LAB conventions

Before starting, it is important to review some lab conventions that will ease your work…

- **Lab steps are in black and numbered for easier reference**
  1. …
  2. …

- **Explanations, notes, warnings are written in blue**
  - Warnings are shown with ⚠
  - Information is marked with ⓘ
  - Tips and answers are marked with 💡
  - Questions are marked with ?

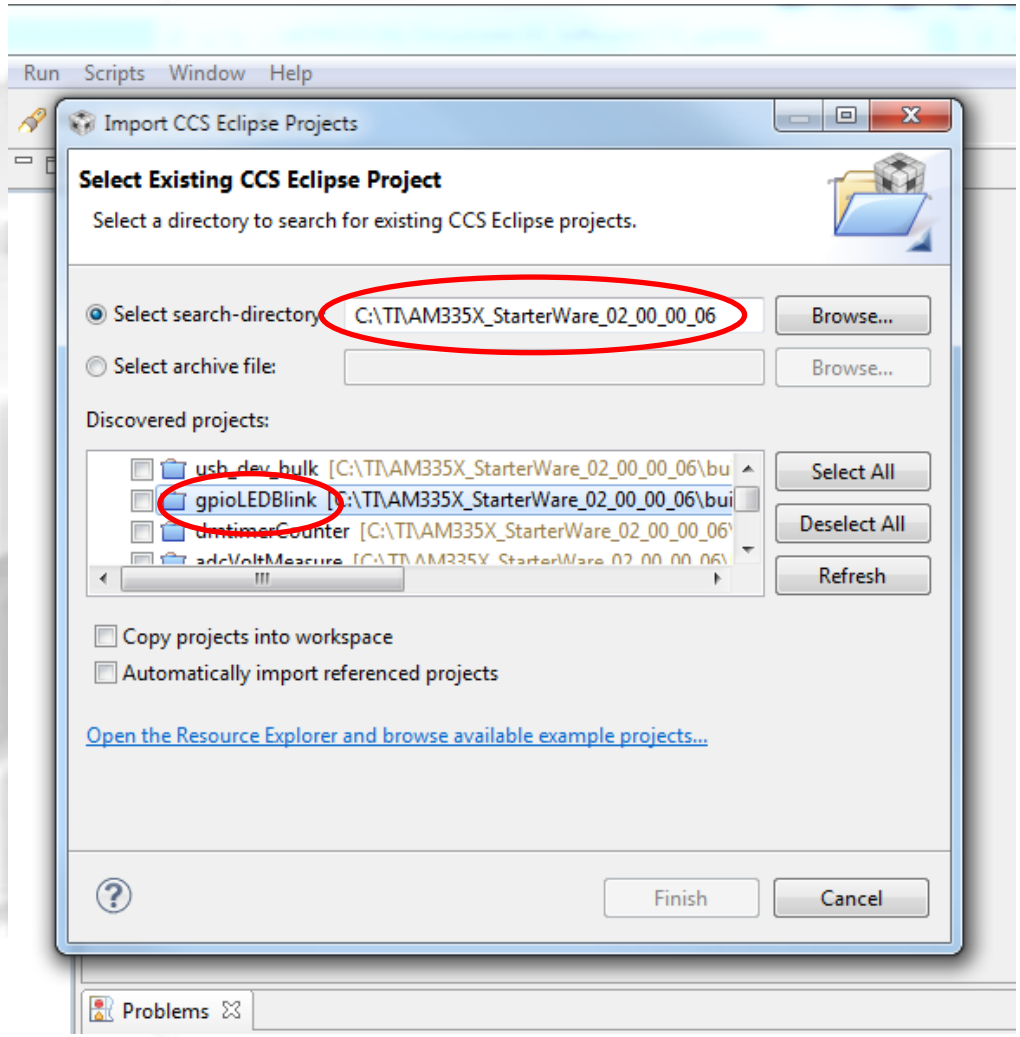
GPIO LED Blink Example: Exercise Summary

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

Import ‘gpioLEDBlink’ Project

1. Import the **gpioLEDBlink** project into the CCS Workspace by going to menu *Project → Import Existing CCS Eclipse Project*
2. In the box *Select search-directory*, type or browse to the Starterware installation directory:

   C:\TI\AM335X_StarterWare_02_00_00_06
3. Wait until the tool finishes discovering the available projects.
4. Select the project **gpioLEDBlink**
5. Click *Finish*



Build ‘gpioBlinkLED’ Project

1. In the *Project Explorer* view, select the **gpioLEDBlink** project (it should appear **[Active – Debug]**) to make it active

2. Click on the Build icon in the toolbar. The project will start the building process.

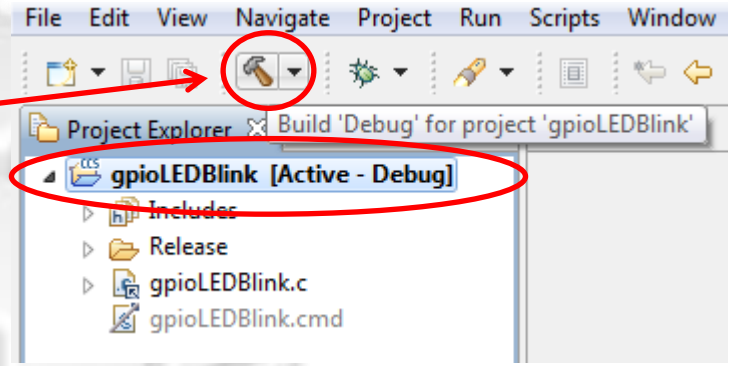
   When using a newly installed CCS, the tool will take extra time to build the Runtime Support Library (RTS) at this time. This is normal and will only happen once.

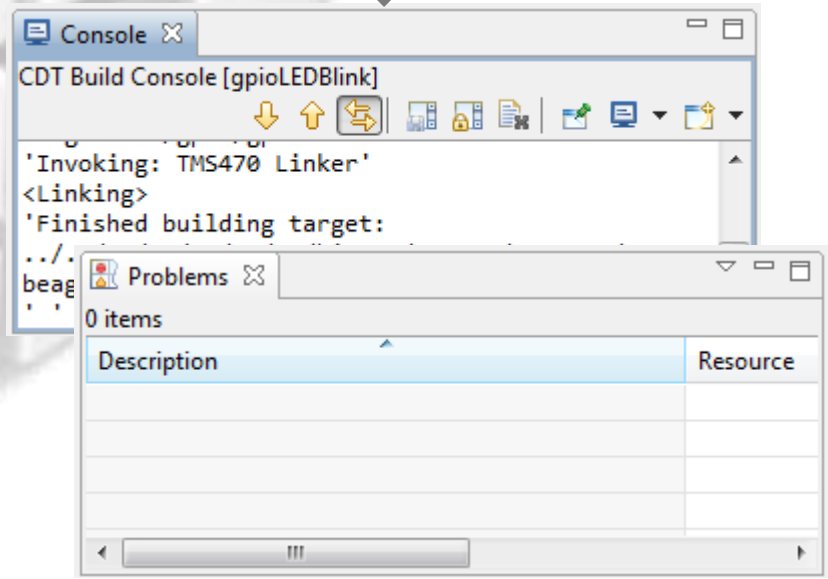
3. The *Console* view will appear at the bottom with build messages (information, warnings, errors) as the project builds

4. The *Problems* view will also appear at the bottom to highlight any possible build errors.

   When building the RTS, some warning messages will appear in the problems view and can be ignored.

5. If the build is successful, the *Problems* view will contain no errors (warnings can still be seen)





Create a Target Configuration File

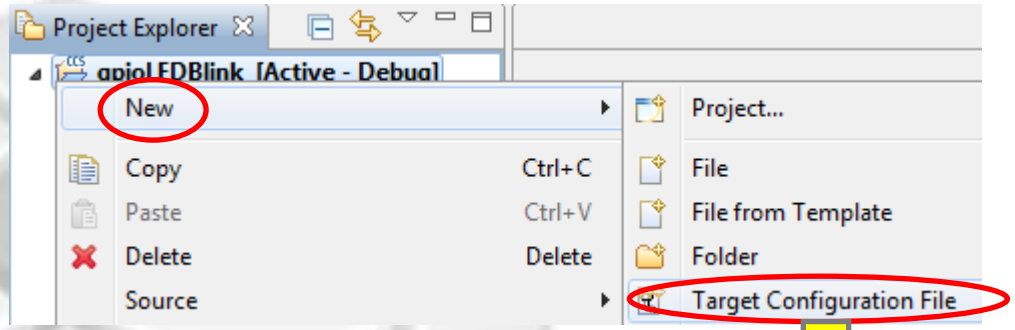
In order to connect to the board, a Target Configuration File must be created. This file contains the information about the JTAG emulator and the device to be connected.

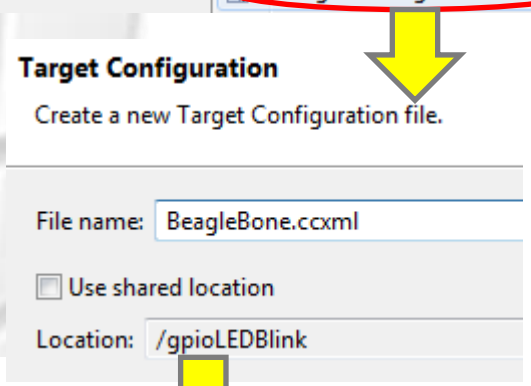
1. In the *Project Explorer*, right-click on the project and select *New* → *Target Configuration File*. A dialog box will be shown - give *BeagleBone* as a name and click *Finish*

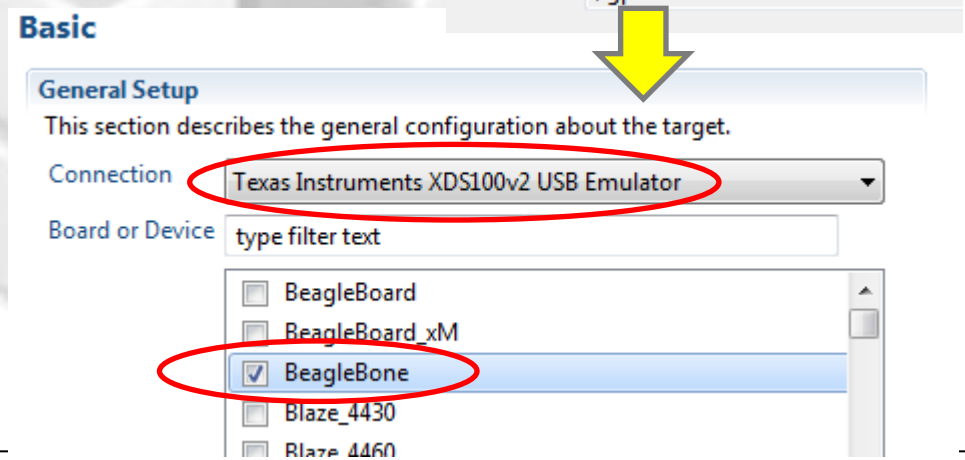
The target configuration editor will be shown

