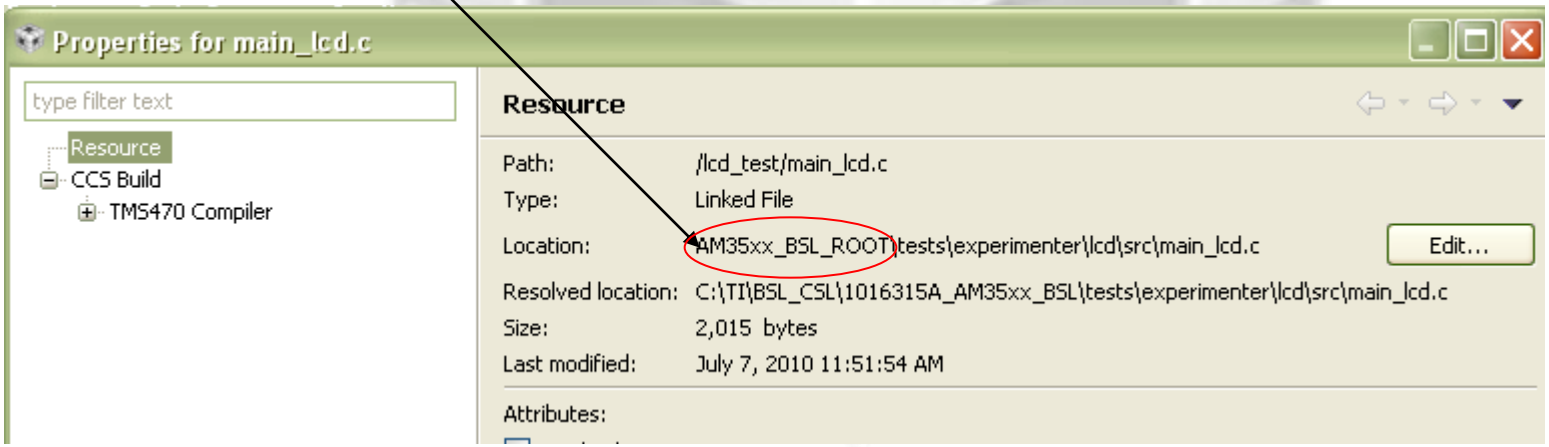
Eclipse Concept: Projects

- **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

LCD Test: Link Files to Project

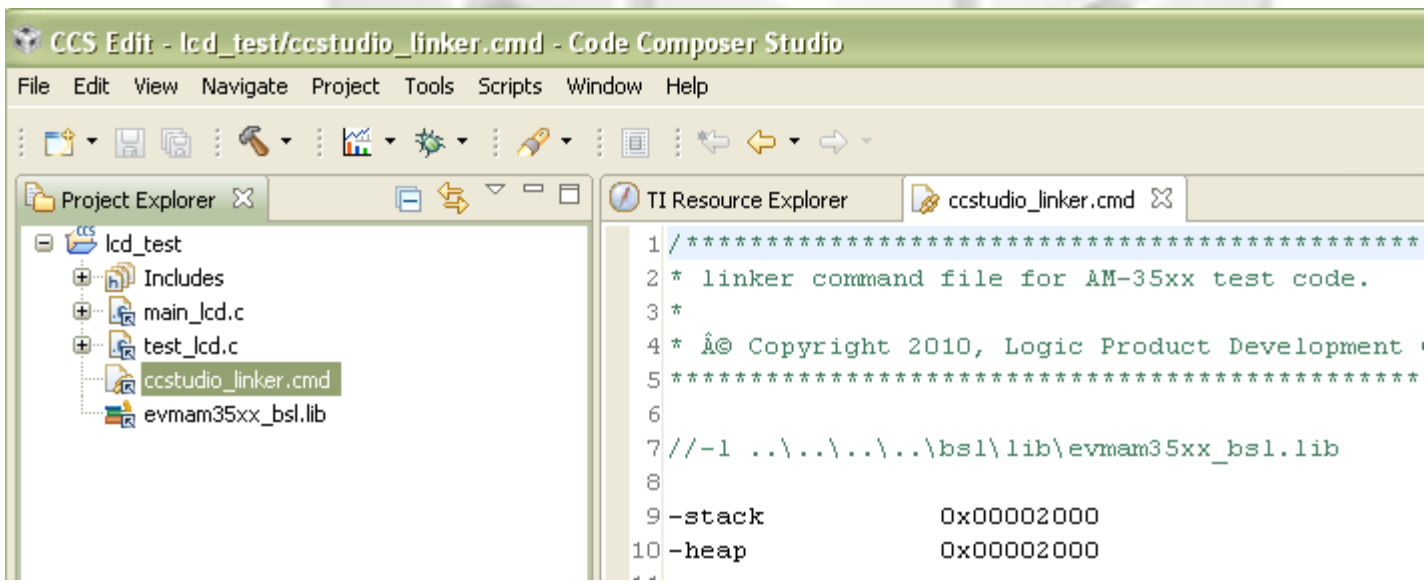
- **Link the 'evmam35xx_bsl.lib' and 'ccstudio_linker.cmd' files the same way:**
  - C:\TI\AM3517XP\1016315A_AM35xx_BSL\bsl\lib\evmam35xx_bsl.lib
  - C:\TI\AM3517XP\1016315A_AM35xx_BSL\ccs_support_files\ccstudio_linker.cmd
- **Four files should appear in the project when all the files are linked:**
- **Right-click on a source file and check the 'Properties'**
  - See how the 'Location' parameter references the Linked Resource Variable





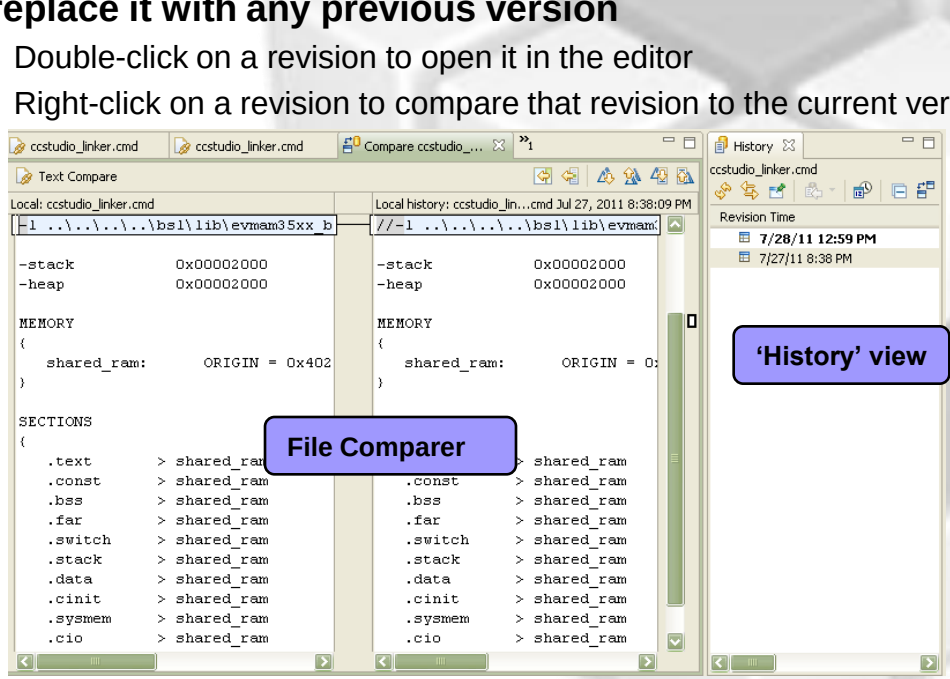
LCD Test: Editing Files

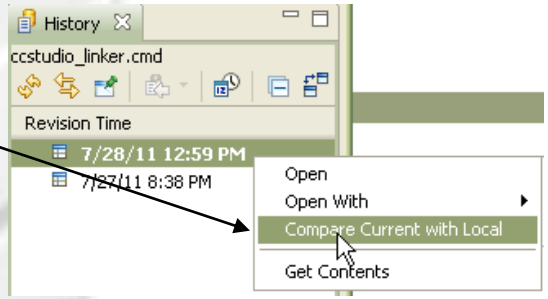
- **Open the ‘ccstudio_linker.cmd’ file**
  - NOTE: If double-clinking on the the file in the Project Explorer does not work, try dragging and dropping it into the editor
  - Comment out line #7 (as shown in the picture)
    - ‘evmam35xx_bsl.lib’ has already been explicitly added to the project
  - Save file when done



View: Local History

- **CCS keeps a local history of source changes**
  - Right-click on a file in the editor an 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

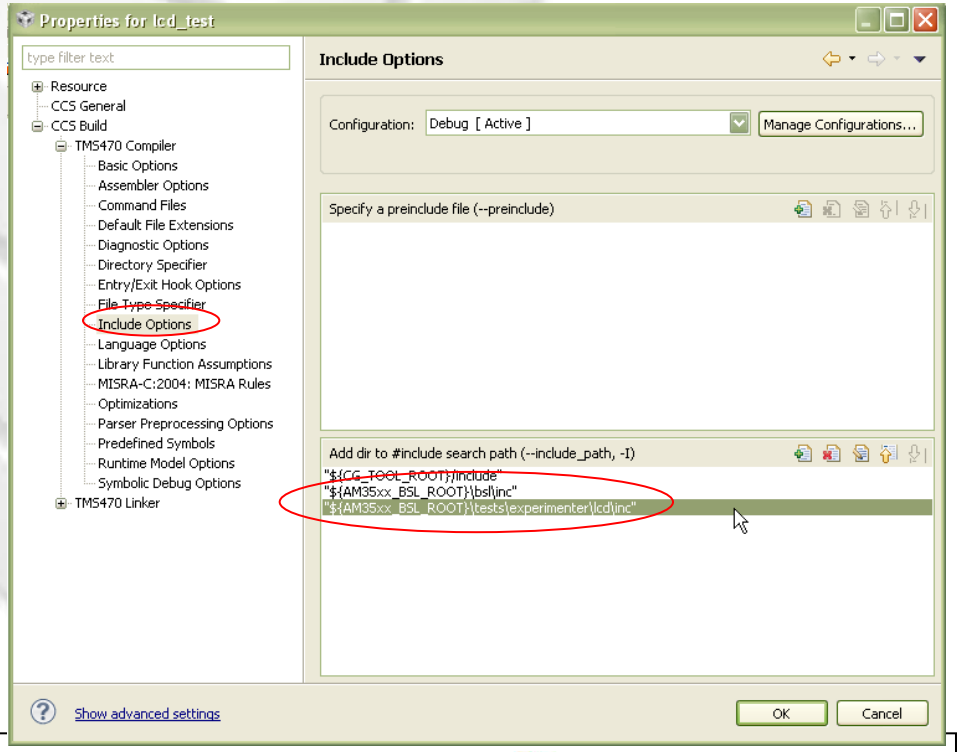




- **CCS also keeps project history**
  - Recover files deleted from the project
  - Right-click on the project and select “Recover from Local History” in the context menu

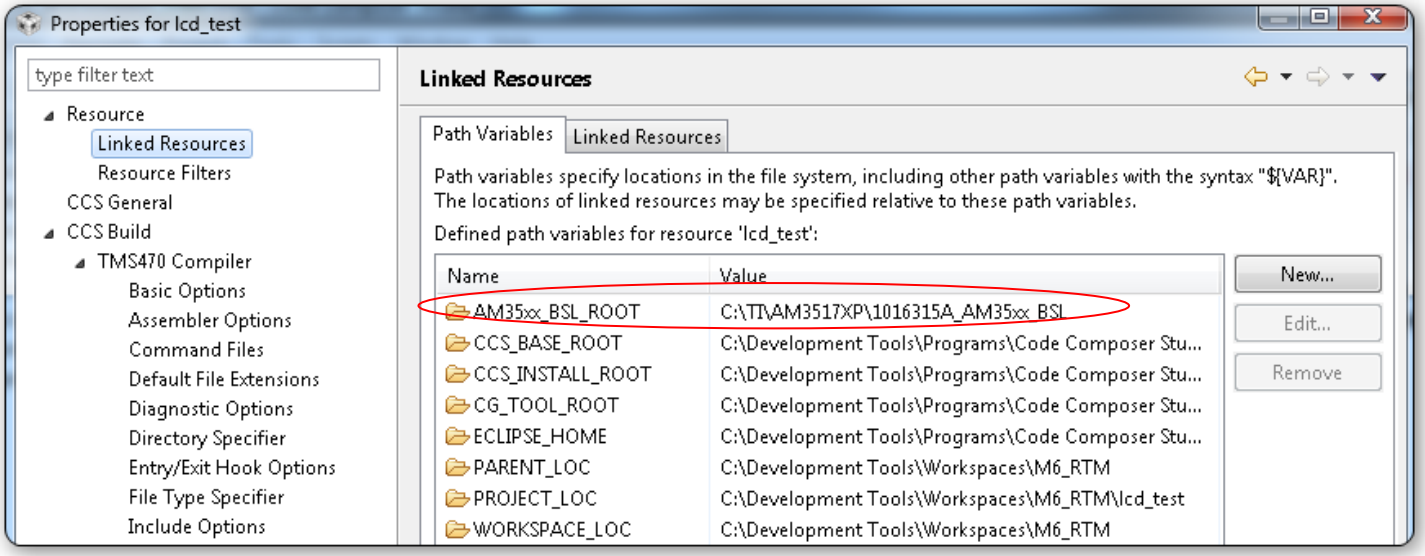
LCD Test: Modifying Project Properties

- **Right-click on the project and select ‘Properties’**
- **In the compiler ‘Include Options’, add the following entries to the list of include search paths:**
  - "${AM35xx_BSL_ROOT}\bsl\inc“
  - "${AM35xx_BSL_ROOT}\tests\experimenter\lcd\inc“
- **‘${<BUILD VARIABLE>}’ is the syntax to use a Build Variable in the project properties**



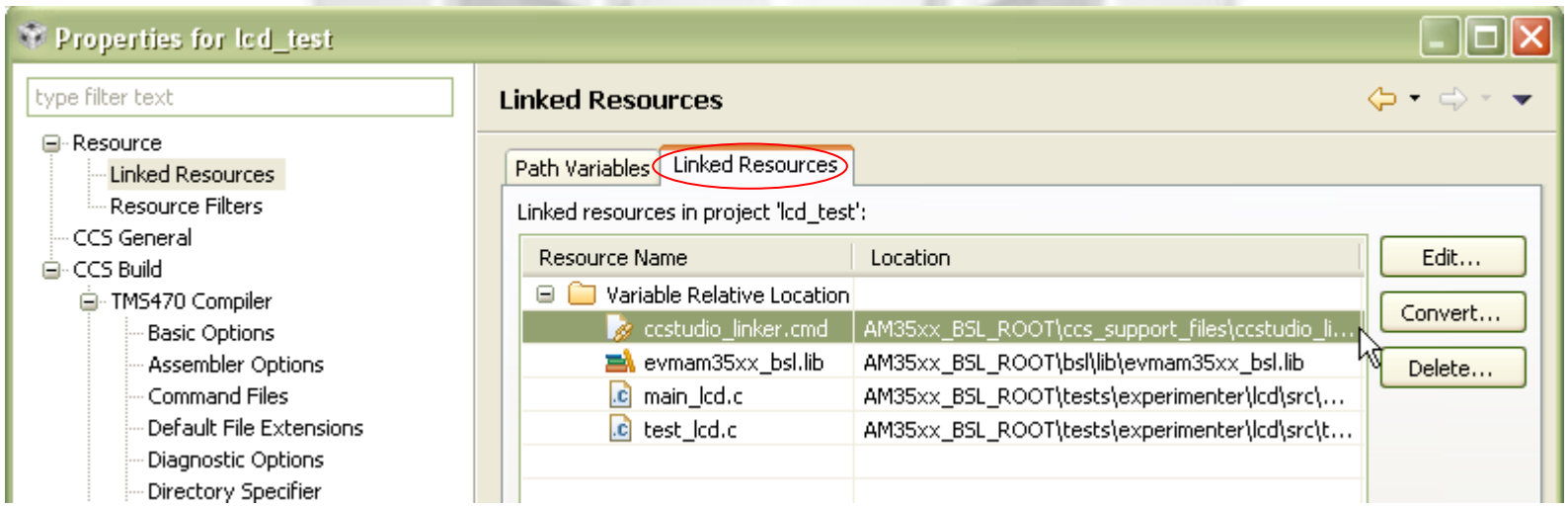
LCD Test: Project Properties

- **Go to ‘Resource -> Linked Resource’ 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)**



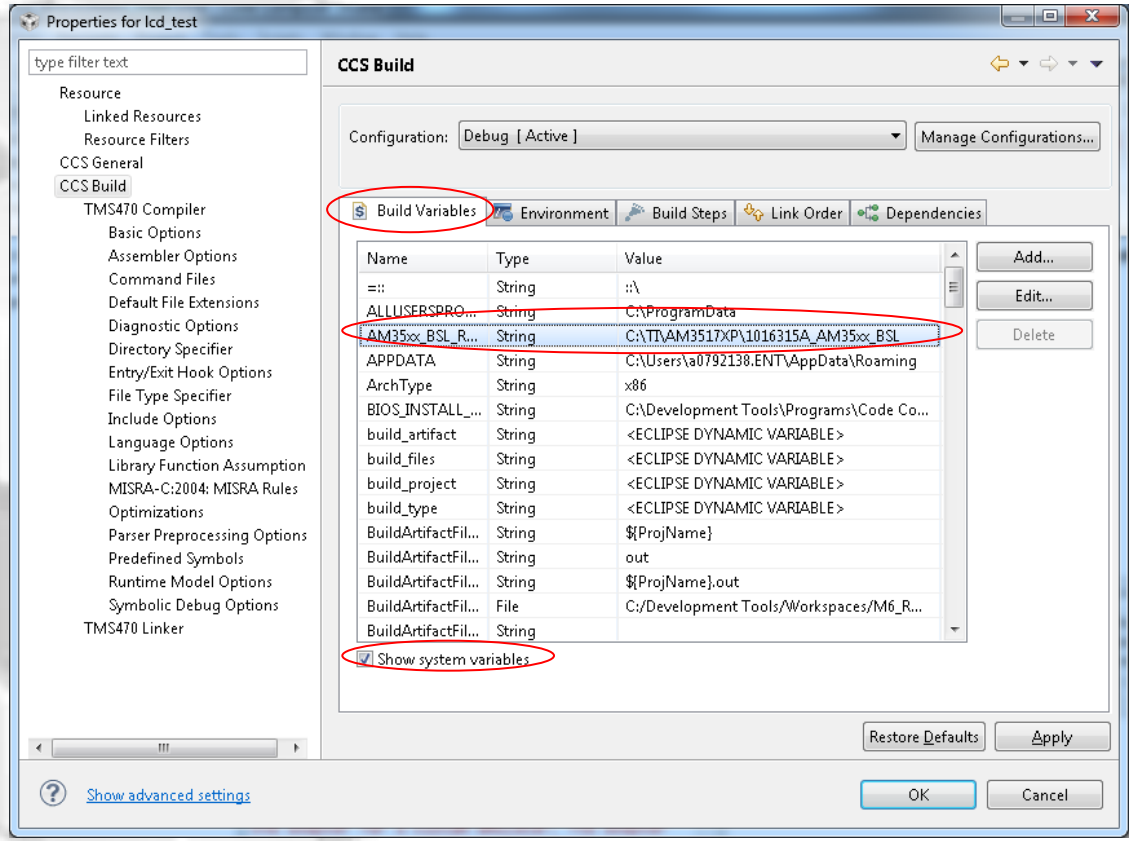
LCD Test: Project Properties

- **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**



LCD Test: 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**

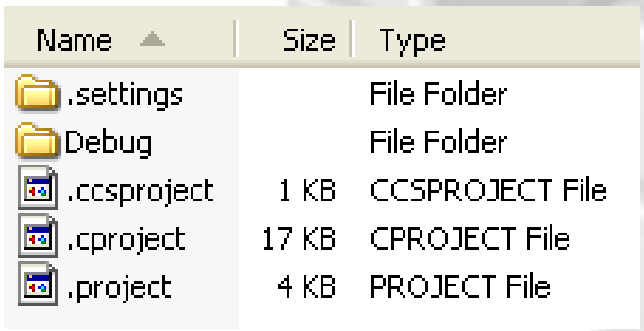


Project vs Workspace Level Variables

- **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!

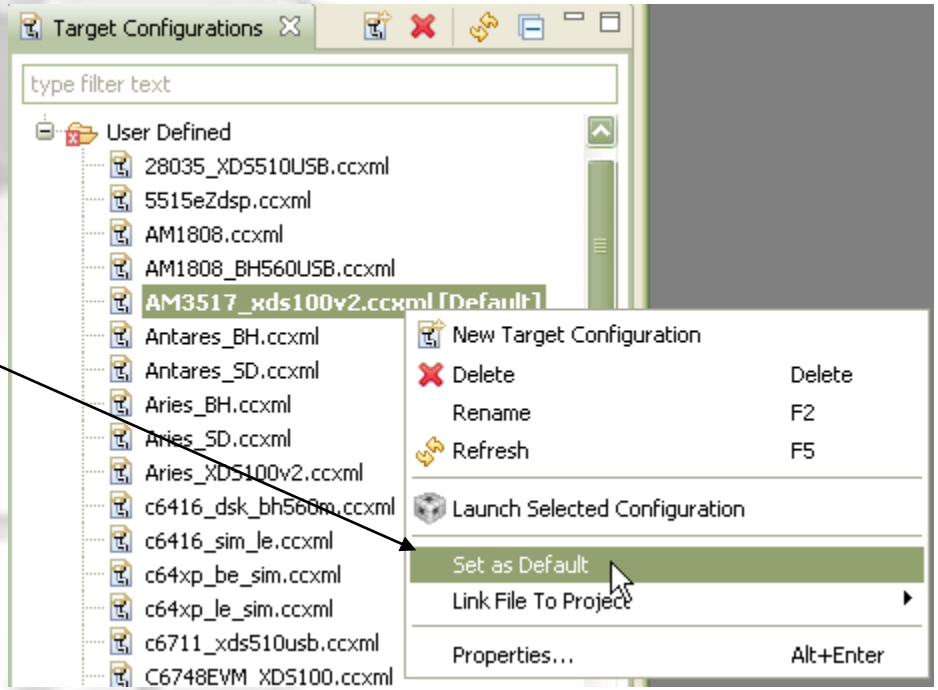
Version Control – Check in Which Files?

- **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
  - The contents of the project configuration folder (Debug/Release) do not need to be checked in



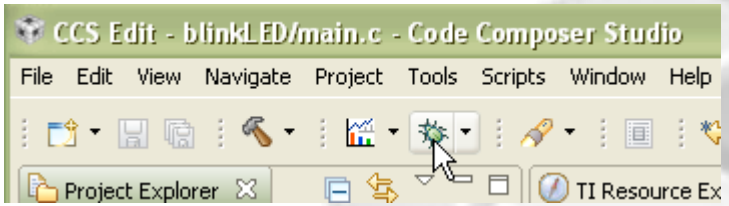
LCD Test: Target Configurations

- **Open the 'Target Configurations' view**
- **Right-click on the target configuration file for the AM3517 and 'Set as Default'**
- **This will instruct CCS to use this target configuration file if one is not defined in the project**
- **Why not just link or add it to the project?**
  - Not everyone may be using the same emulator (XDS100v2) so let people have their own target configuration file defined locally



LCD Test: Run the Project

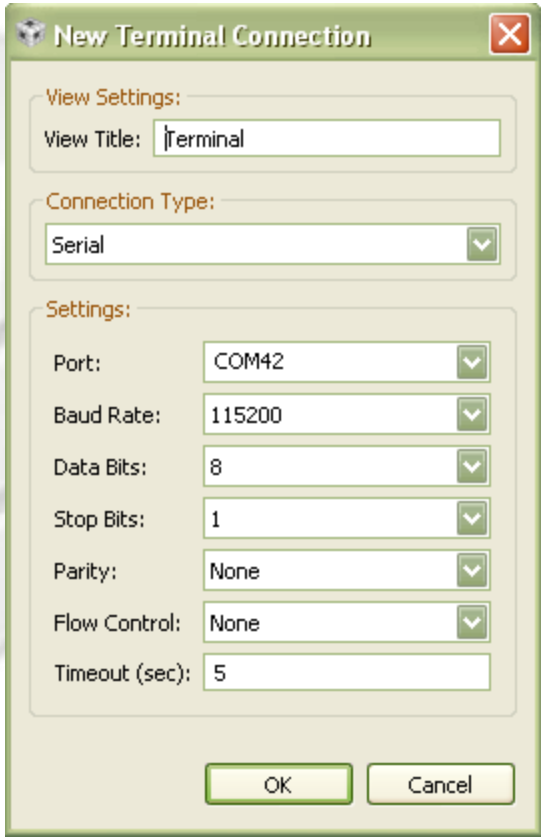
- **Use the ‘Debug’ button to:**



- **‘Debug’ button does multiple steps at once:**
  - Build the project
  - Start the debugger (CCS will switch to the ‘CCS Debug’ perspective)
  - Connect CCS to 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)**

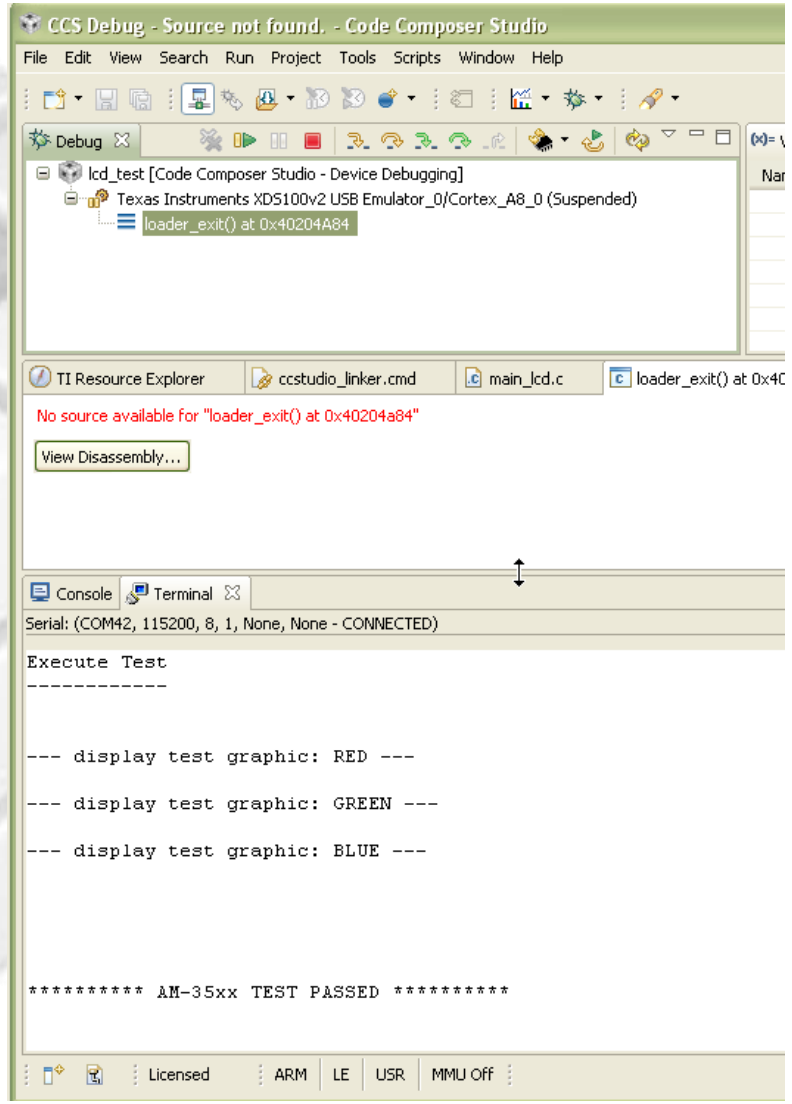
LCD Test: Run the Project

- **If everything is successful:**
  - The project built successfully
  - Debug session was started for the target (perspective switch to ‘CCS Debug’)
  - Program loaded
  - Program is at main
- **Open the ‘Terminal’ view**
  - Window -> Show View -> Other
  - Listed under Terminal
- **and start a new session with the settings as shown:**
  - Use the COM port that has a connection with the COM port on the board (may have to use Windows Device Manager to check which COM port number applies to you)



LCD Test: Run the Project

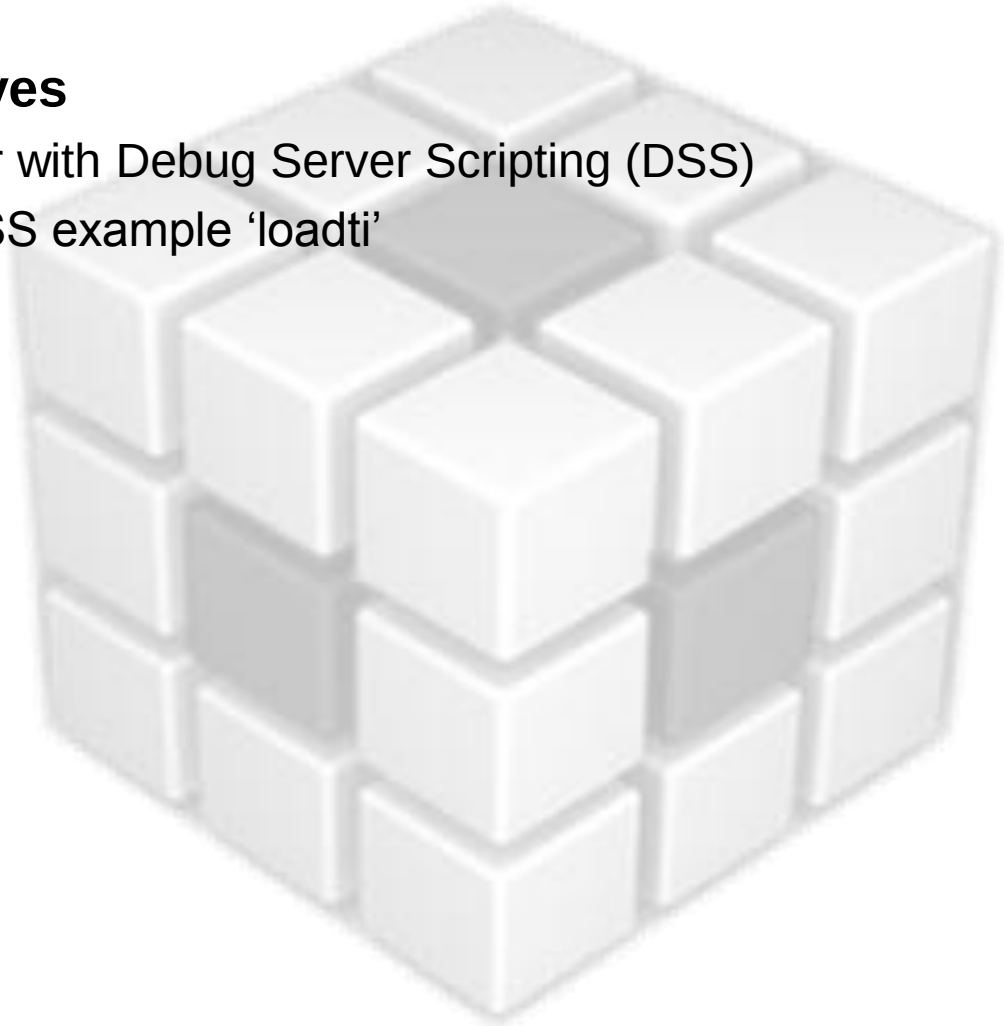
- **Press the 'run' button to run the program**
  - There should be output streaming to the 'Terminal' view
  - The LCD on the board will change color three times (RED, GREEN, BLUE)
  - Then the program will reach the exit point and be halted.