2. For the *Connection* choose *Texas Instruments XDS100v2 USB Emulator*

3. For the *Board or Device* choose *BeagleBone*



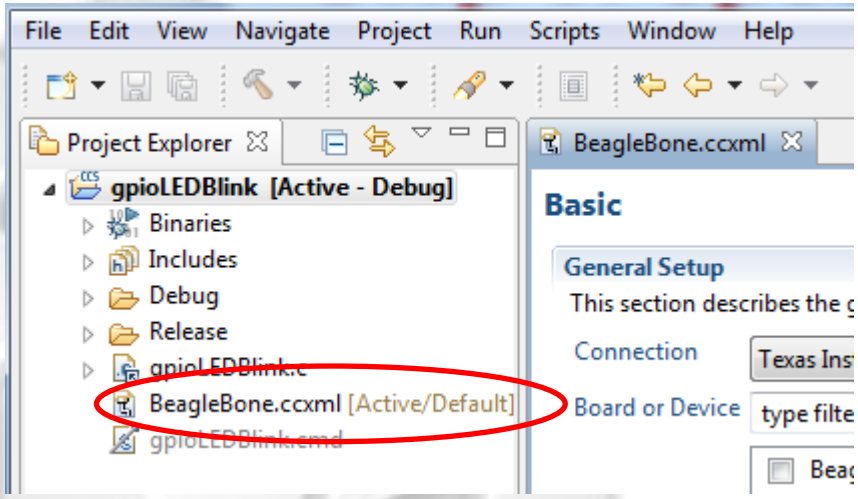




4. Click *Save*

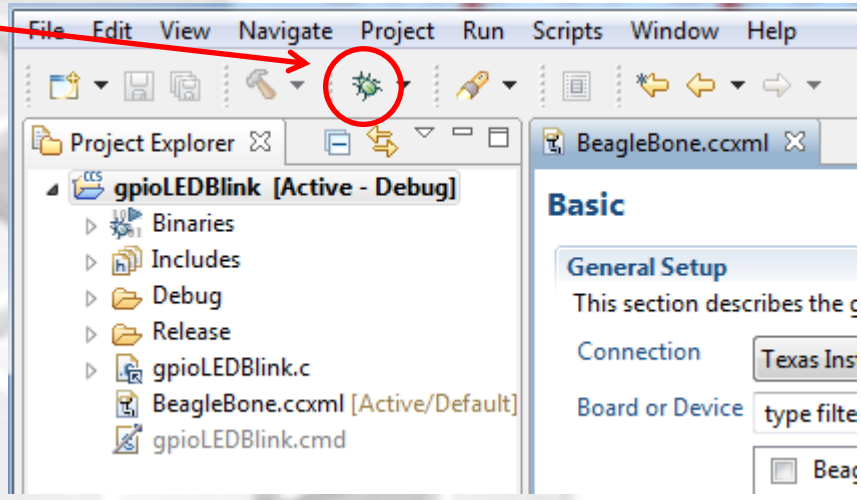
Debugger is ready for launch

- The Target Configuration File will be added to the project
- The [Active/Default] indicates this is the target configuration that will be used to debug this project (Active) and is also the Default for all other projects of the workspace (unless they have active configurations of their own).

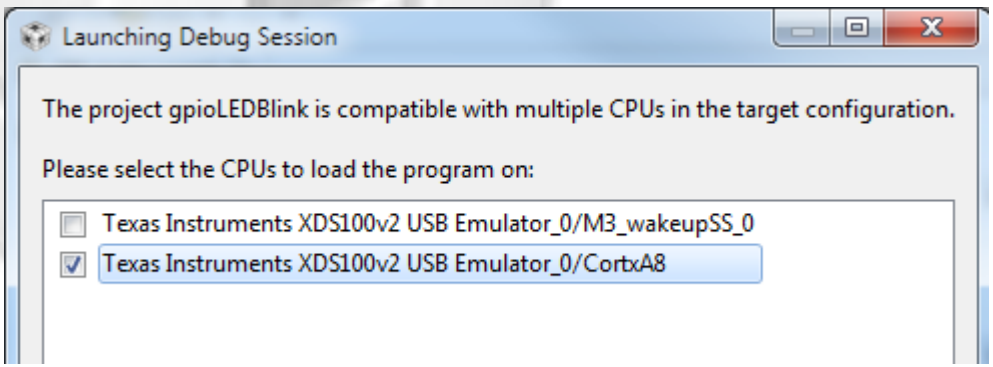


Debug ‘gpioBlinkLED’ Project

1. Click on the “green bug” button – make sure the project is selected!
2. When the Debug Session is launching, CCS asks which cores to load. De-select **M3_wakeupSS_0**



- This happens because BeagleBone has two cores compatible with the project (ARM cores)
- When you hit the green bug button, several actions are done automatically
  - Prompt to save source files
  - Build the project (incrementally)
  - Start the debugger (CCS will switch to the *CCS Debug* perspective)
  - Connect CCS to the target
  - Load the program on the target
  - Run to **main()**



Debug 'gpioBlinkLED' Project

CCS Debug - gpioLEDBlink/gpioLEDBlink.c - Code Composer Studio

File Edit View Project Tools Run Scripts Window Help

CCS Debug

Debug

gpioLEDBlink [Code Composer Studio - Device Debugging]
Texas Instruments XDS100v2 USB Emulator_0/M3_wakeupSS_0 (Disco
Texas Instruments XDS100v2 USB Emulator_0/CortxA8 (Suspended - S
main() at gpioLEDBlink.c:66 0x80000D90
start_boot() at startup.c:135 0x80000F54

Variables | Expressions | Registers

Name	Type	Value	Location

Switched to *CCS Debug* perspective

BeagleBone.ccxml | gpioLEDBlink.c

```
60 **                INTERNAL FUNCTION DEFINITIONS
61 ******************************************************************************/
62 /*
63 ** The main function. Application starts here.
64 */
65 int main()
66 {
67     /* Enabling fun
68     GPIO1ModuleClkC
69
70     /* Selecting GPIO1[23] pin for use. */
71     GPIO1Pin23PinMuxSetup();
72
73     /* Enabling the GPIO module. */
74     GPIOModuleEnable(GPIO_INSTANCE_ADDRESS);
75
76     /* Resetting the GPIO module. */
77     GPIOModuleReset(GPIO_INSTANCE_ADDRESS);
78
```

Program counter at main()

Console

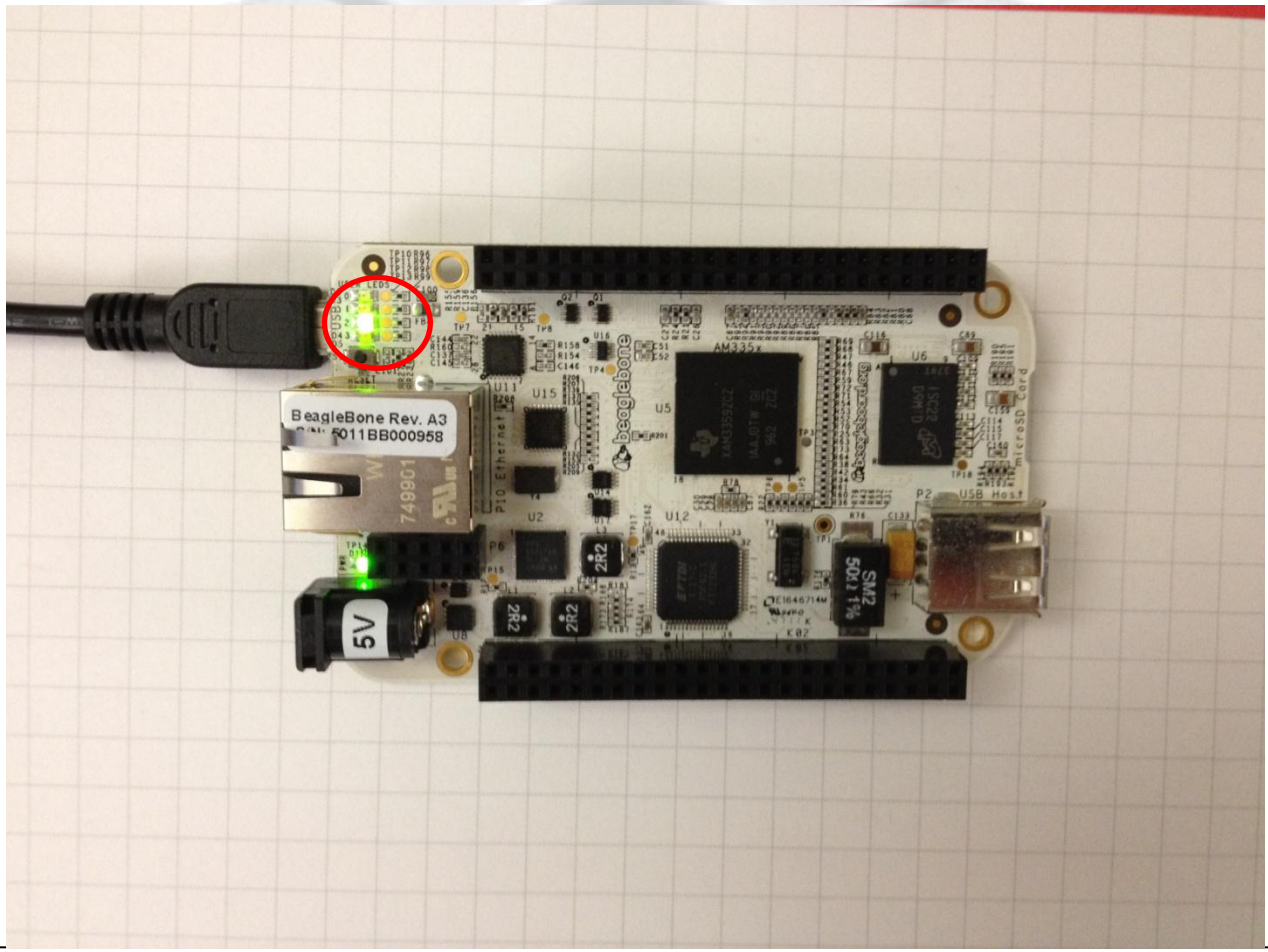
```
gpioLEDBlink
CortxA8: Output: DDR PHY DATA1 Register configuration is in progress .......
CortxA8: Output: EMIF Timing register configuration is in progress .......
CortxA8: Output: EMIF Timing register configuration is done .......
CortxA8: Output: DDR PHY Configuration done
CortxA8: Output: ****  AM335x DDR2 EMIF and PHY configuration is done ****
CortxA8: Output: ****  AM335x 15x15 EVM Initialization is Done *****************
```