DEBUG SERVER SCRIPTING

Debug Server Scripting: Briefing

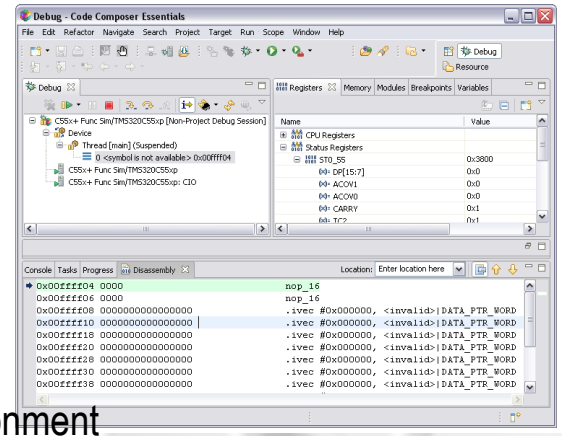
- **Key Objectives**
  - Get familiar with Debug Server Scripting (DSS)
  - Run the DSS example ‘loadti’

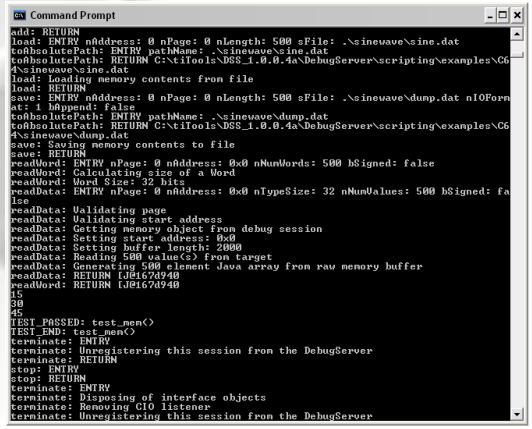


Debug Server Scripting: Overview

CCS:

Development environment based on Eclipse





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: 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

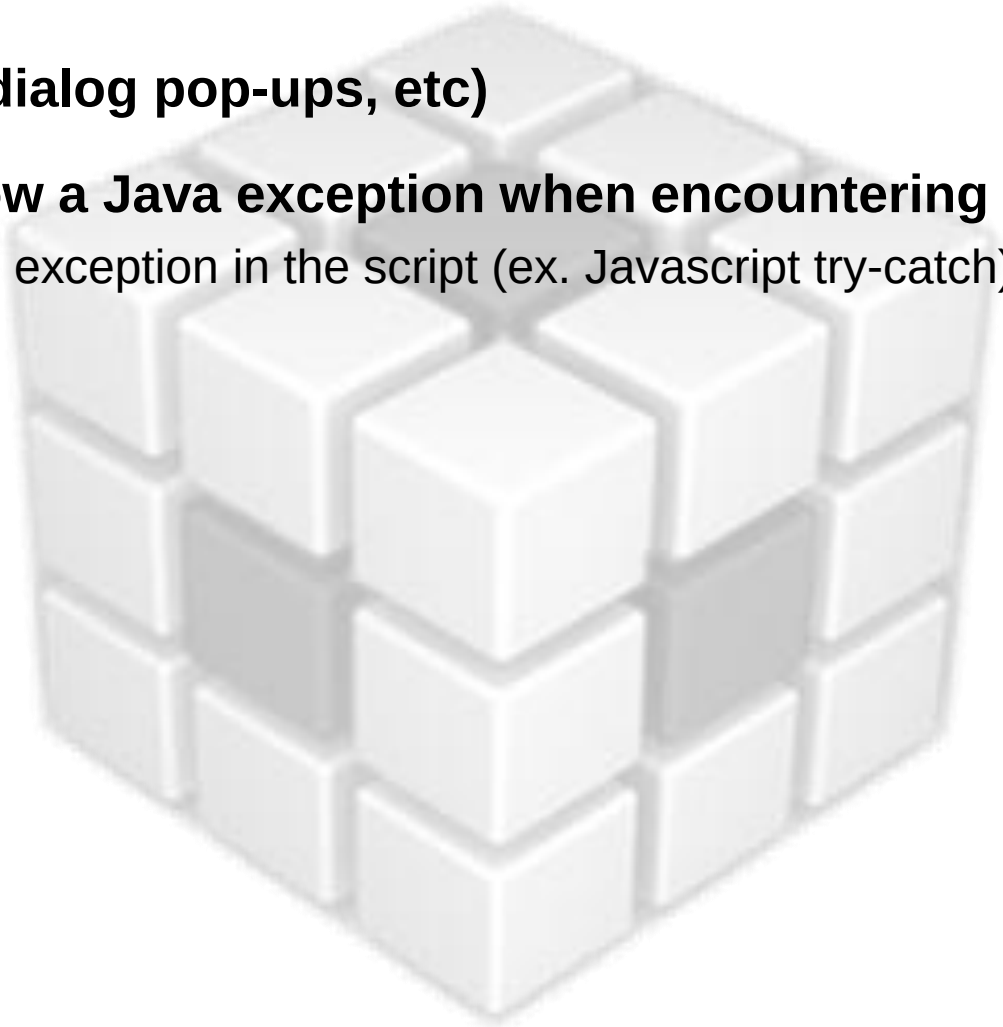
Debug Server Log

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

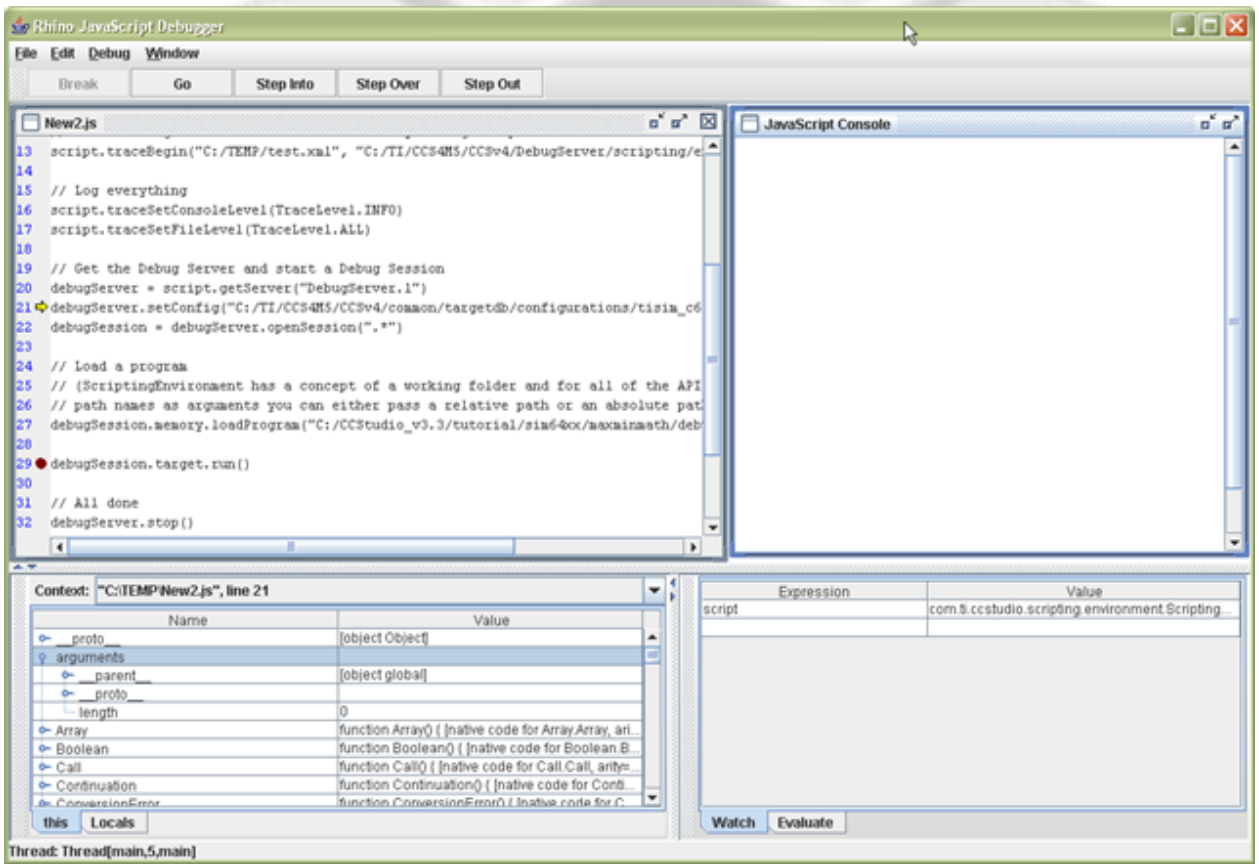
- DSS APIs throw Java exceptions

- 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.xsl");

// 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 ‘led_test.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>\lcd_test\Debug
- **`> 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 -n -c C:\TI\AM3517XP\am3517_xds100v2.ccxml -o led_test.out`

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

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

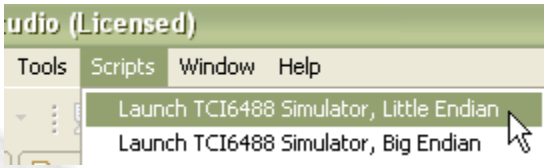
- **View- > Scripting Console**
- **Press TAB for a list of commands**
  - Press TAB for partially typed commands for auto-complete feature
- **To get documentation for a command**
  - js:> help <command>

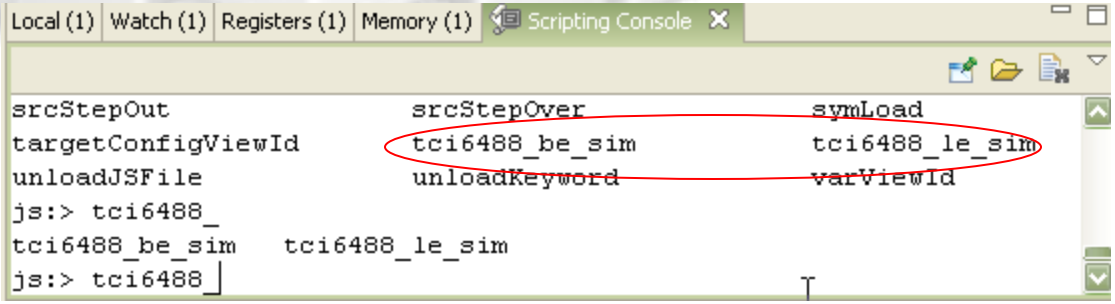
Scripting Console

- **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

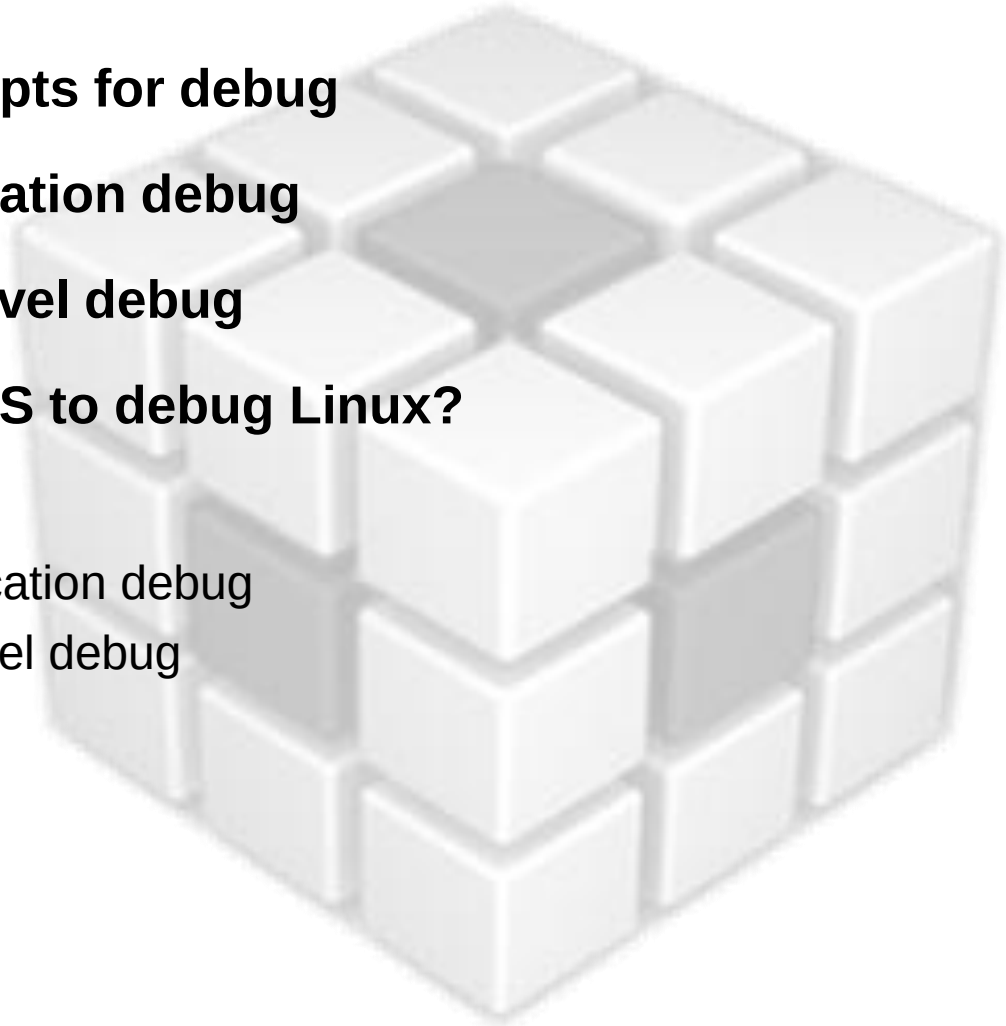
```
// Add entries to the 'Scripts' menu
hotmenu.addJSFunction("Launch TCI6488 Simulator, Little Endian", "tci6488_le_sim()");
hotmenu.addJSFunction("Launch TCI6488 Simulator, Big Endian", "tci6488_be_sim()");

// 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");
}

...
```





LINUX DEBUGGING

Agenda



- **Linux concepts for debug**
- **Linux application debug**
- **Linux low-level debug**
- **Why use CCS to debug Linux?**
- **Lab**
  - GDB application debug
  - JTAG Kernel debug
- **Summary**

Linux concepts for debug

- **Linux is an open-source High Level Operating System (HLOS) initially developed for PCs but has been widely adopted by TI in its System-on-a-chip (SoC) embedded processors**
- **The differences between the PC and the embedded environments impose a different approach to the debugging process:**
  - PCs have a more standard baseline hardware and more system resources (CPU, memory, peripherals, etc.), thus requiring very little development and optimization of low-level device drivers

Linux concepts for debug

- **Therefore debugging an embedded Linux system is very different for both types of developers**
  - For experienced embedded developers familiar with low-level debugging, the use of a JTAG debugger is too raw to be useful in such a complex system. The majority of debugging happens in User Mode
  - For experienced Linux developers familiar with high-level debuggers (like GDB), the debugging process may require visibility into lower levels of the system that are only visible in Kernel Mode

Linux concepts for debug

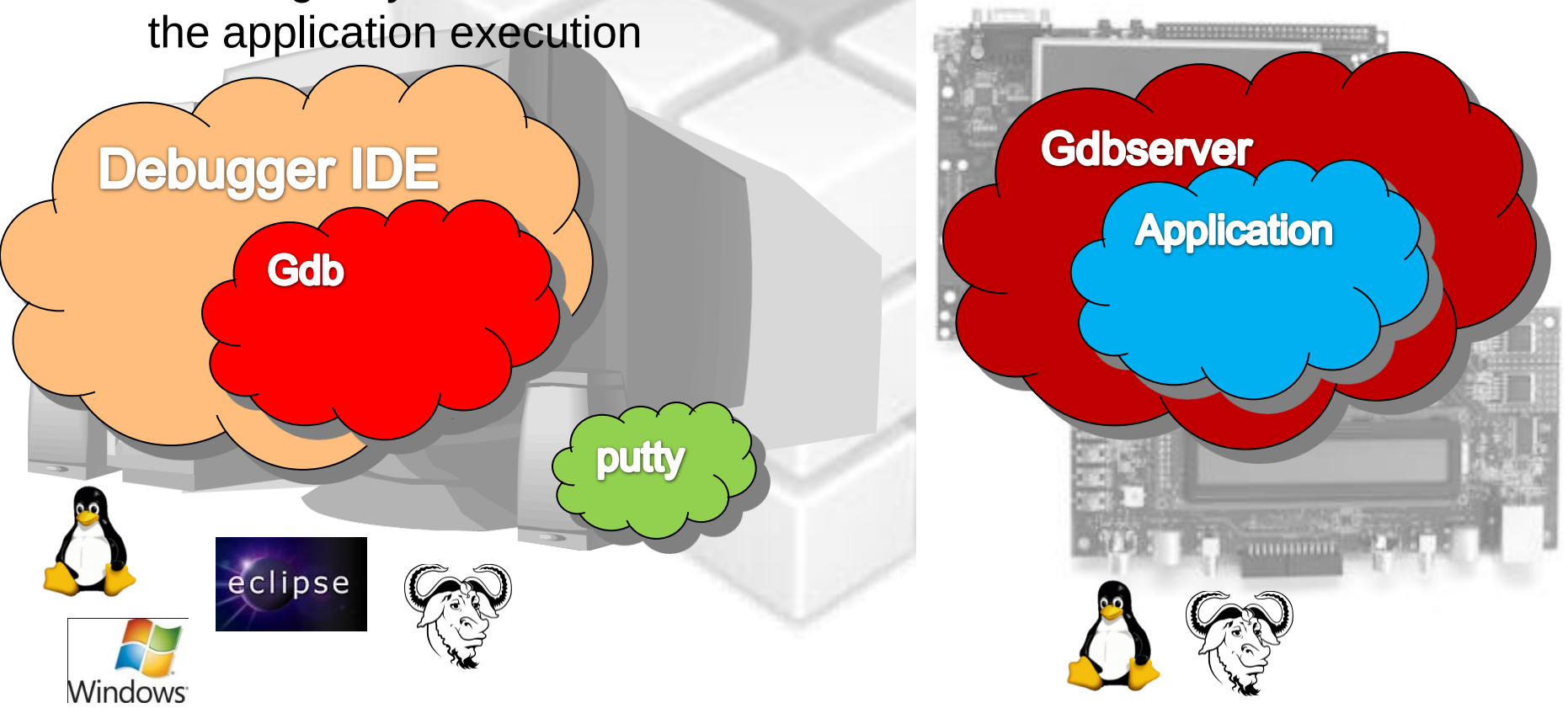
- **The modes mentioned before affect the ability to debug the components in a typical Linux system:**
- **User mode: a contained runtime environment designed to execute applications (what we see in a typical command prompt) but with controlled access to critical system functions and hardware**
  - The access is done via device drivers
- **Kernel mode: a non-contained mode designed to execute the core kernel and device drivers.**
  - This mode allows complete access to hardware and critical functions of the system, therefore Linux is highly protective of it.

Linux application debug

- **Linux Application debugging is the most common scenario among developers**
  - The application is the focus of product development
- **GDB is a popular tool to perform application development, comprised of two components:**
  - **Gdbserver** that runs on the target system (board) and is responsible for loading the application to be monitored and responds to commands from the **Gdb** component
  - **Gdb** that runs on the debug system (PC) and allows the developer to issue commands that control the **Gdbserver** component

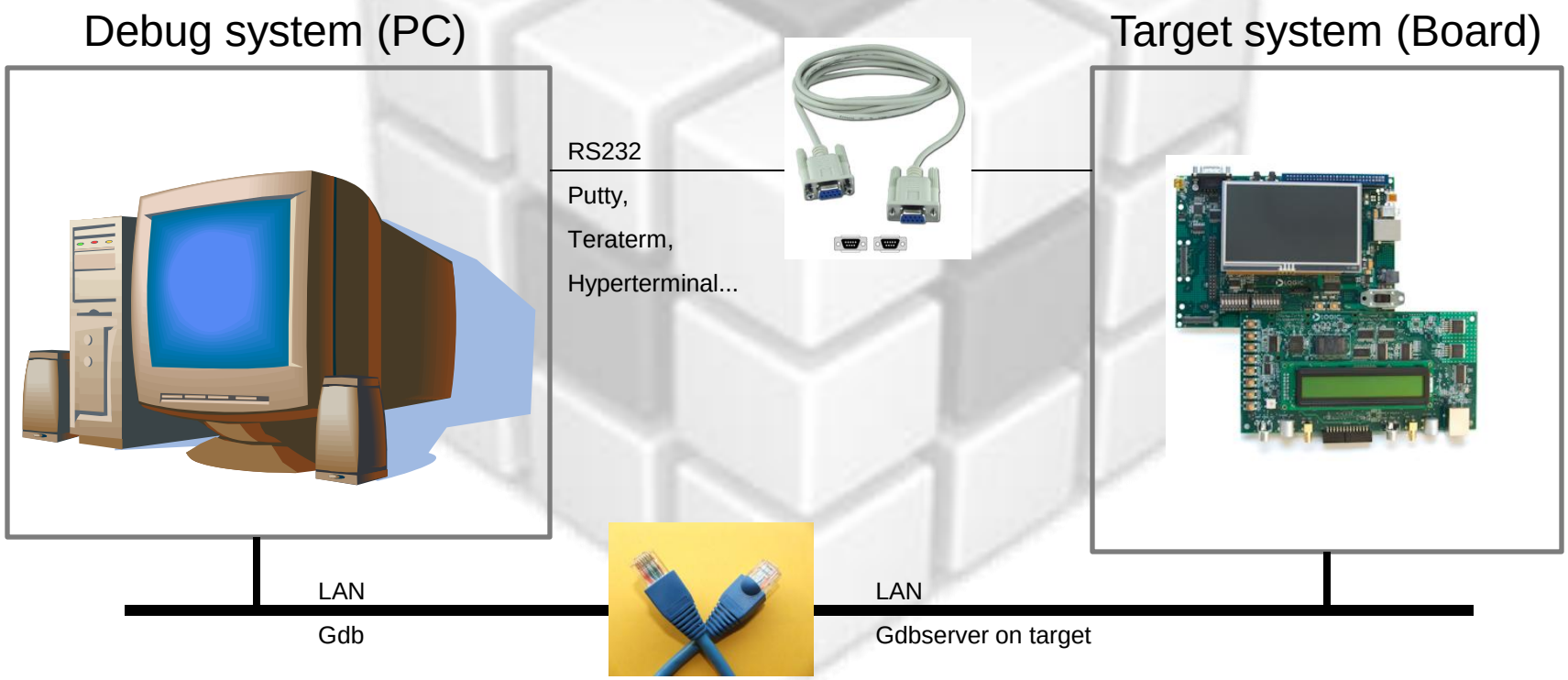
Linux application debug

- The debug system runs its own OS, a debugger IDE (optional), Gdb and a terminal application (hyperterminal, putty, etc.) to control the application environment
- The target system runs the embedded OS and Gdbserver that controls the application execution



Linux Application Debug Setup

- The connections of a typical Linux application debug system are shown below.



Linux application debug

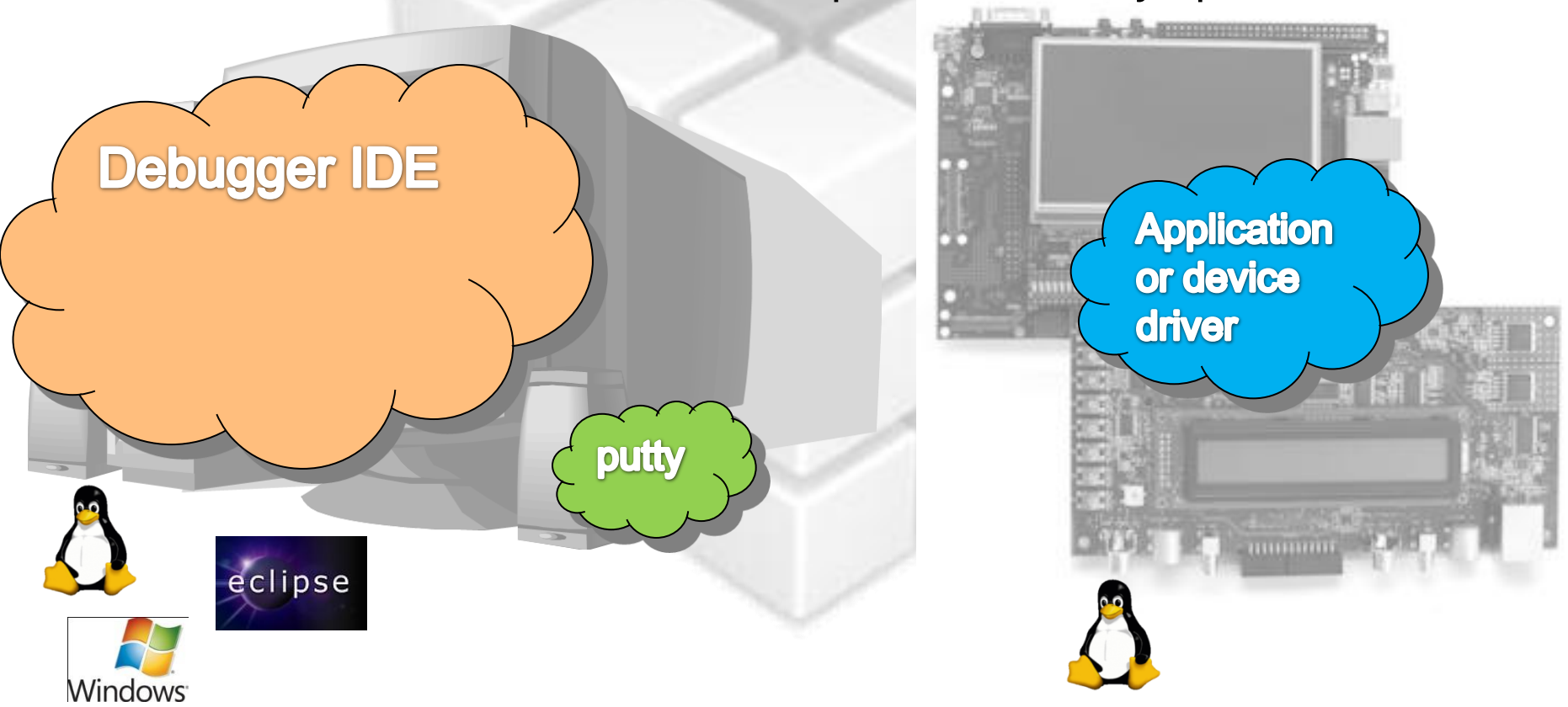
- **Linux application debugging can be done using a standard open source Eclipse CDT IDE running on the Debug system**
  - Natively supports GDB debugger
- **It is designed for application development using GNU toolchain, allowing source code debugging**
  - Linux applications are usually developed with GNU
- **Its environment is easily extensible via plugins that increase the debugging process:**
  - High-level debuggers for Android, Mylyn, Qt...
  - Productivity tools like a terminal application

Linux low-level debug

- **Kernel and device driver debug require access to the Kernel Mode**
  - Gdb and Gdbserver are not well suited for this task
- **The addition of an external JTAG emulator grants complete access to the embedded processor peripherals, registers and memory**
  - Even below the Linux Kernel Mode!
- **The standard Eclipse CDT IDE does not have support for low-level debugging**
  - It is heavily dependent on both the emulation technology and the embedded processor being used

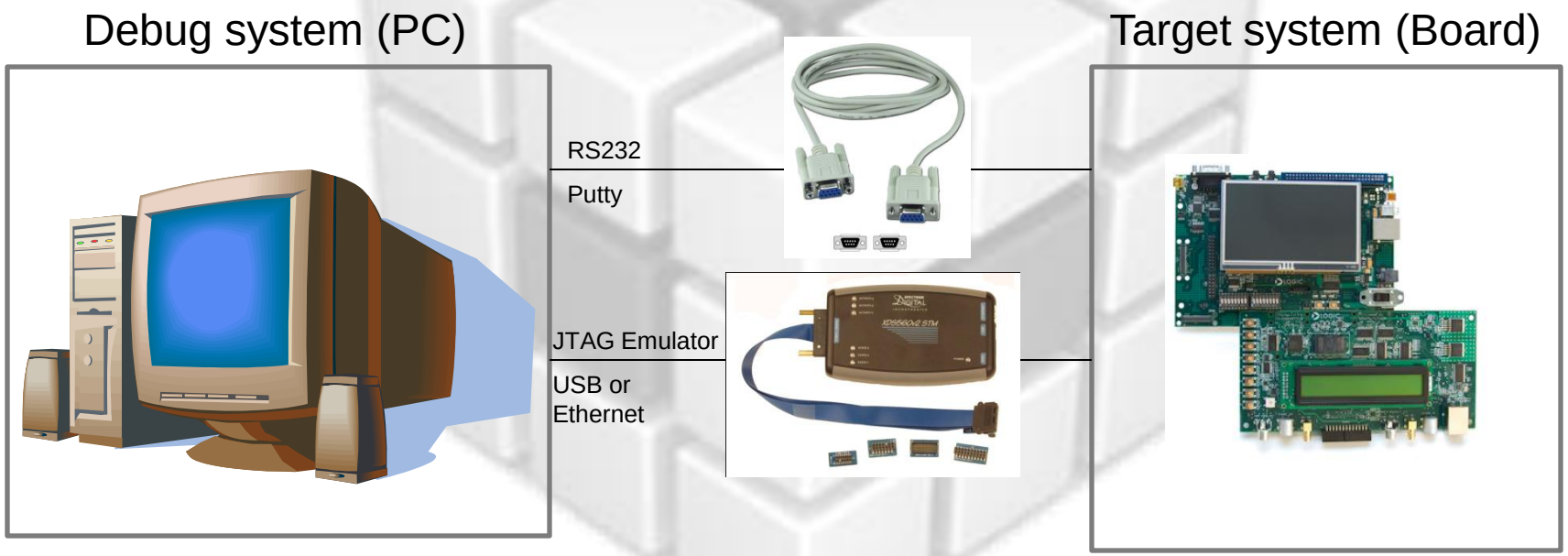
Linux low-level debug

- The debug system runs its own OS and a debugger IDE that supports a JTAG emulator. A terminal application (e.g., putty) is optional.
- The target system runs the embedded OS. Running any application or custom device driver under development is entirely optional.



Linux low-level debug setup

- Physically the connections of a typical linux low-level debug system are shown below.



Why use CCS to debug Linux?

- **CCSv5 addresses both scenarios of Linux debug:**
- **Application debugging:**
  - It is based on the latest release of the standard Eclipse CDT IDE and keeps its standard functionality, including the support for the GNU toolchain
  - Much more compatible with third-party plug-ins
- **Low-level debugging:**
  - It is compatible with several JTAG emulators
  - XDS100v2 if low cost is desired
  - XDS560v2 USB/Ethernet if remote debugging is desired

Why use CCS to debug Linux?