Licensed | ARM | LE | SYS | MMU Off

Blink USR2 LED (D4)

1. **Press the *Run* button to run the program**

USR2 LED on the BeagleBone should now be blinking

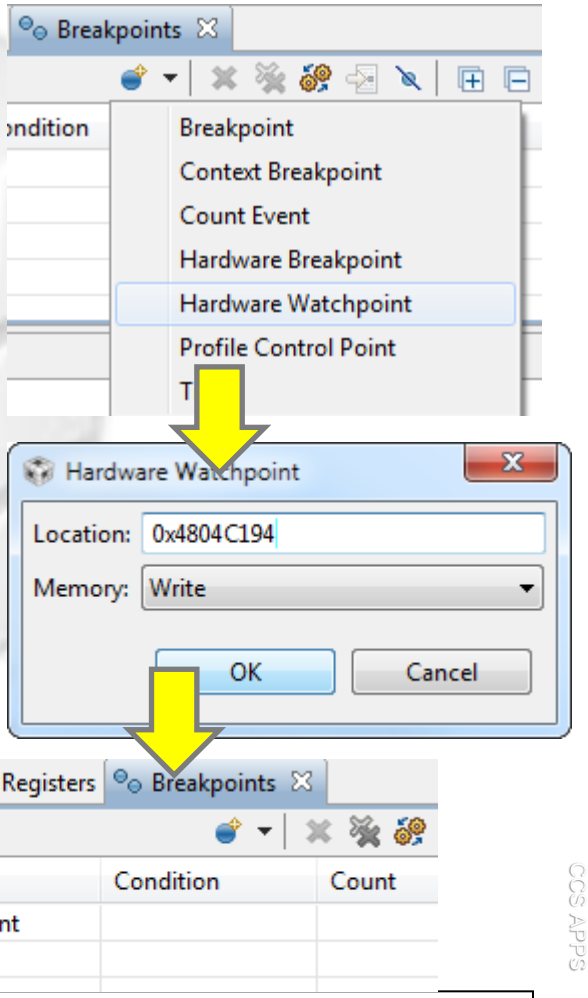


Debugging: Using Watchpoints

A Watchpoint is a type of breakpoint that monitors activity on a memory address

In this step we will set a watchpoint to halt the CPU anytime the LED will toggle

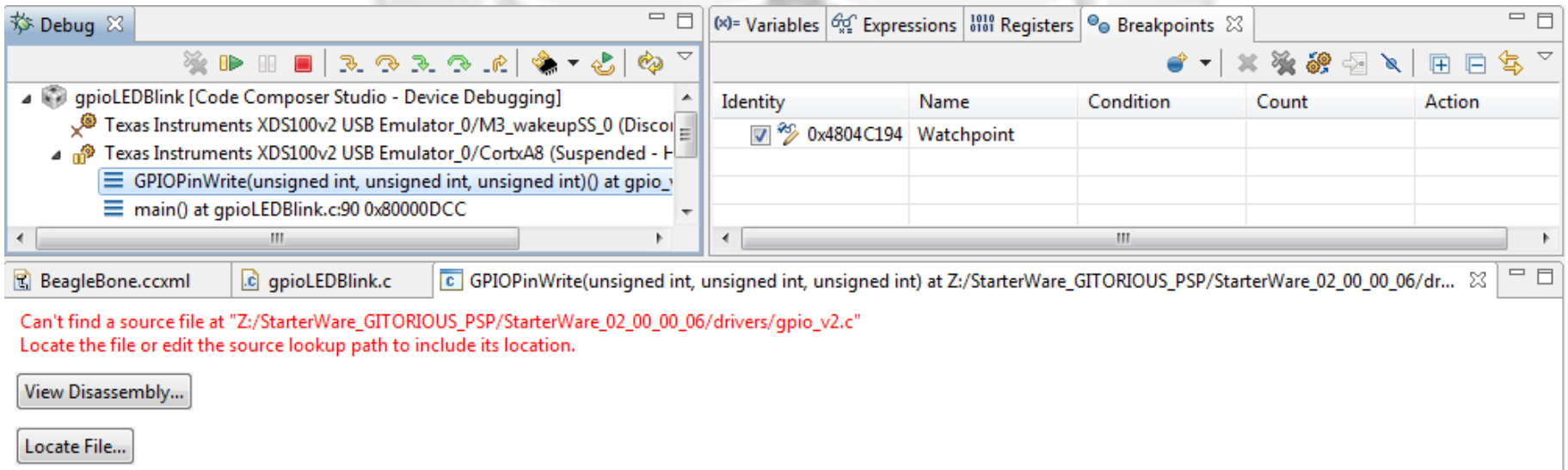
1. **Press the *halt/suspend* button to halt the running program**
   - The code should stop either in the **Delay()** function or somewhere inside the **while(1)** loop
2. **Open the *Breakpoints* view**
   - *View → Breakpoints*
3. **Create a new *Hardware Watchpoint***
   - *Location*: set it to 0x4804C194 (**GPIO_SETDATAOUT** register)
   - *Memory*: *Write*



Debugging: Using Watchpoints

1. Run the target again. Execution will automatically halt when USR2 LED is toggled (the GPIO_SETDATAOUTPUT register is written to)

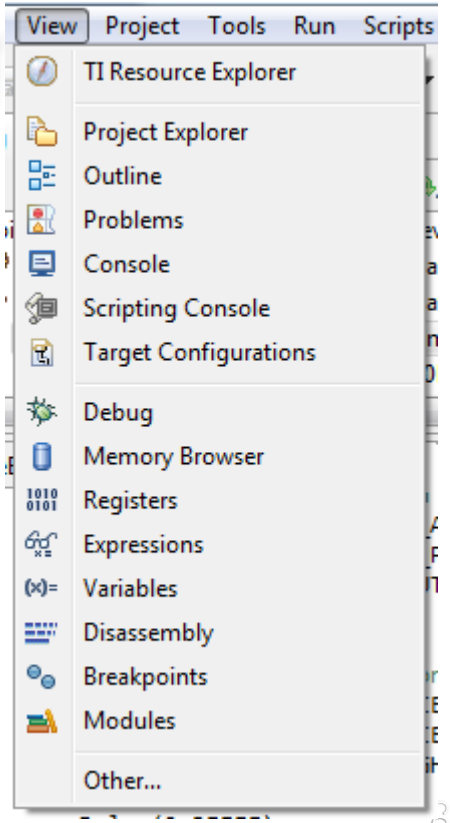
Notice that CCS will show a warning about a missing source file… If the project builds fine (with no errors), why does this happen?



CCS halted in the middle of code that is located inside a library, but it can't find its corresponding source file. You can click on the button *Locate File* and point to your local copy of the Starterware library source file.

More Debugging

- **Investigate other debugging views (Open via *View* menu)**
  - Memory Browser
  - Registers
  - Disassembly (see next slide)
- **Set breakpoints**
  - Double click on a source line to set/clear
  - See list of breakpoints with the *Breakpoints* view
- **Use the buttons in the *Debug* view to:**
  - Restart the program
  - Source stepping
  - Assembly stepping

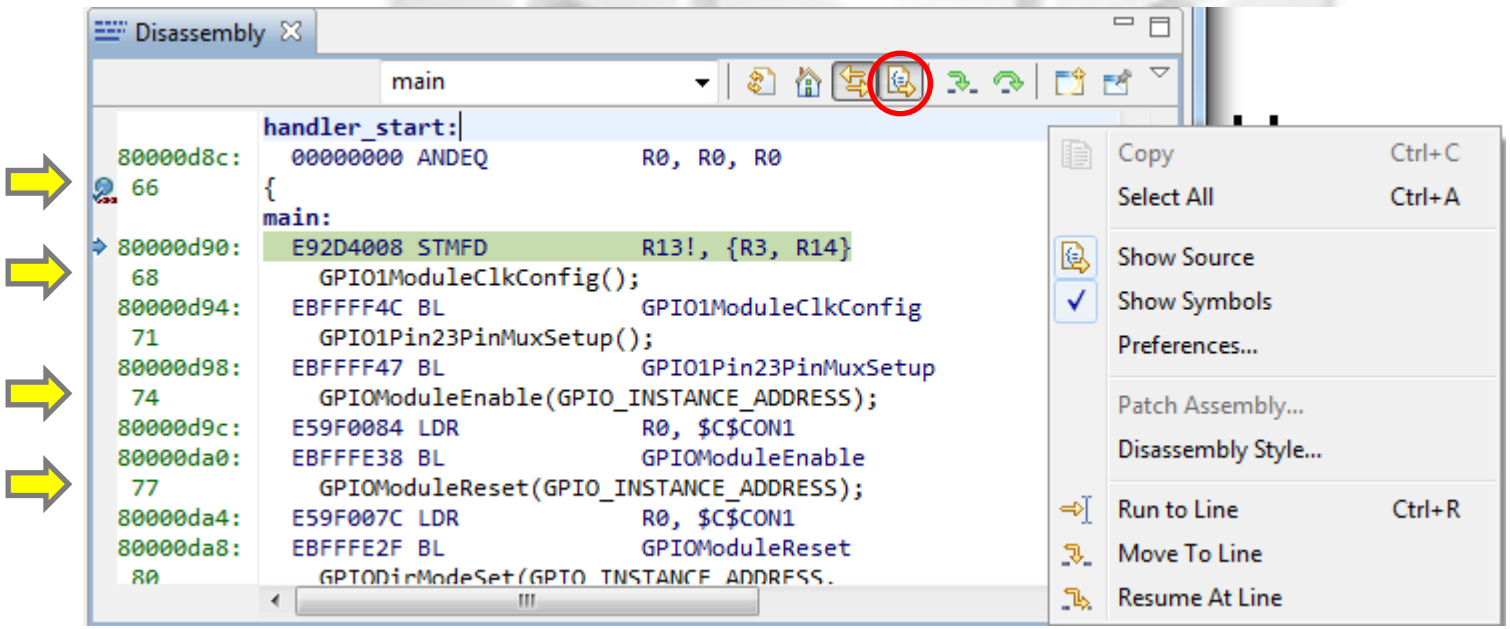


View: Disassembly

1. Open the Disassembly view. Go to the main() symbol in this view by typing “main” in the address field and hit <ENTER>

   Just as an example, the picture below shows the current location of the PC (small blue arrow) and any breakpoints (small blue circles)

2. Toggle the *Show Source* button. Note the toggling of interleaved source with the disassembly



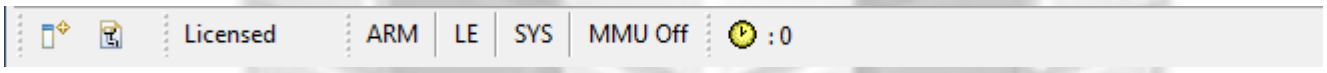
Debugging: Using Profile Clock

The profile clock

- Available on most devices and can be used to count cycles
- On some targets it can be used to count other events like cache hits/misses, bus stalls, etc.