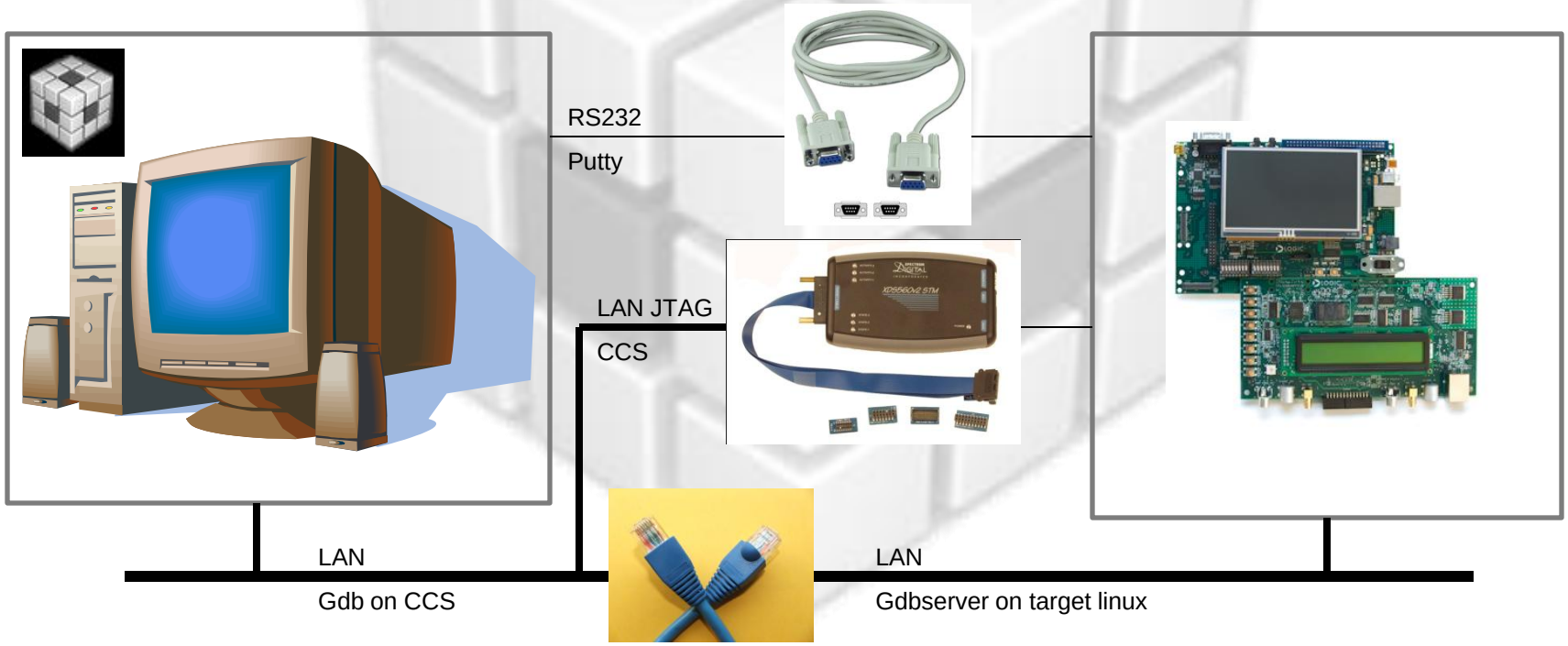
- **Simultaneous application and low-level debug**
  - Debug the kernel in conjunction with an application
- **It allows simultaneous debugging of ARM and DSPs/coprocessors**
  - Useful to debug custom codecs running on Codec Engine or low-level DSP code running on DSP/Link
- **The debug plug-ins available for other OSes (Android, Mylyn, etc.) allow a one-stop-shop debugging environment**

Why use CCS to debug Linux?

- **Gives you source-level application debug capabilities on the ARM**
  - Breakpoints, stepping, memory view, expressions…
- **Linux version is available:**
  - Easier to develop code with TI's Linux-based SDKs.
    - http://processors.wiki.ti.com/index.php/Linux_Host_Support
- **Third party plug-ins available:**
  - Terminal view
    - http://processors.wiki.ti.com/index.php/How_to_install_the_terminal_plugin_in_CCSv5
  - Eclipse Linux Tools Project
    - http://www.eclipse.org/linuxtools/

Linux debug setup

- **Demo configuration:**
  - Current build of v5 Linux on Ubuntu 10.04
  - JTAG emulator connected to the LAN
  - Board running Linux



Run Mode Debug

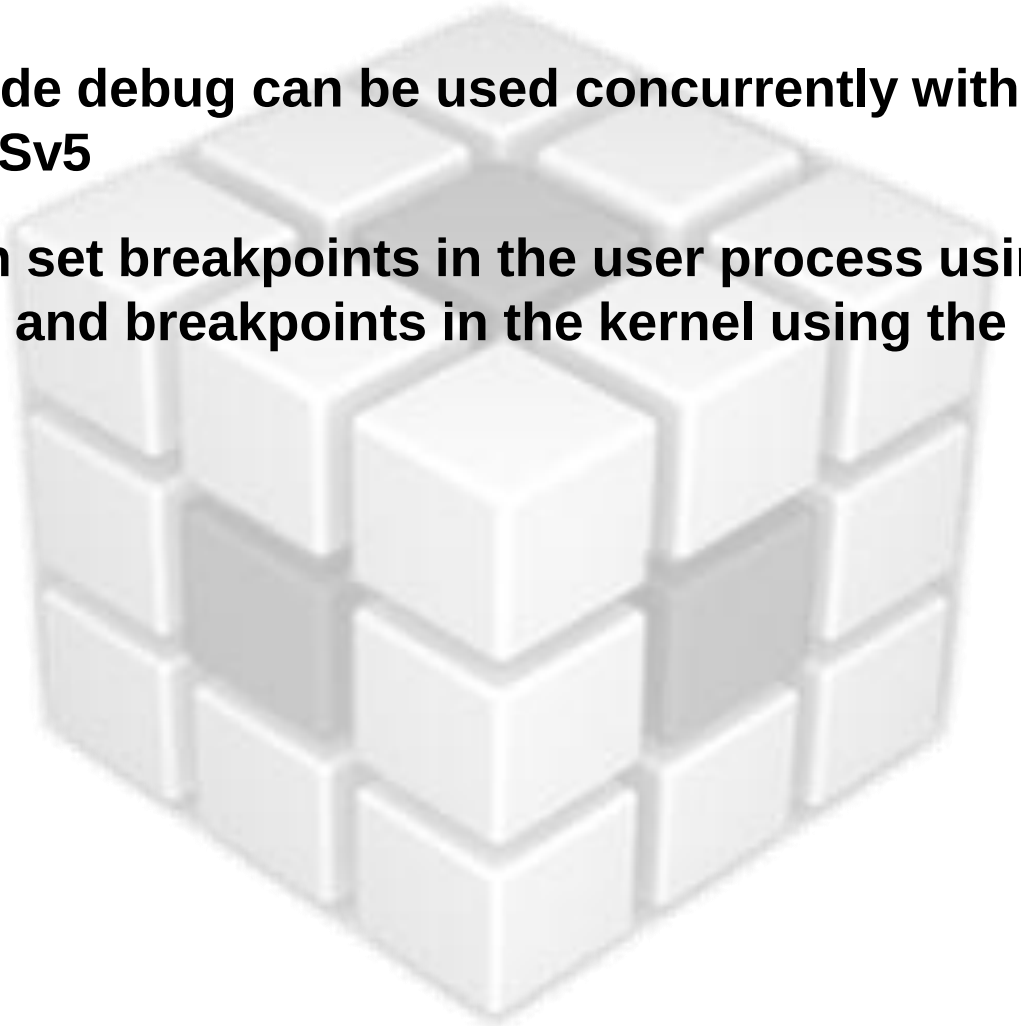
- **Dependencies**
  - CCS versions: CCS 5.0 or greater
  - Devices: Cortex A8, ARM926 (may work on other ARM devices)
  - Host requirement: a cross platform GDB debugger
  - Target requirement:
    - GDB server that is compatible with the host GDB debugger
    - Two connections to the target system
      - One connection, typically, via a serial port, to the target console is used to execute Linux commands
      - The other connection, typically, via an ethernet port, is used by the gdb debugger to communicate with the gdb server running on the target

Stop Mode Debug

- **Dependencies**
  - CCS versions: CCS 5.0 or greater
  - Devices: Cortex A8, ARM926 (may work on other ARM devices)
  - Target Linux version: 2.6
  - JTAG connection to the target system. An additional connection to the target console may be helpful.

Mixed Mode Debug

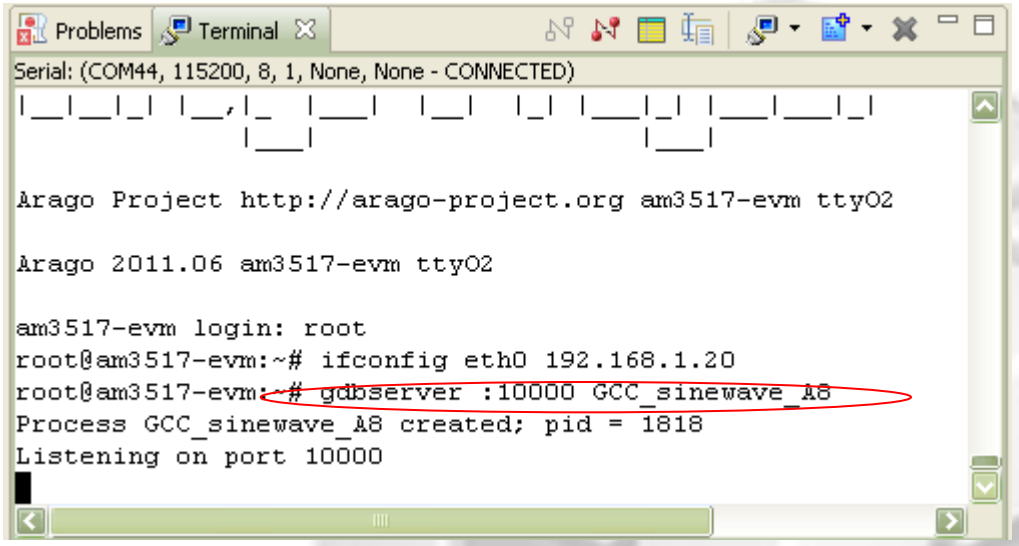
- The stop mode debug can be used concurrently with the run mode debug in CCSv5
- The user can set breakpoints in the user process using the run mode debug and breakpoints in the kernel using the stop mode debug



Configure U-boot

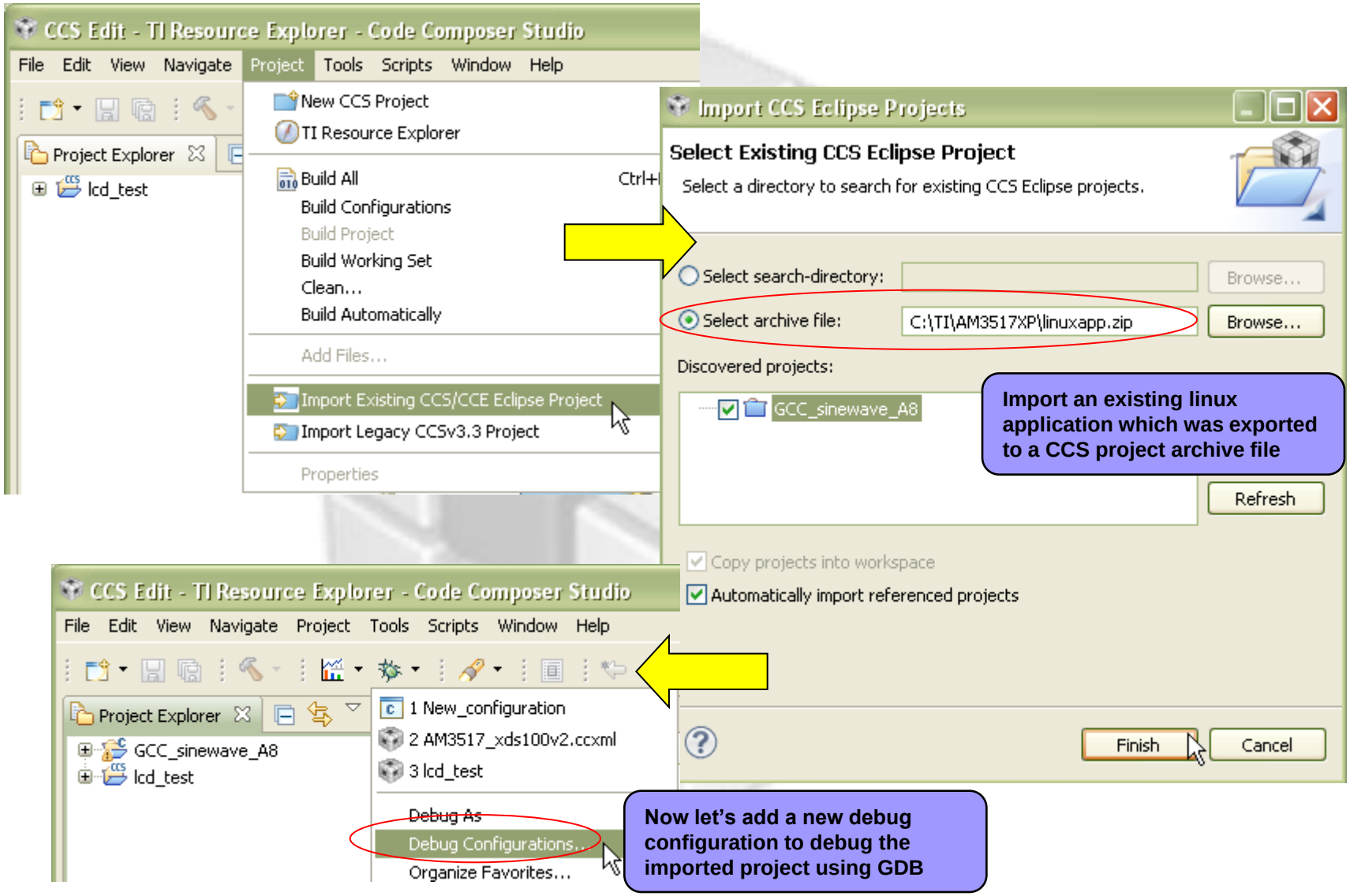
- Open Terminal view
- Turn off board
- Insert SD card
- Follow steps 3 and 5 as documented in:
  - **C:\TI\AM3517XP\AM3517evm_uboot_sdcard.txt**

Run Mode Debug: Start GDB Server

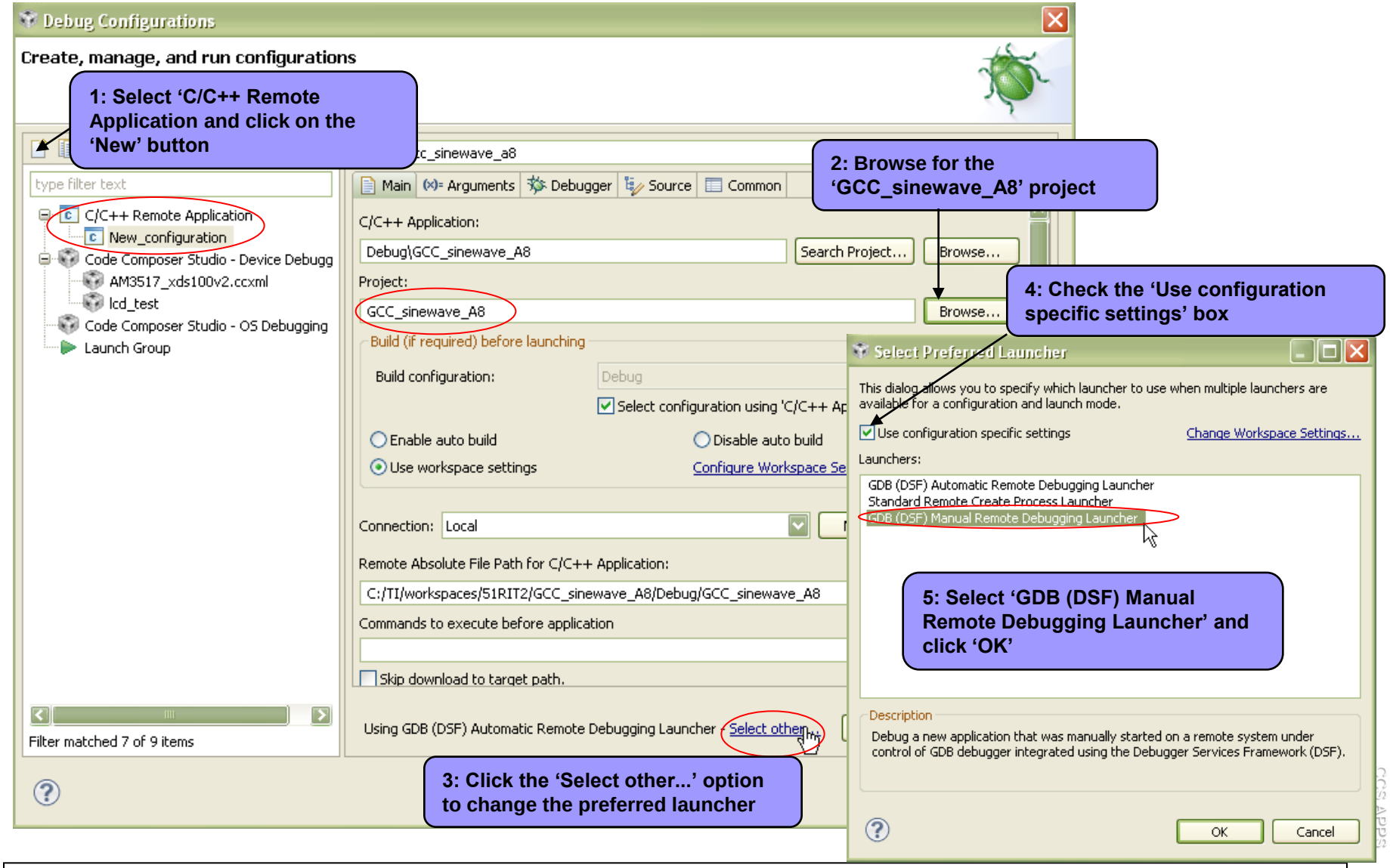


- If using ethernet crossover cable:
  - Set a static IP address of 192.168.1.20 for the AM3517 EVM
  - **`ifconfig eth0 192.168.1.20`**
- Start the GDB server via target console
- Use port number 10000

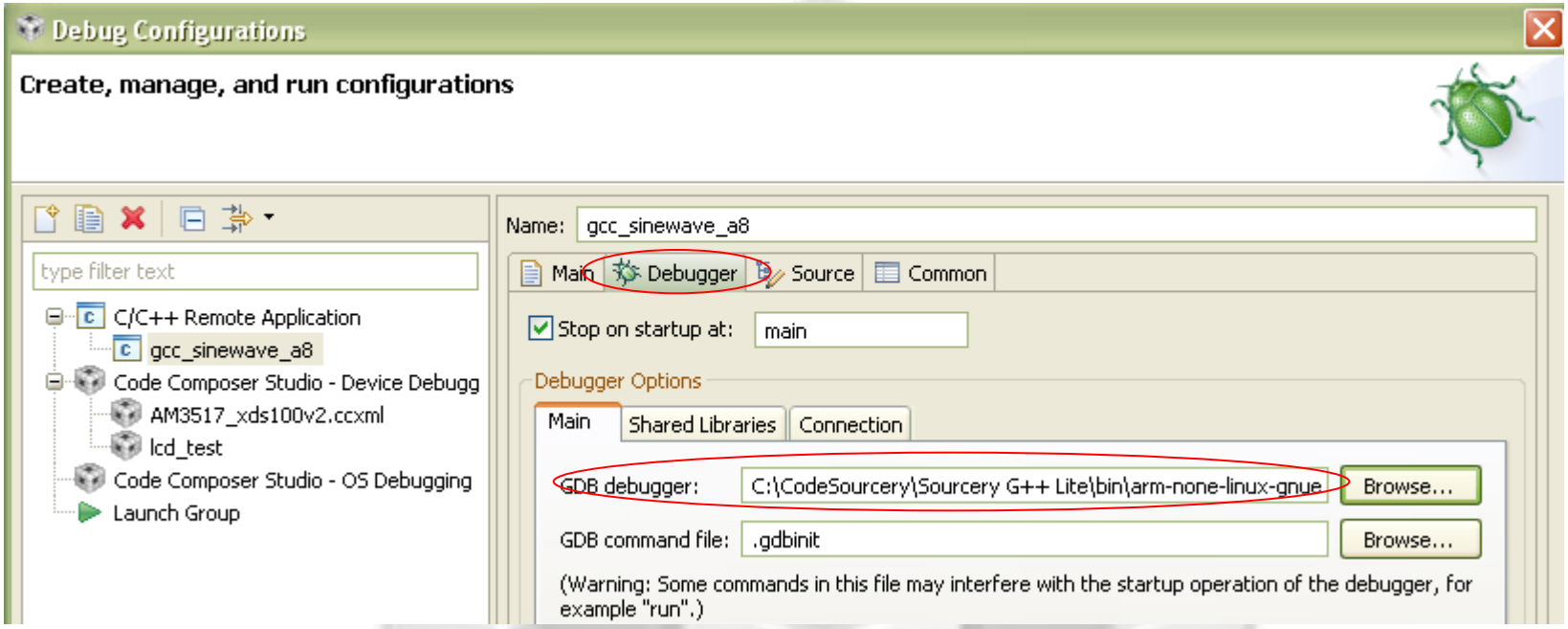
Run Mode Debug: Import Project



Run Mode Debug: Select Launcher

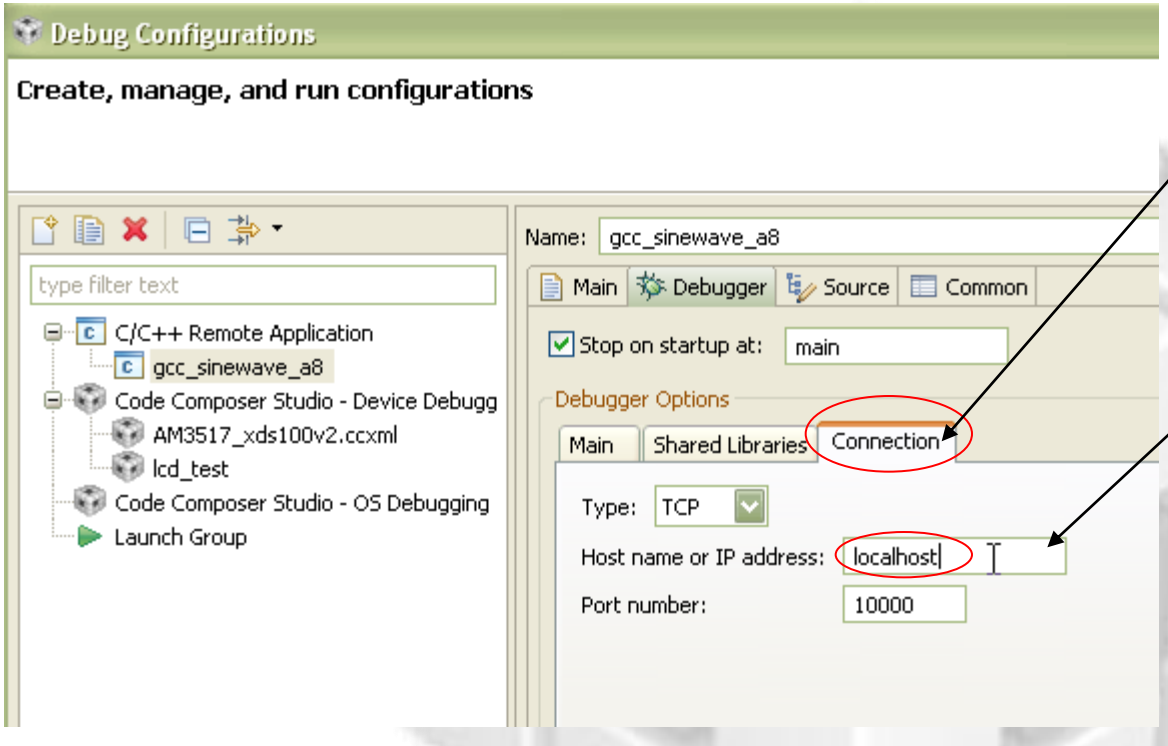


Run Mode Debug: Specify GDB Debugger



- Specify the GDB Debugger to use:
  - Browse to the location of the' arm-none-linux-gnueabi-gdb.exe' in the CodeSourcery installation

Run Mode Debug: Specify Board IP Address

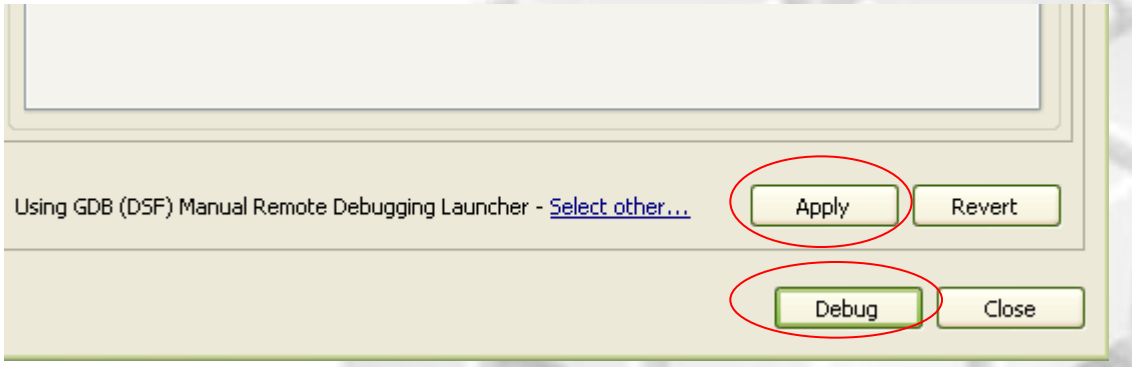


- Go to the 'Connection' tab of the 'Debugger' tab
- Enter the IP address of the target running gdbserver

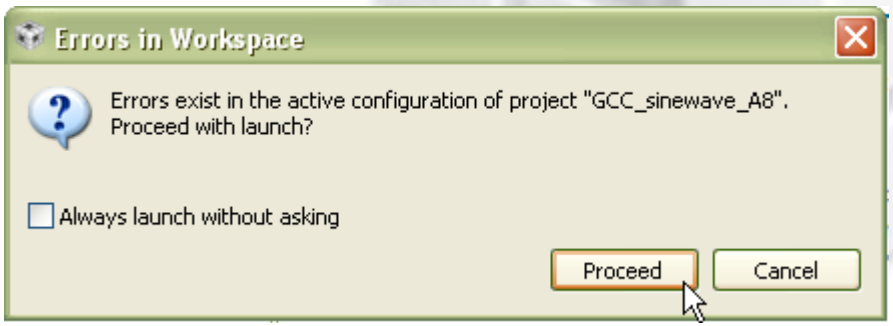
- Find out by tying in the Terminal view:
  - > **`ifconfig | grep inet`**
- If using ethernet crossover cable:
  - 192.168.1.20

Run Mode Debug: Launch CCS Debugger

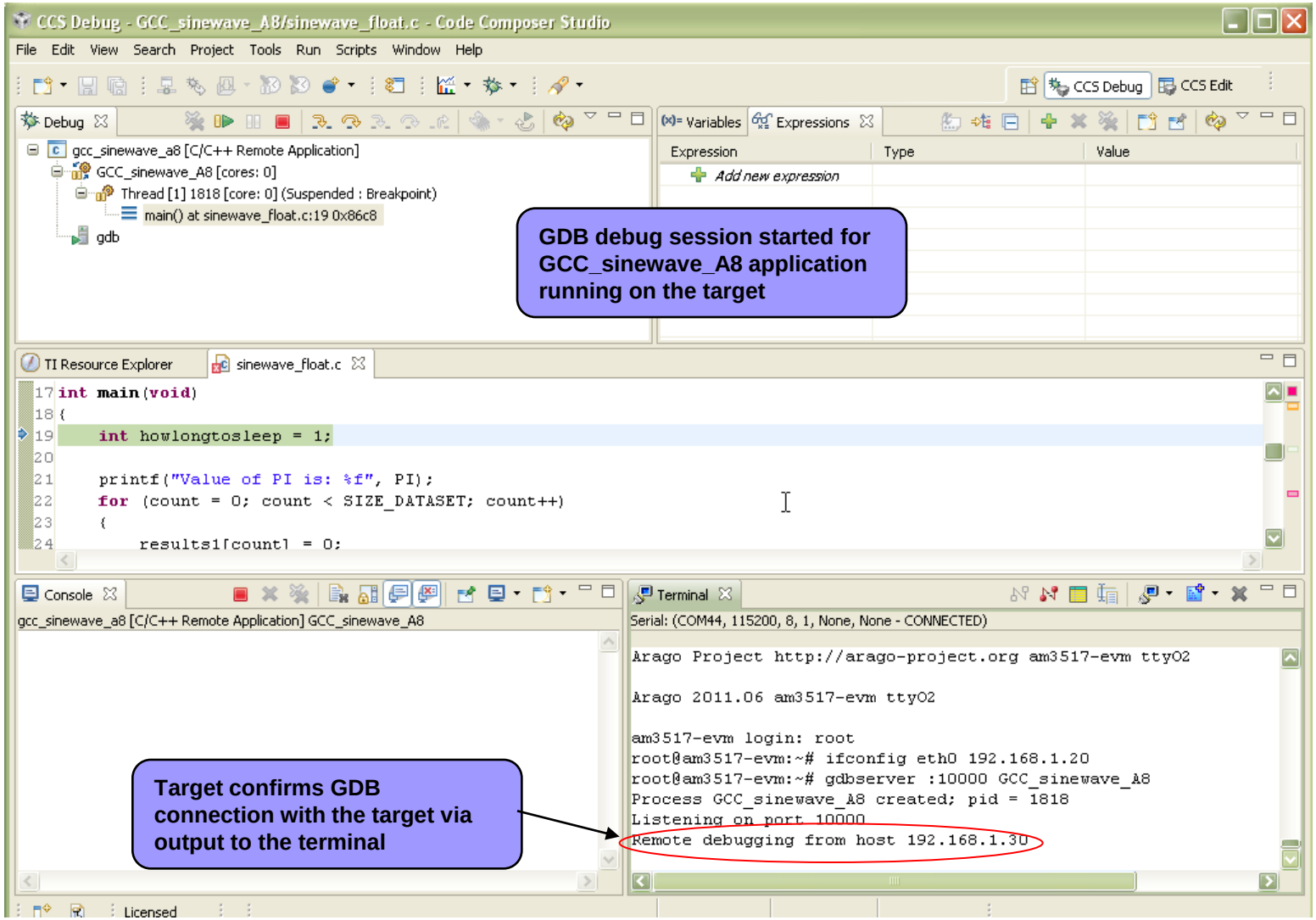
- **Press ‘Apply’ and then ‘Debug’ to start the GDB Debugger**