1. Enable the Clock

- Menu *Run* → *Clock* → *Enable*
- The clock will now be displayed on the status bar



Check that the watchpoint that is watching for writes to GPIO_SETDATAOUT is enabled

2. Click the *Run* button

- Clock should now show ~267M cycles

Tip: Double-clicking on the clock icon will reset the count to '0'

Increase USR2 LED Blink Rate

The blink rate of USR2 LED can be increased by changing the value of the delay

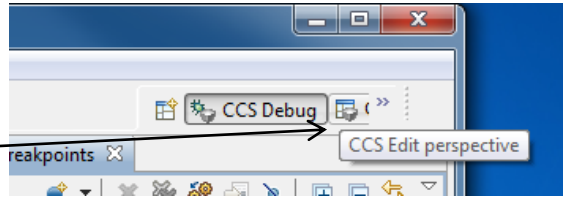
1. **Modify lines 90 and 97 of gpioLEDBlink.c**
   - From: `Delay(0x3FFFF);`
   - To: `Delay(0x1FFFF);`

BeagleBone.ccxml | *gpioLEDBlink.c | GPIOPinWrite(unsigned int, unsigned i...

```
82                      GPIO_DIR_OUTPUT);
83    while(1)
84    {
85        /* Driving a logic HIGH on the GPIO pin. */
86        GPIOPinWrite(GPIO_INSTANCE_ADDRESS,
87                     GPIO_INSTANCE_PIN_NUMBER,
88                     GPIO_PIN_HIGH);
89
90        Delay(0x1FFFF);
91
92        /* Driving a logic LOW on the GPIO pin. */
93        GPIOPinWrite(GPIO_INSTANCE_ADDRESS,
94                     GPIO_INSTANCE_PIN_NUMBER,
95                     GPIO_PIN_LOW);
96
97        Delay(0x1FFFF);
98    }
99
100 }
```

2. **Remember to disable the watchpoint set before! Go to the Breakpoints view and uncheck its checkbox**

View: Local History

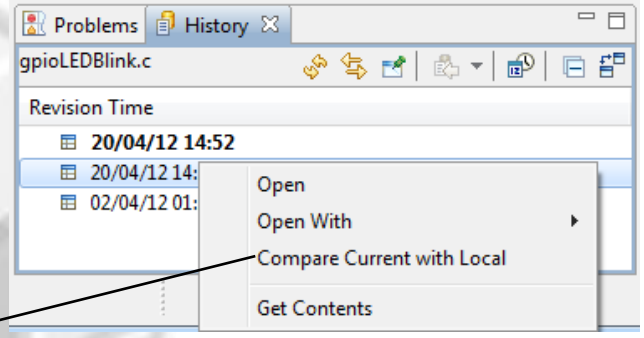


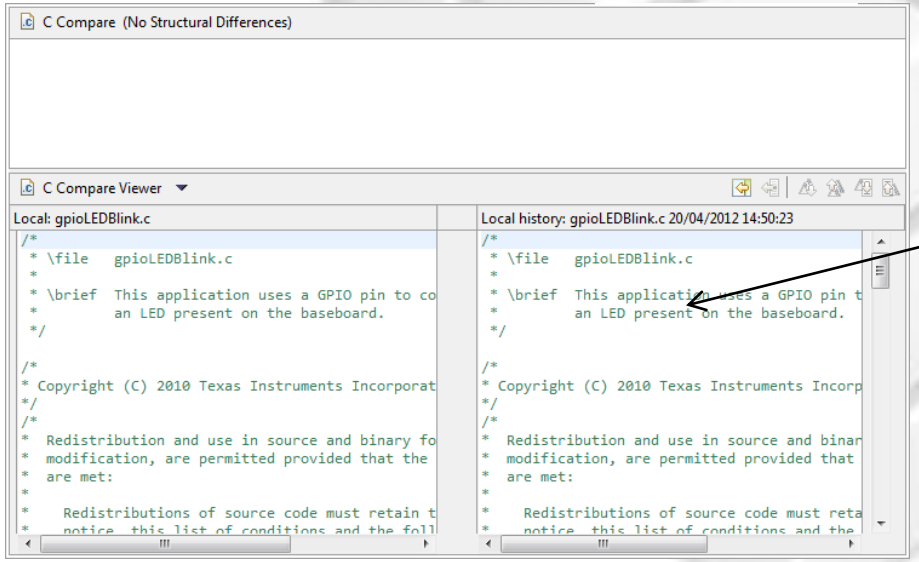
CCS keeps a local history of source changes

1. Switch to the *CCS Edit* perspective
2. Right-click on a file in the editor and select *Team* → *Show Local History*

You can compare your current source file against any previous version or replace it with any previous version

1. Double-click on a revision to open it in the editor
2. Right-click on a revision to compare that revision to the current version





CCS also keeps project history

- If files are deleted from the project, CCS can recover them

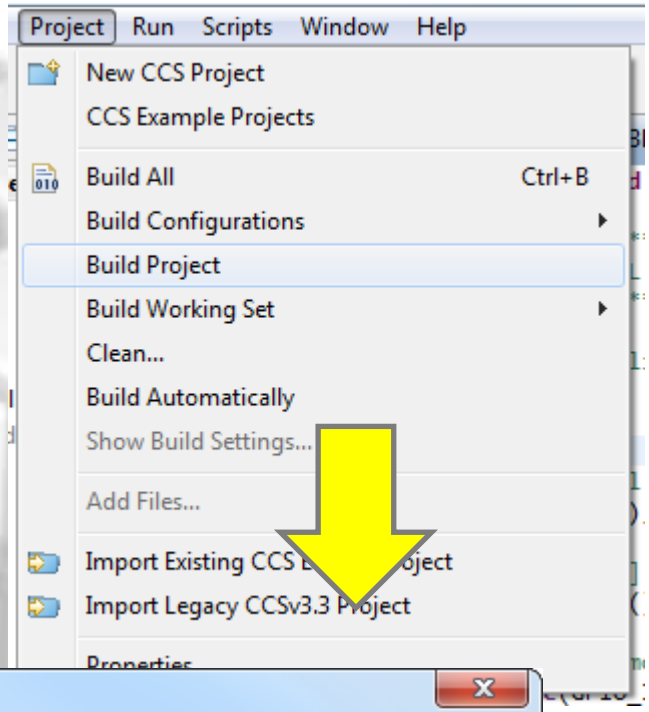
1. Right-click on the project and select *Restore from Local History* in the context menu

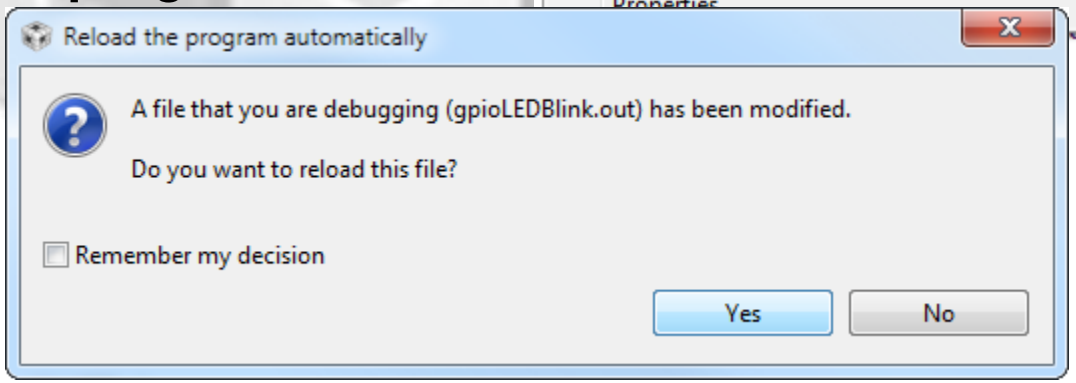
Rebuild Project

1. **Switch to the *CCS Edit* perspective**
2. **Rebuild the project**
   - Menu *Project* → *Build Project*

CCS will automatically detect that the currently program being debugged has changed/rebuilt and ask if it should reload the file

3. **Select *Yes* and CCS will reload/reflash the program and run to main()**

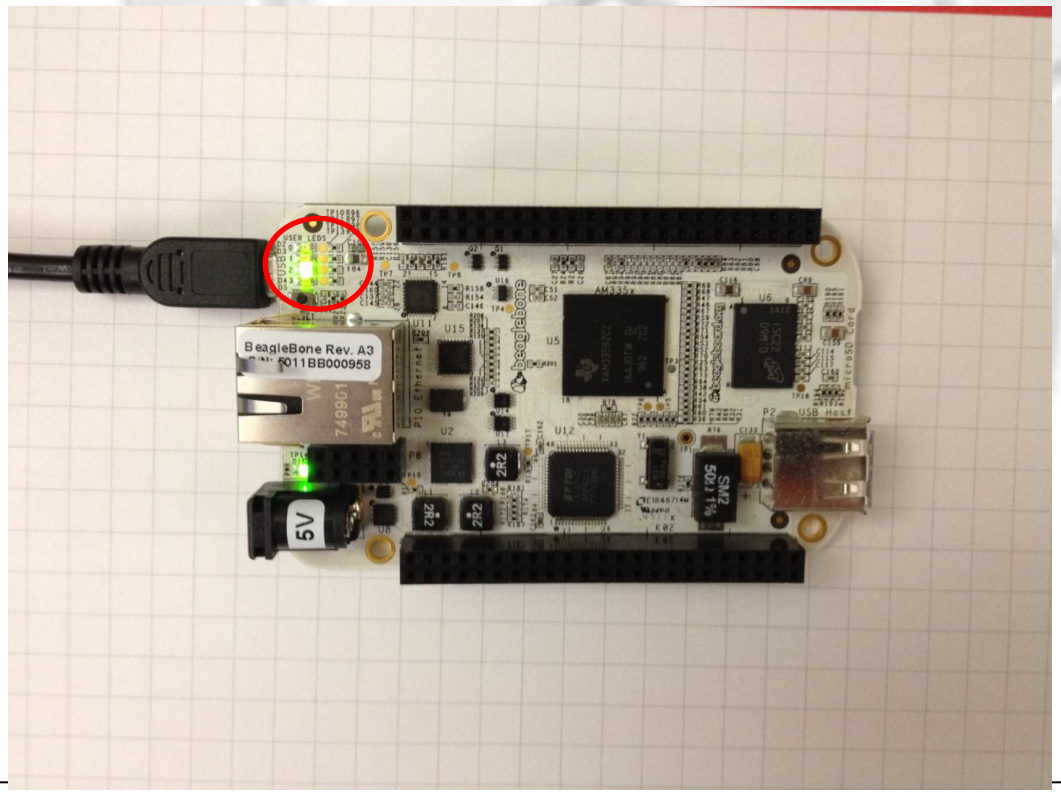




Blink USR2 LED

1. **Switch to the CCS Debug perspective**
2. **Press the *Run* button to run the program**