- **The below error can be safely ignored when launching by pressing ‘Proceed’**

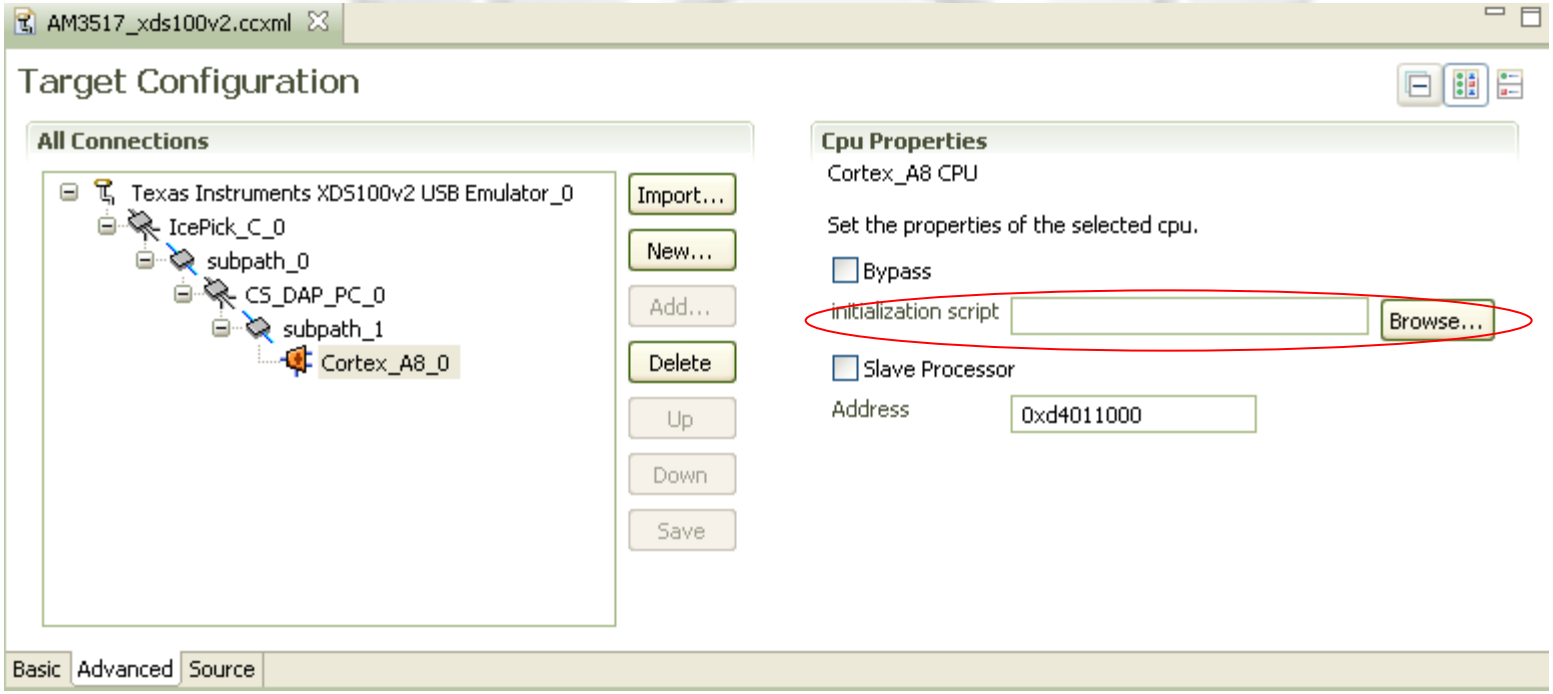


Run Mode Debug



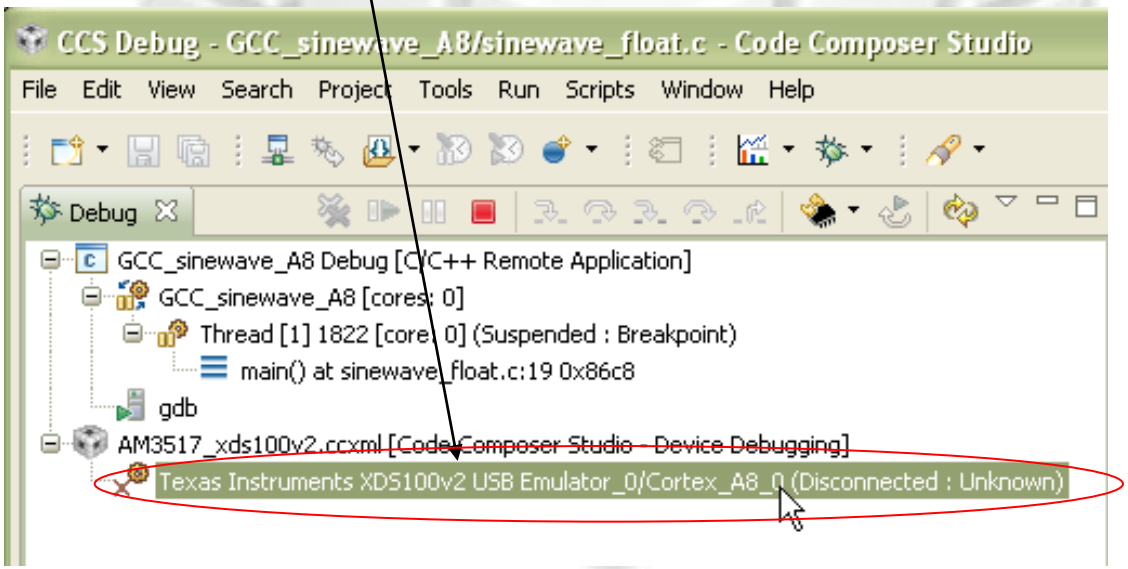
Stop Mode Debug: Modify Target Configuration File

- **Target initialization has already been done by the kernel running on the target so no need to reinitialize the target with the startup GEL file**
- **Open the Target Configuration file and modify it so no startup GEL file is used**

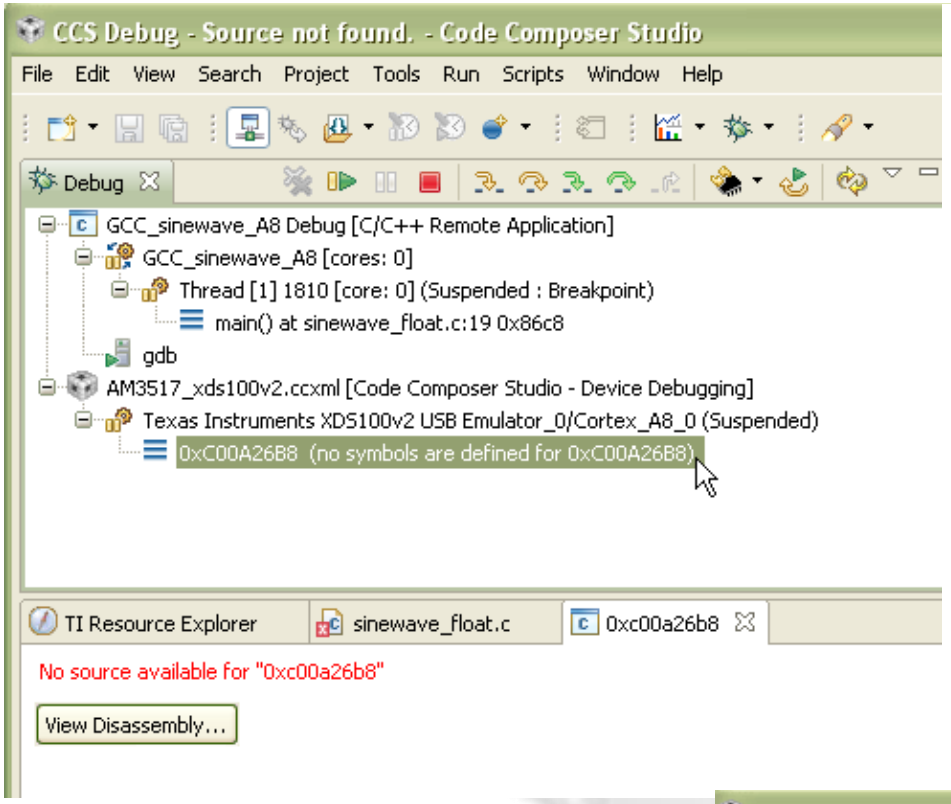


Stop Mode Debug: Launch CCS Debugger

- **Start a CCS debug session**
  - Use the Target Configuration view
- **Connect to the target**
  - Pay attention to the focus/context!
    - Make sure the node for CCS debug session is selected

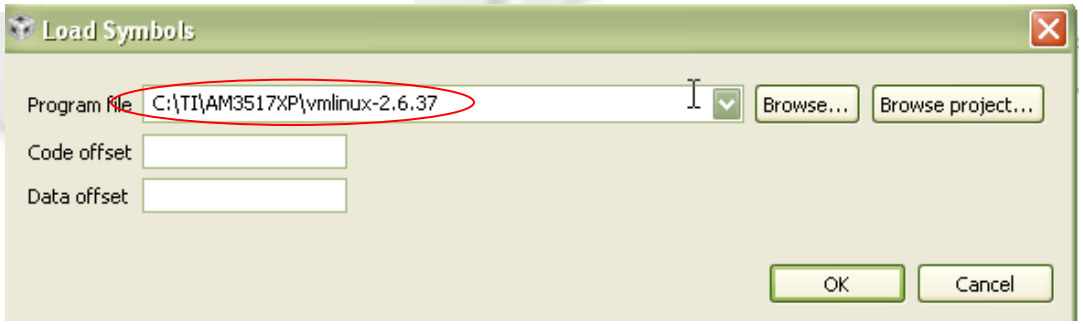


Stop Mode Debug: Load Debug Symbols



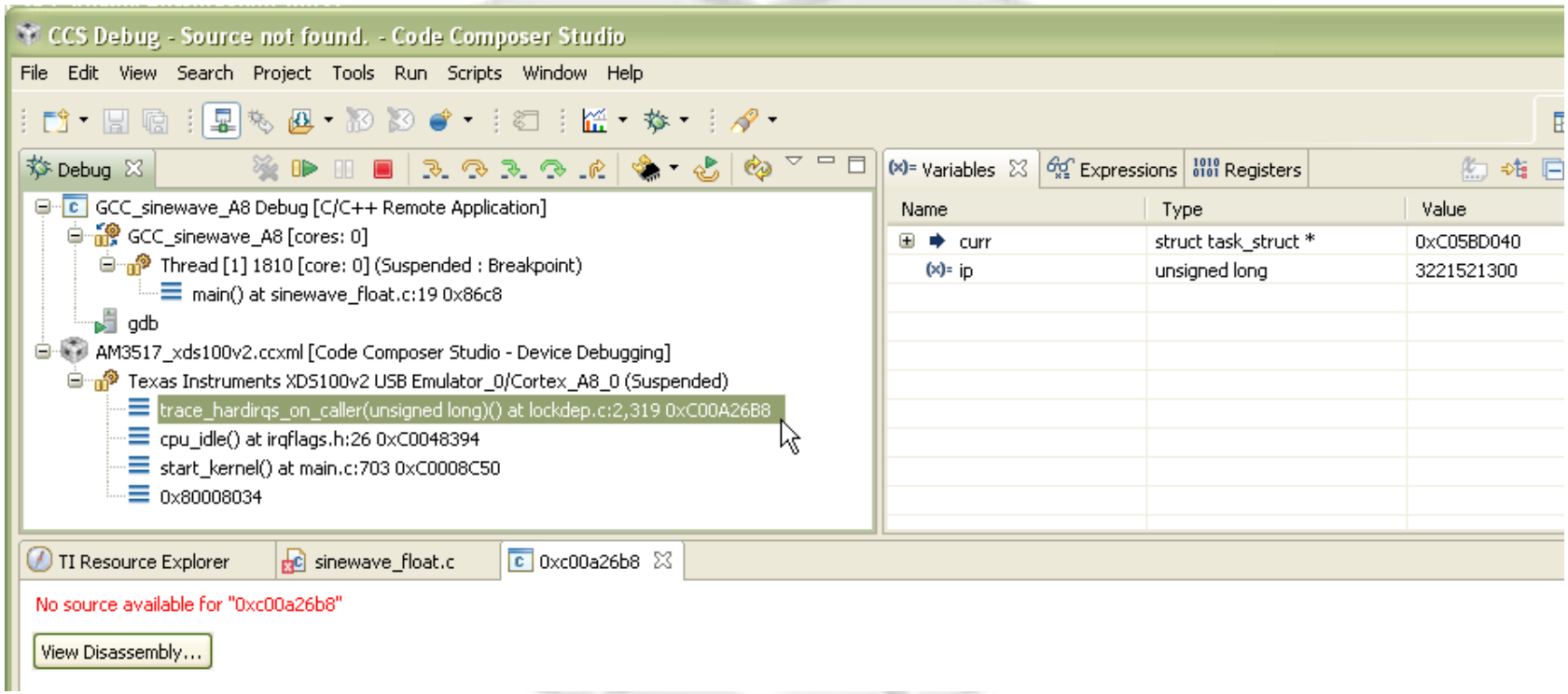
- **Target is connected but no symbols have been loaded**

- **Load Linux kernel debug symbols**
  - Run -> Load -> Load Symbols



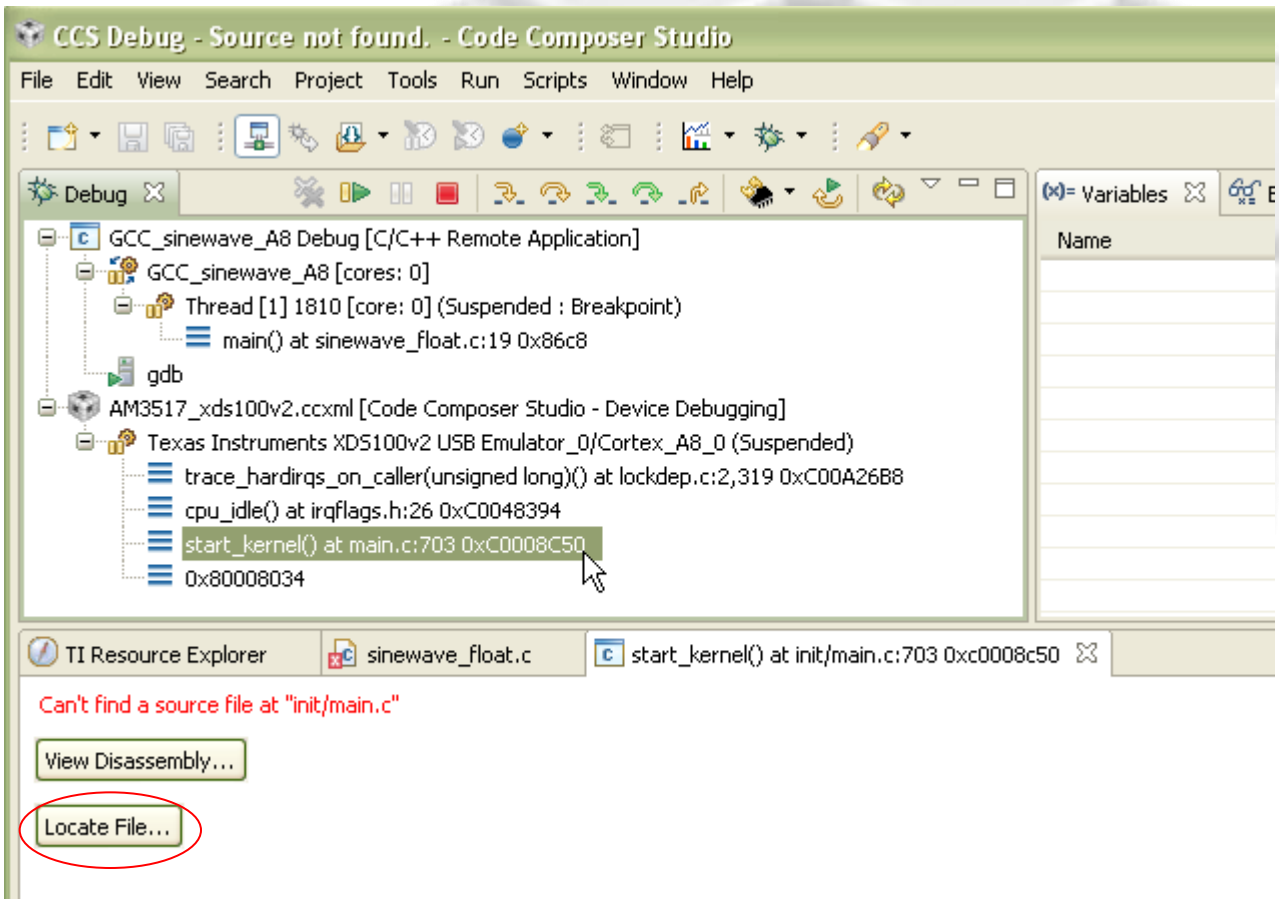
Stop Mode Debug: Source File Search Paths

- **Linux kernel symbols have been loaded but CCS does not know where to look for source files**