The USR2 LED on the BeagleBone should now be blinking at a much quicker rate
If not, remember to disable (not delete) the watchpoint set in the previous session! ☺



Debugging: Profile Clock – Part 2

1. **Pause execution of the program**
2. **Double-click on the clock icon in the status bar to reset the value to ‘0’**
3. **Enable the watchpoint for GPIO_SETDATAOUT**
4. **Click the *Run* button**
   - Clock should now show ~133M cycles (1/2 of 267M)

Terminate the Debug Session

1. In the Breakpoint view, <u>delete</u> (not disable) the watchpoint, as it will not be used in the next lab – simply highlight it and press the <Delete> key.
2. Go to the Debug View
3. Click on the terminate button
4. This will kill the debugger and return you to the Edit perspective

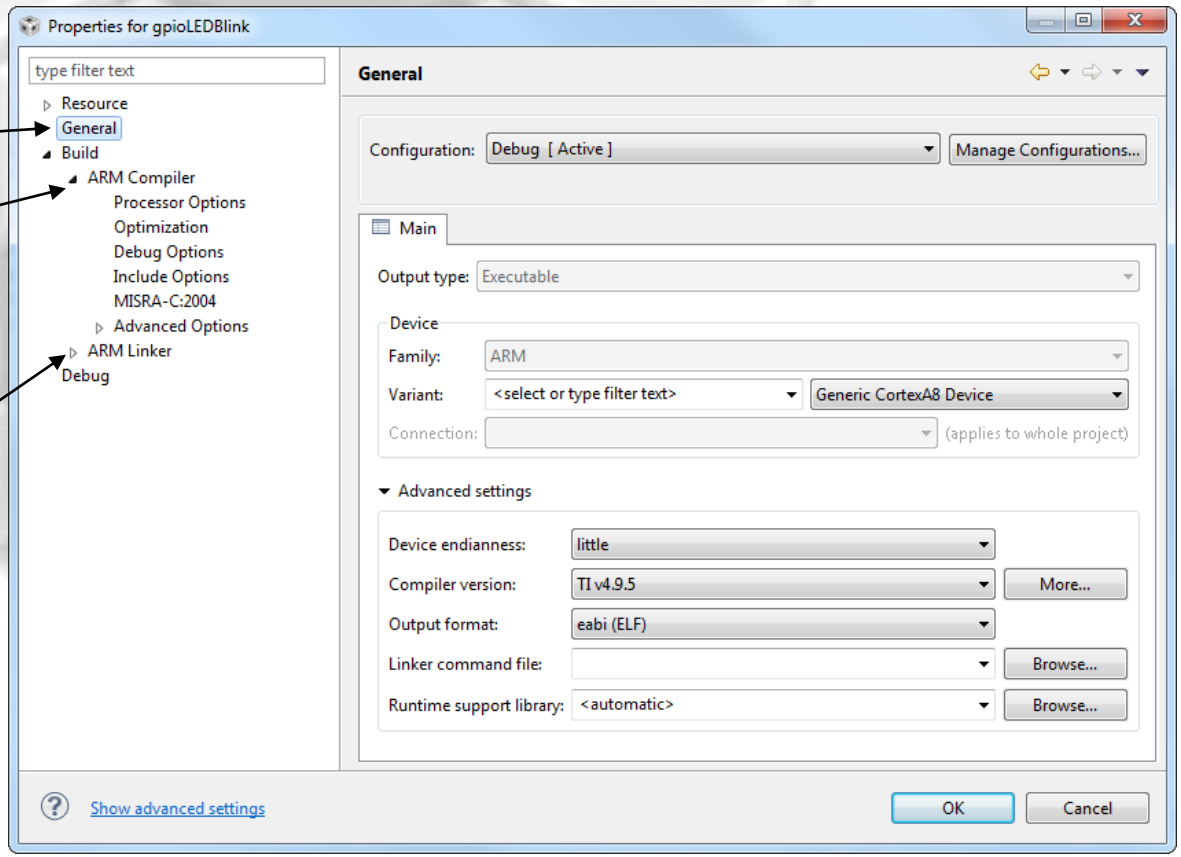
Changing Project Properties

1. Make sure you are in the *CCS Edit* perspective
2. Right click on the gpioLEDBlink project and select *Properties*

Device and high level settings

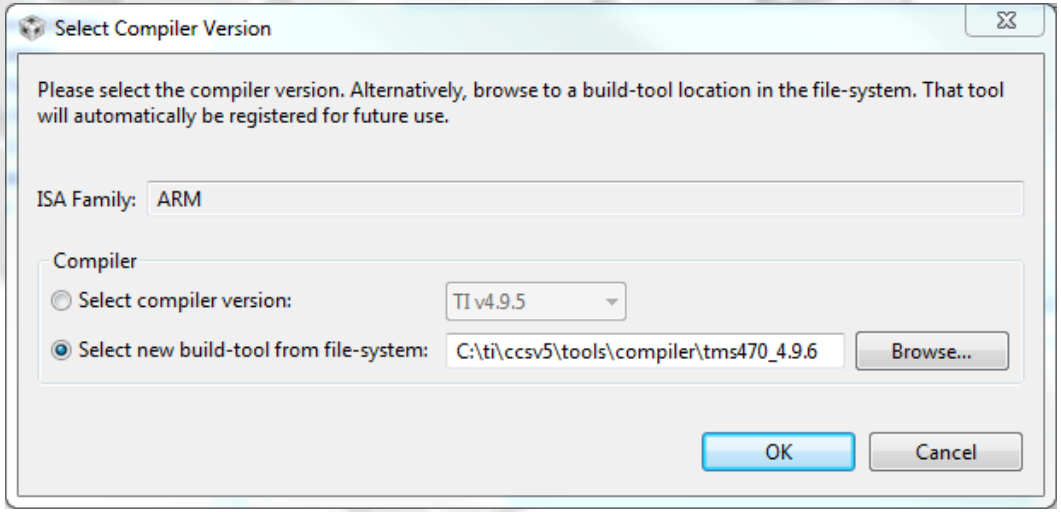
Compiler Options

Linker Options



Changing the Compiler Version

1. **Click on** *General*
2. **Click on the** *More…* **button beside the Compiler version** *TI v4.9.5*
3. **Check the option** *Select new build-tool from file-system*
4. **Browse to the location of the new compiler tools and click OK**
   C:\ti\ccsv5\tools\compiler\tms470_4.9.6

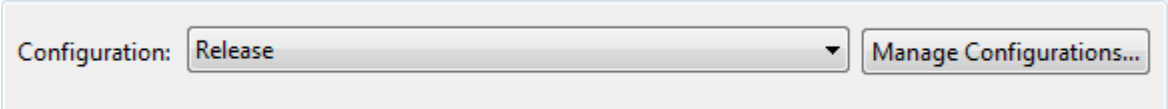


CCS will determine what compiler is located there and select it for your active configuration. You will see that TI v4.9.6 is now specified

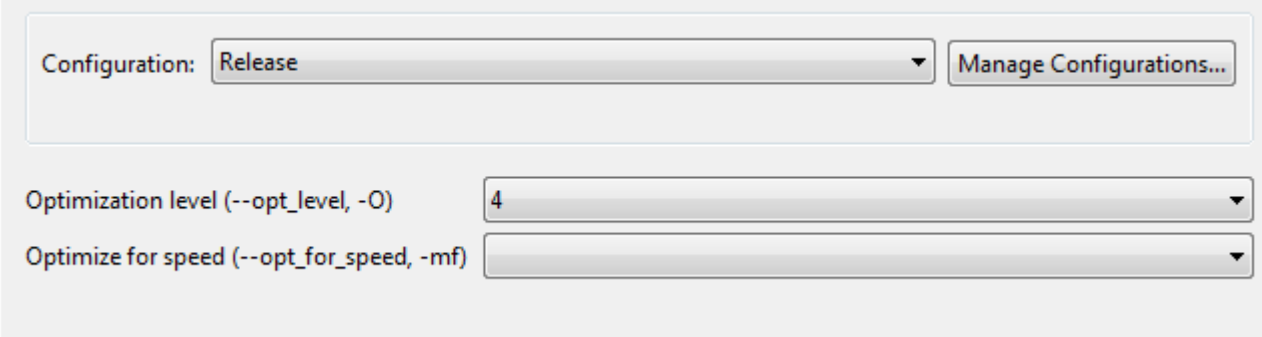
Changing Build Options

Build options are set per build configuration

1. **Change your Configuration to *Release***



2. **Change the optimization settings**
   - Go to the *Build* → *TMS470 Compiler* → *Optimization*
   - Change the optimization level to 4



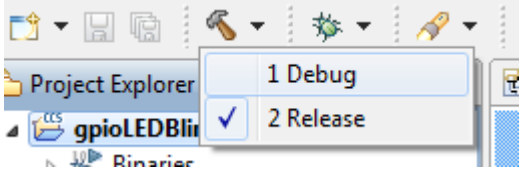
3. **Click OK**

Changing Build Options

1. **Change the active configuration to *Release***
   - Right click on the Project
   - Select *Build Configurations* → *Set Active* → *Release*
2. **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 clicking on the arrow beside the build button and selecting the configuration you want to build

- Select *Release* and it will build this configuration
- The active configuration is indicated by the Checkmark



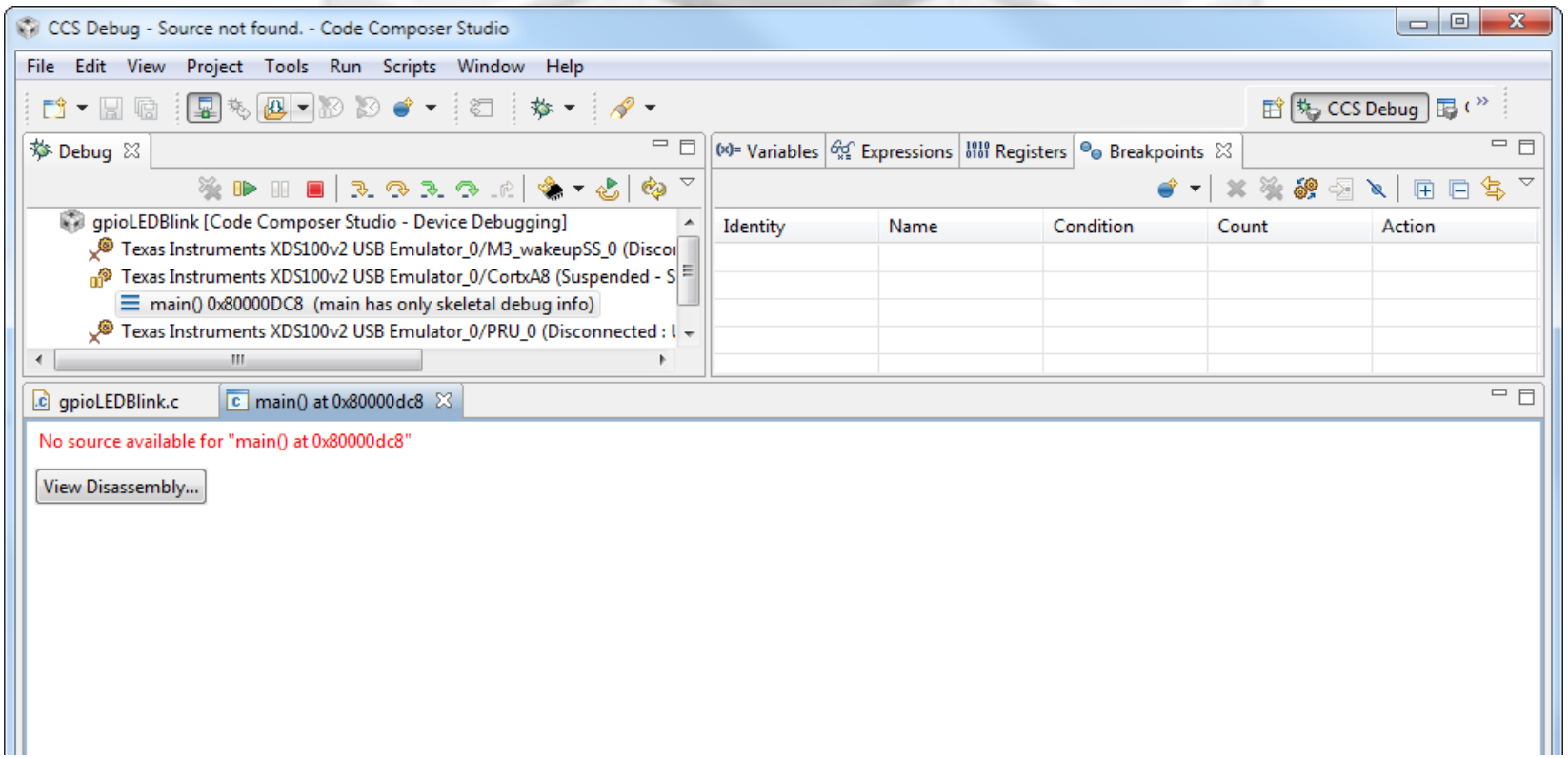
GPIO LED Blink Example: Exercise Summary

- **Congratulations! You finished all the lab steps for the first half.**
- **At this point you experimented the following concepts:**
  - Workspaces
  - Welcome screen / Resource Explorer
  - Project concepts
  - Basics of working with views
  - Debug launch
  - Debug control
  - Profile Clock
  - Local History
  - Build Properties
  - Changing compiler versions
- **If you still have some time before the next half, please move to the optional steps of the lab:**
  - Debug symbols
  - Profile

Optional: Load the optimized code

1. After building the code, hit the “green bug” button

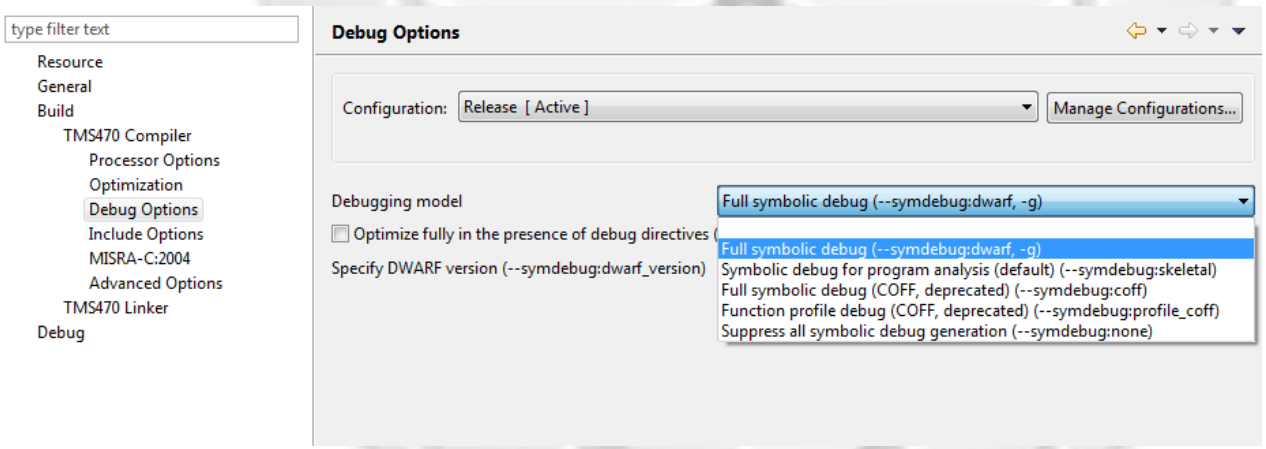
Do you see any source code? Why not? (answer in the next slide)



Optional: Add the debug symbols

A: The code loaded before does not have debug symbols...

1. **Switch back to *CCS Edit* Perspective**
   - Right click on the Project; select *Properties* → *Compiler* → *Debug Options*
   - Set *Debugging Model* to *Full Symbolic Debug*



2. **Build the project by clicking the build button**
   - CCS will ask if you want to reload the code. Click *Yes*.

3. **Switch to *CCS Debug* Perspective**

You should see the source code now. The option above allows the debugger to properly correlate with the source code.

Optional: Profile the optimized code

1. **Add the watchpoint for 0x4804C194 (GPIO_SETDATAOUT register)**
   - Refer to slide 10
2. **Enable the Profile clock**
   - Refer to slide 14
3. **Run the code**
   - In the first time you should get around 70k cycles.
   - In the second time you should get around 130M cycles.

Why did the optimization not change the cycle count too much?

- The code spends most of its time in the Delay() function (slide 15), which is mostly a do-nothing loop and thus not very optimizable.
- Therefore the optimizer can only optimize some conditional and control calls across the entire code.

GPIO LED Blink Example: Optional Exercise Summary

- **Congratulations! You finished all the optional lab steps.**
- **At this point you experimented the following concepts:**
  - Debug symbols
  - Profile

⚠ **Stop at this time and wait for the next half of the presentation.**

BLANK

UART Echo Example: Exercise Summary

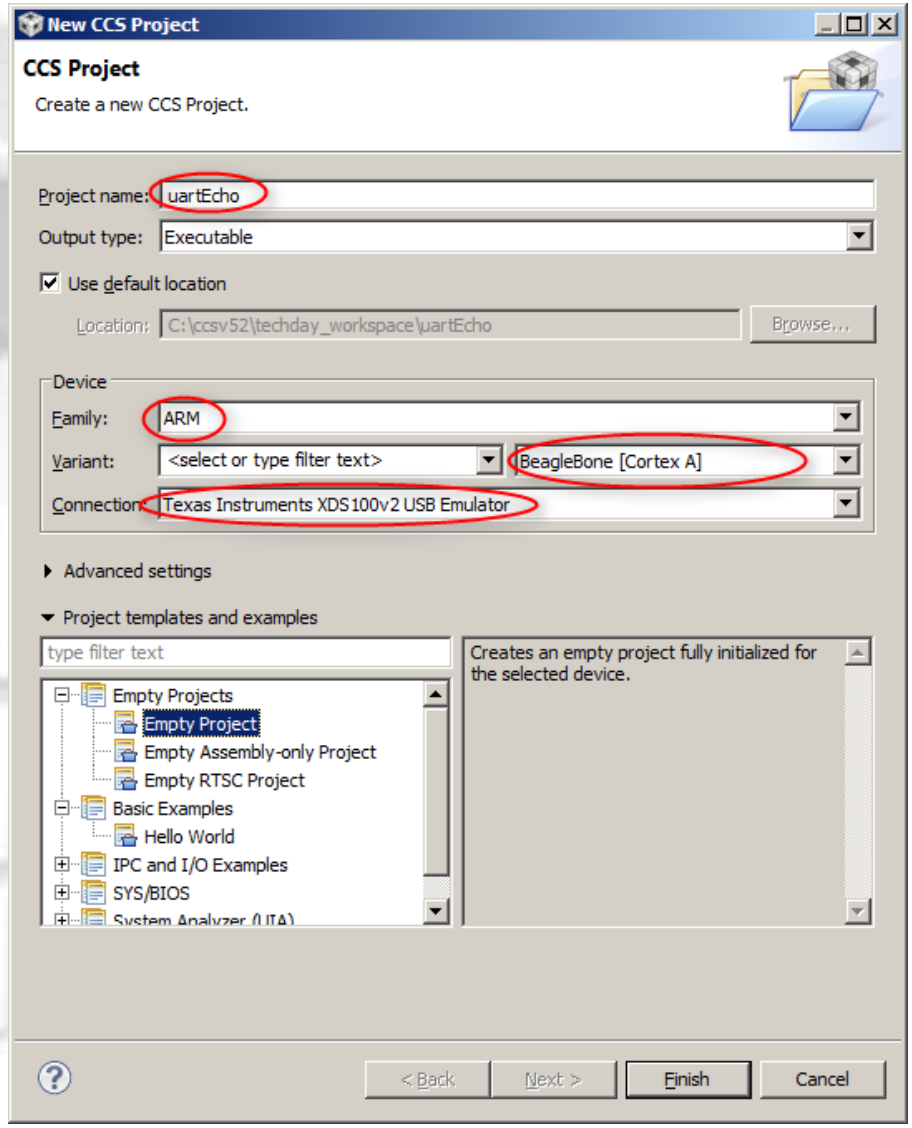
- **Key Objectives**
  - Create a new portable project based on the UART Echo example
  - Create workspace level variables for the project
  - Link files to the project using variables
  - Configure build properties using variables
  - Validate project by building, loading and running the program
- **Tools and Concepts Covered**
  - Portable Projects
  - Linked resources
  - Linked resource path variables
  - Build variables

Create a New Project

1. **Launch the *New CCS Project* Wizard**
   - Go to menu File → *New* → *CCS Project*
2. **Fill in the fields as shown in the right**
   - *Project name:* uartEcho
   - *Device* → *Family:* ARM
   - *Variant:* BeagleBone [Cortex A]
   - *Connection:* Texas Instruments XDS100v2 USB Emulator
3. **Click *Finish* when done**