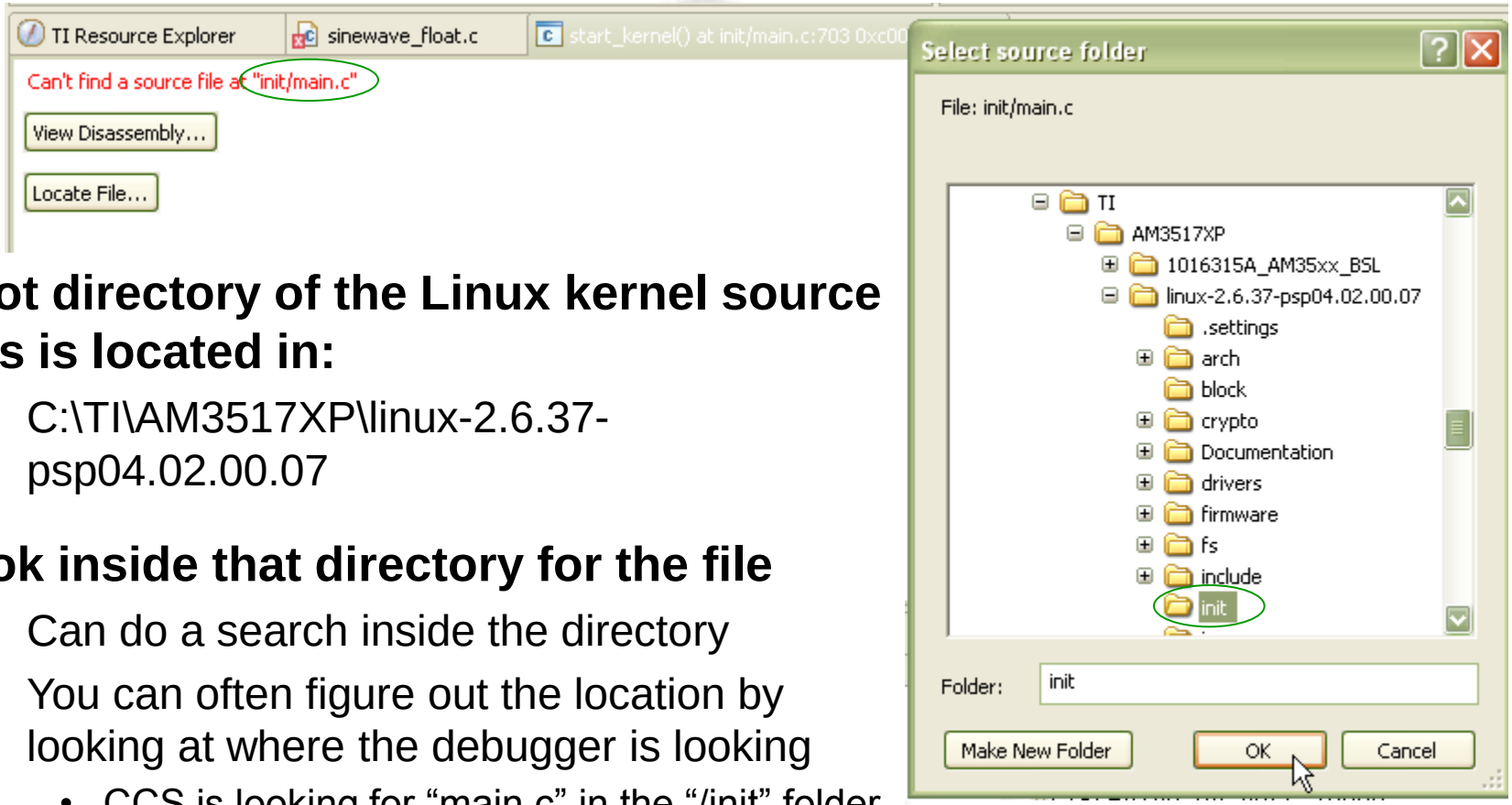


Stop Mode Debug: Source File Search Paths

- CCS will try to find source files and complain
- Use the ‘Locate File’ button and browse to the missing file

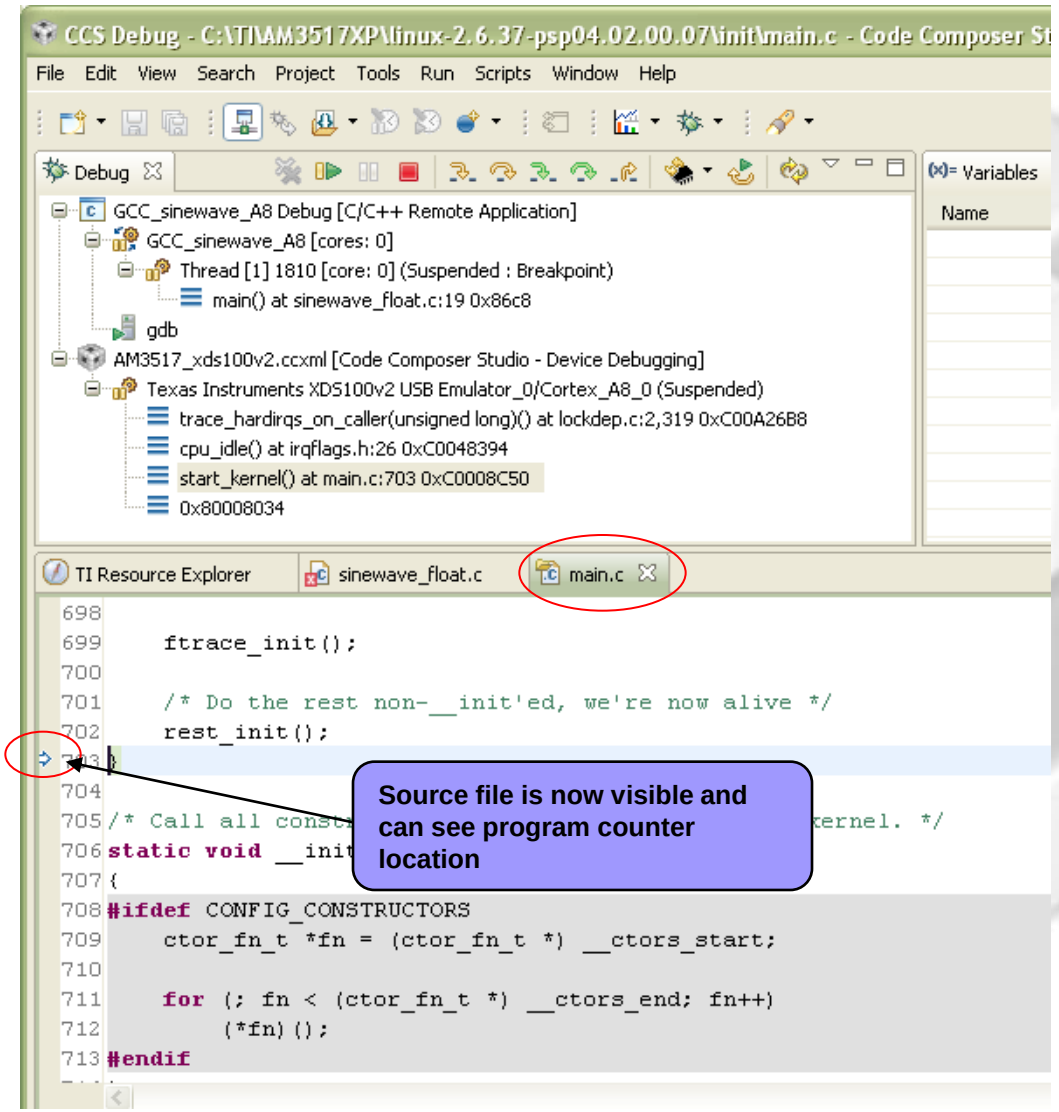


Stop Mode Debug: Source File Search Paths

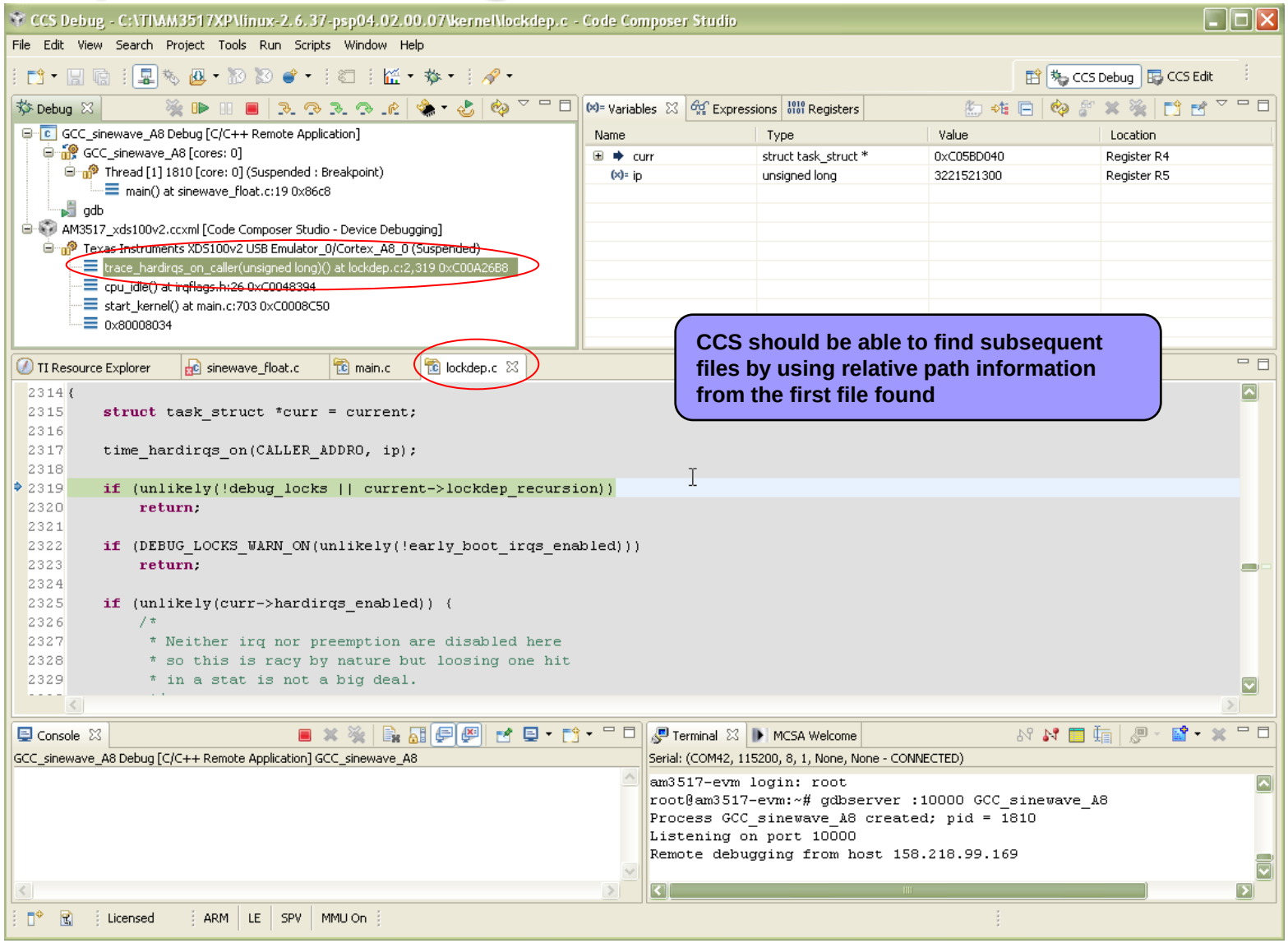


- **Root directory of the Linux kernel source files is located in:**
  - C:\TI\AM3517XP\linux-2.6.37-psp04.02.00.07
- **Look inside that directory for the file**
  - Can do a search inside the directory
  - You can often figure out the location by looking at where the debugger is looking
    - CCS is looking for “main.c” in the “/init” folder

Stop Mode Debug: Source File Search Paths

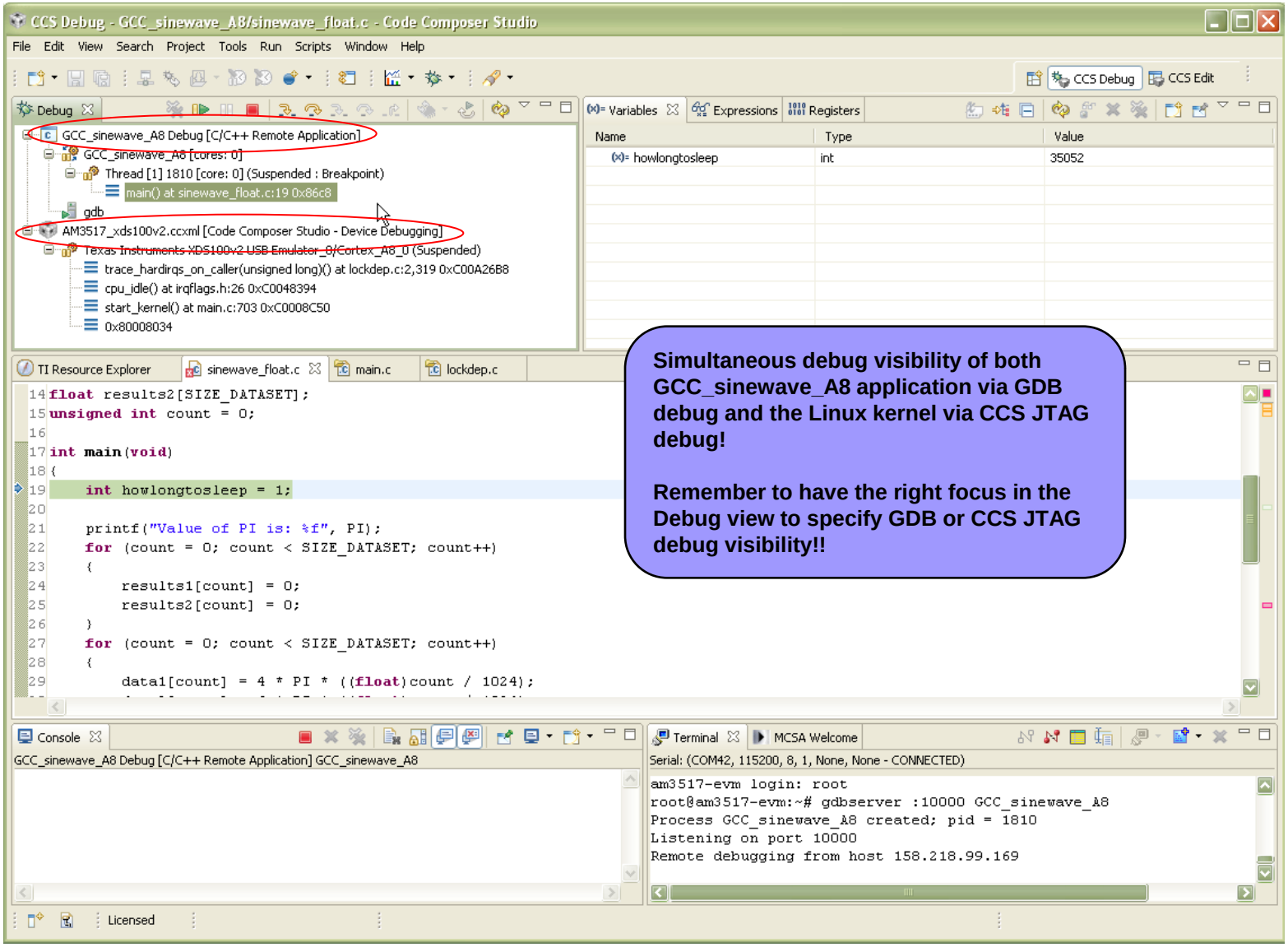


Stop Mode Debug: Source File Search Paths



CCS should be able to find subsequent files by using relative path information from the first file found

Mixed Mode Debug



Simultaneous debug visibility of both GCC_sinewave_A8 application via GDB debug and the Linux kernel via CCS JTAG debug!

Remember to have the right focus in the Debug view to specify GDB or CCS JTAG debug visibility!!

Summary

- **The JTAG debugger and CCSv5 are mandatory only for device driver and kernel development**
- **The application debugging is entirely done with open source components:**
  - Gdb client on the host side
  - Gdbserver on the target side
  - CCSv5 or standard Eclipse CDT
- **Step-by-step procedure available at:**
  - http://processors.wiki.ti.com/index.php/Linux_Debug_in_CCSv5