Generated project will appear in the Project Explorer view

4. **Remove the generated files <main.c> and <AM3358.cmd> from the project**

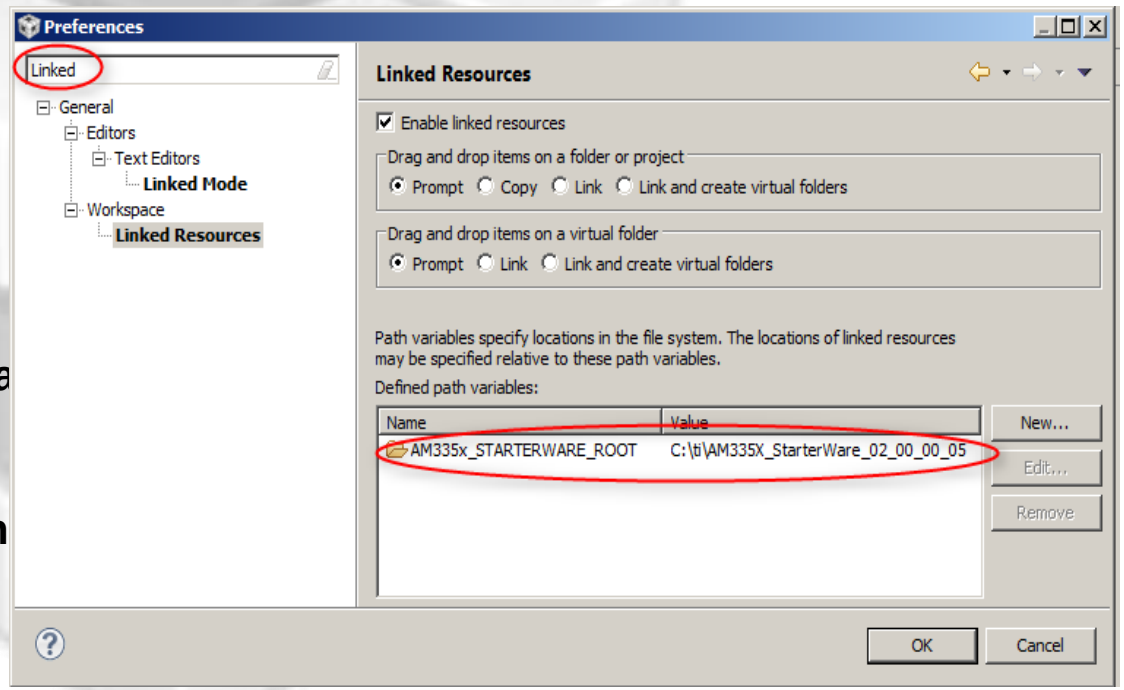


Create a Linked Resource Path Variable

Here we will create the Linked Resource Path Variable which will be used when linking source files (resources) to the project

1. **Open the workspace preferences**
   - Menu *Window* → *Preferences*
2. **Go to the *Linked Resources* preferences**
   - Type 'Linked' in the filter field to make it easier to find
3. **Use the *New* button to create a 'Linked Resource Variable' (AM335x_STARTERWARE_ROOT) that points to the root location of the AM335x StarterWare directory**

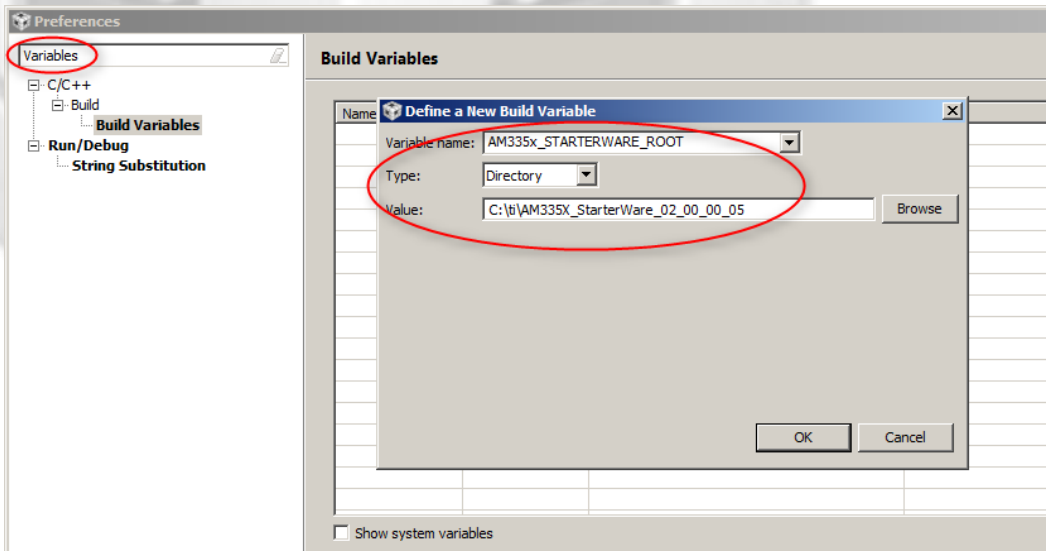
C:\ti\AM335X_StarterWare_02_00_00_06



Create a Build Variable

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

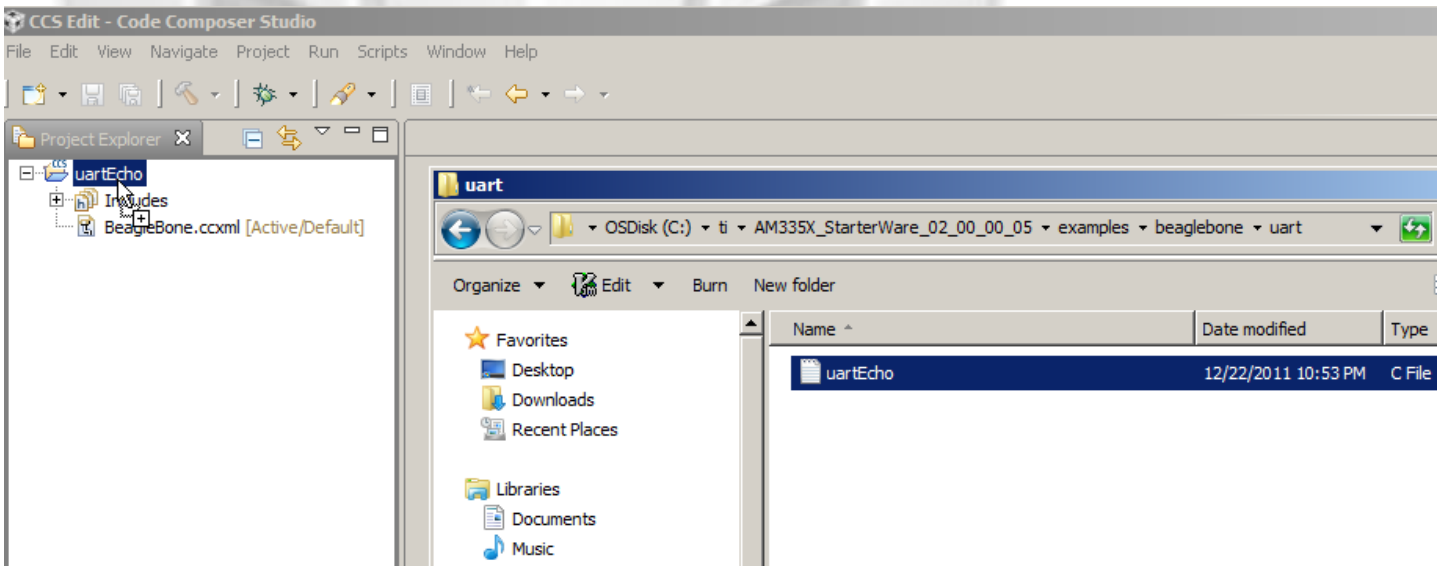
1. **Go to the *Build Variables* preferences**
   - Type 'Variables' in the filter field to make it easier to find
2. **Build Variables allow you to use variables in the project properties**
   - Linked Resource variables are only used for linked files
3. **Use the *Add* button to create a 'Build Variable' (AM335x_STARTERWARE_ROOT) that points to the root location of the AM335x StarterWare directory**
4. **Hit *OK* when done**



Link Source Files to Project

Here we will link the source file relative to the Linked Resource Path Variable previously created

1. **Open Windows Explorer and browse to:**
   C:\ti\AM335X_StarterWare_02_00_00_06\examples\beaglebone\uart
2. **Drag and drop the following file into the new project in the CCS Project Explorer view**
   - <uartEcho.c>

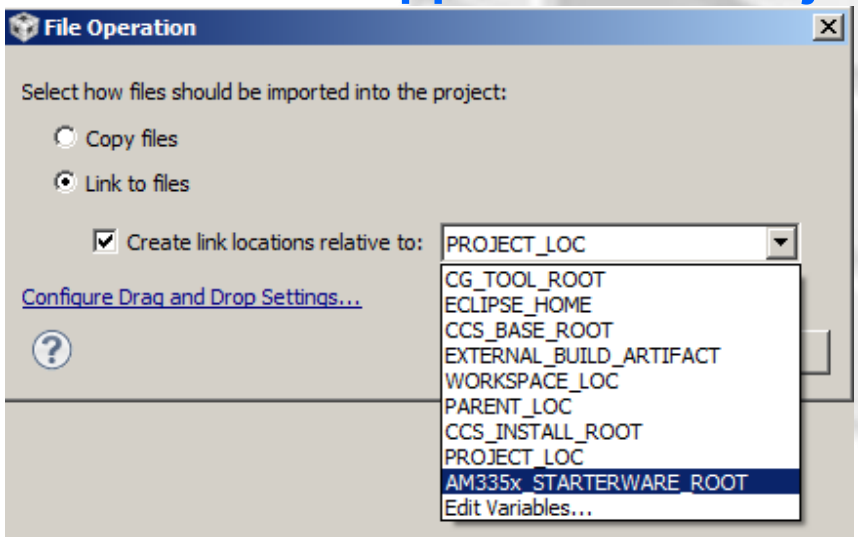


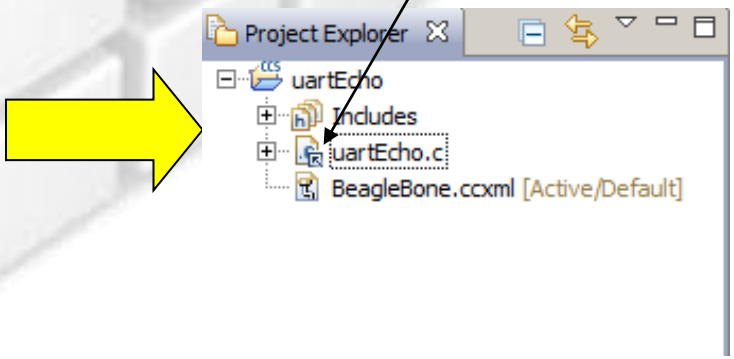
Link Source Files to Project

A dialog will appear asking if you wish to Copy or Link the files:

1. **Select *Link to files***
2. **Select *Create link locations relative to:***
   - Use the new Linked Resource variable we created (**AM335x_STARTERWARE_ROOT**)
3. **Hit *OK***

Files will now appear in the Project Explorer with the 'link' icon

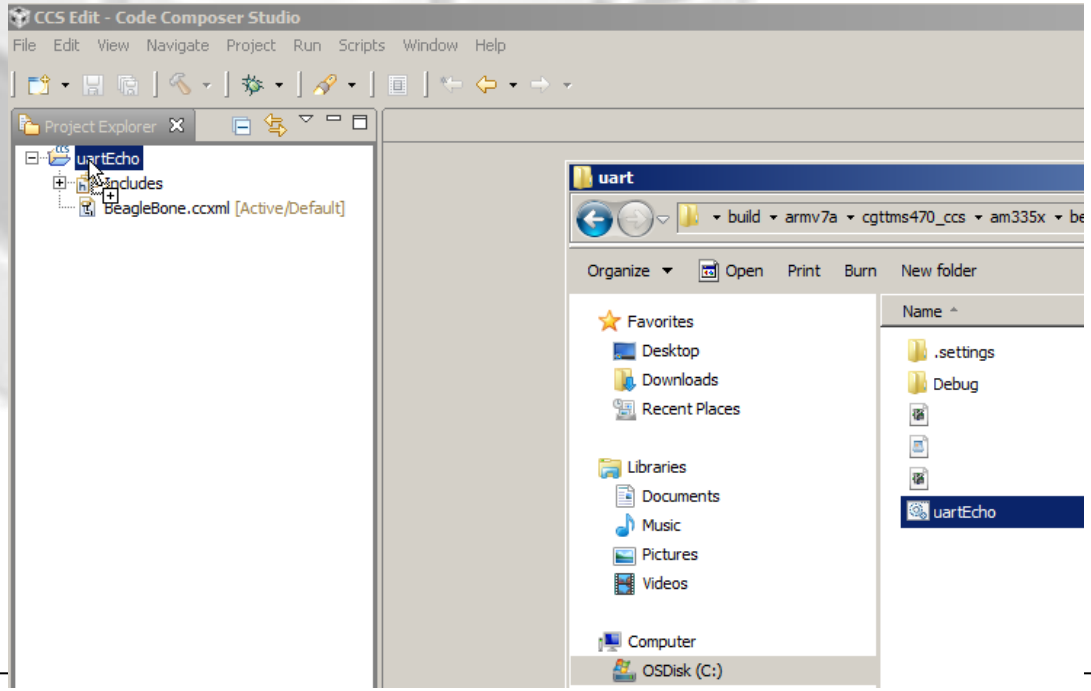




Add Files to Project

Here we will add the linker command file to the project. Ideally this file should also be "linked" but since we will be editing this file for the lab, we will "copy" the file instead

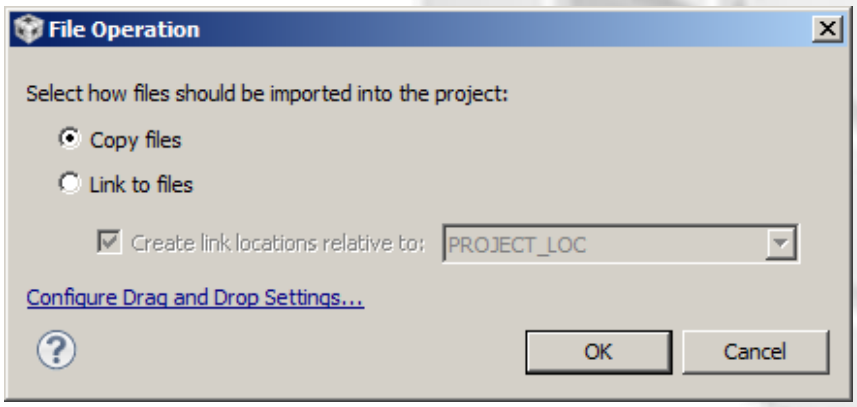
1. **Open Windows Explorer and browse to:**
   C:\ti\AM335X_StarterWare_02_00_00_06\build\armv7a\cgt_ccs\am335x\beaglebone\uart
2. **Drag and drop the following file into the new project in the CCS Project Explorer view**
   – <uartEcho.cmd>

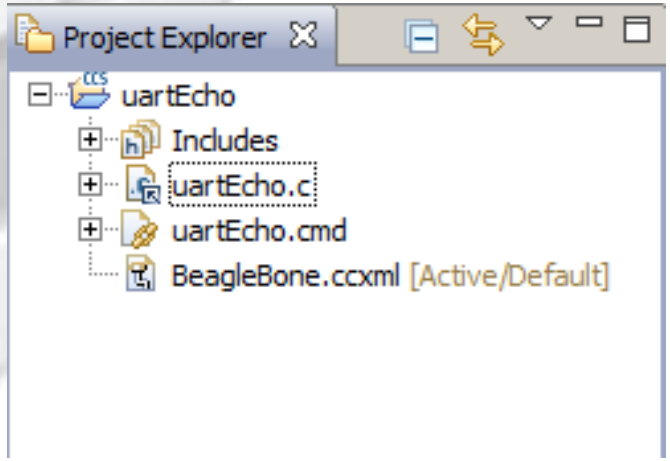


Add Files to Project

A dialog will appear asking if you wish to Copy or Link the files:

1. Select *Copy files*
2. Hit *OK*

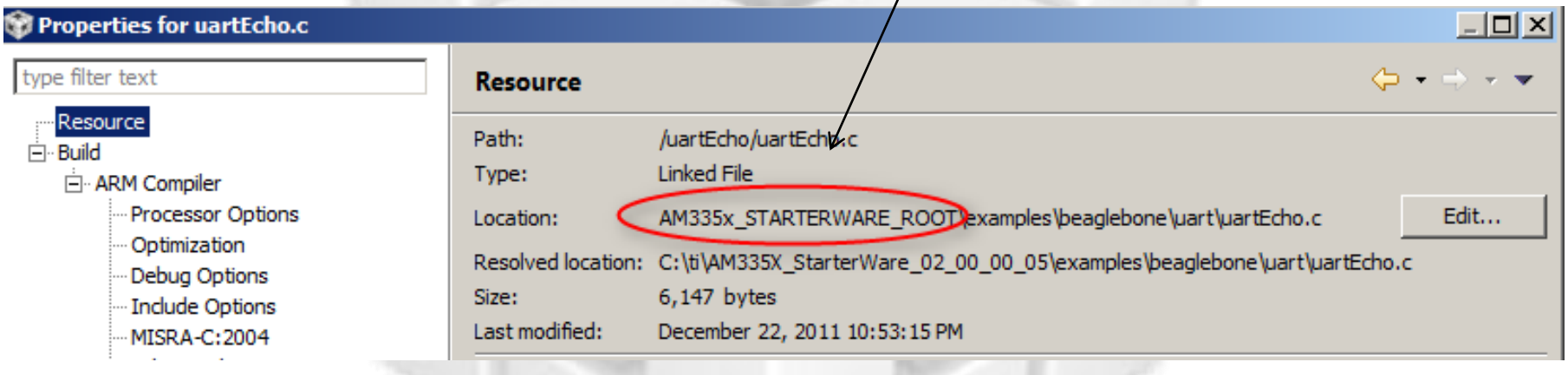




Link Files to Project

1. **Right-click on the C source file and check the *Properties***

See how the *Location* parameter references the Linked Resource Variable



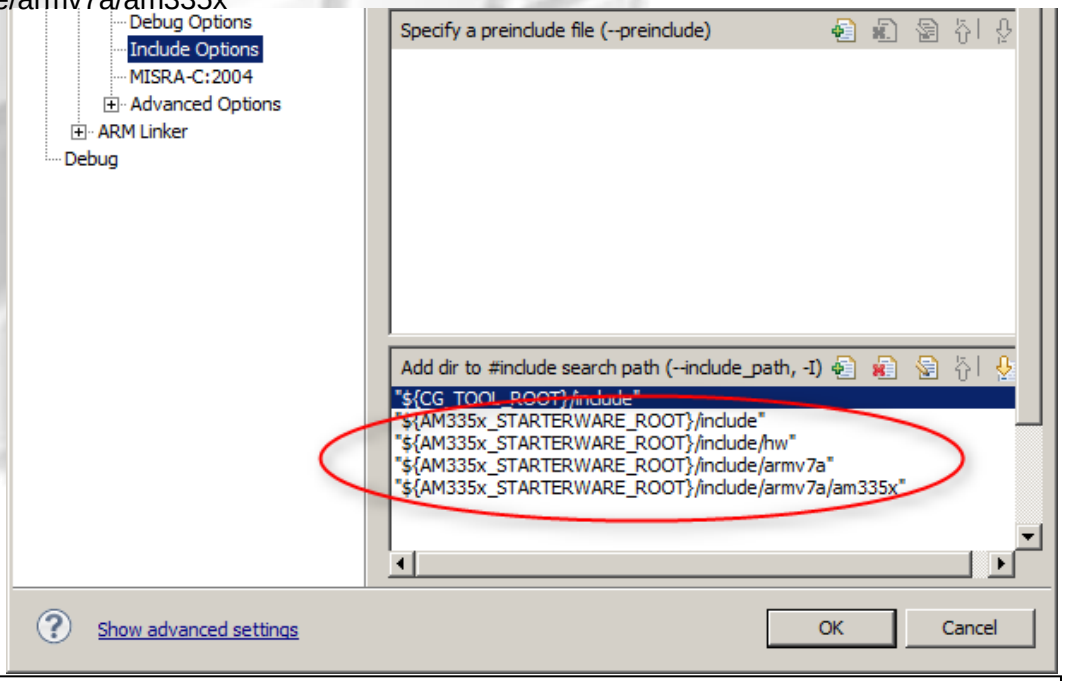
Modifying Project Properties

Here we are adding paths to include files using the Build Variable

1. **Right-click on the project and select *Properties***
2. **In the compiler *Include Options*, add the following entries to the list of include search paths:**
   - ${AM335x_STARTERWARE_ROOT}/include
   - ${AM335x_STARTERWARE_ROOT}/include/hw
   - ${AM335x_STARTERWARE_ROOT}/include/armv7a
   - ${AM335x_STARTERWARE_ROOT}/include/armv7a/am335x
3. **Click *OK***

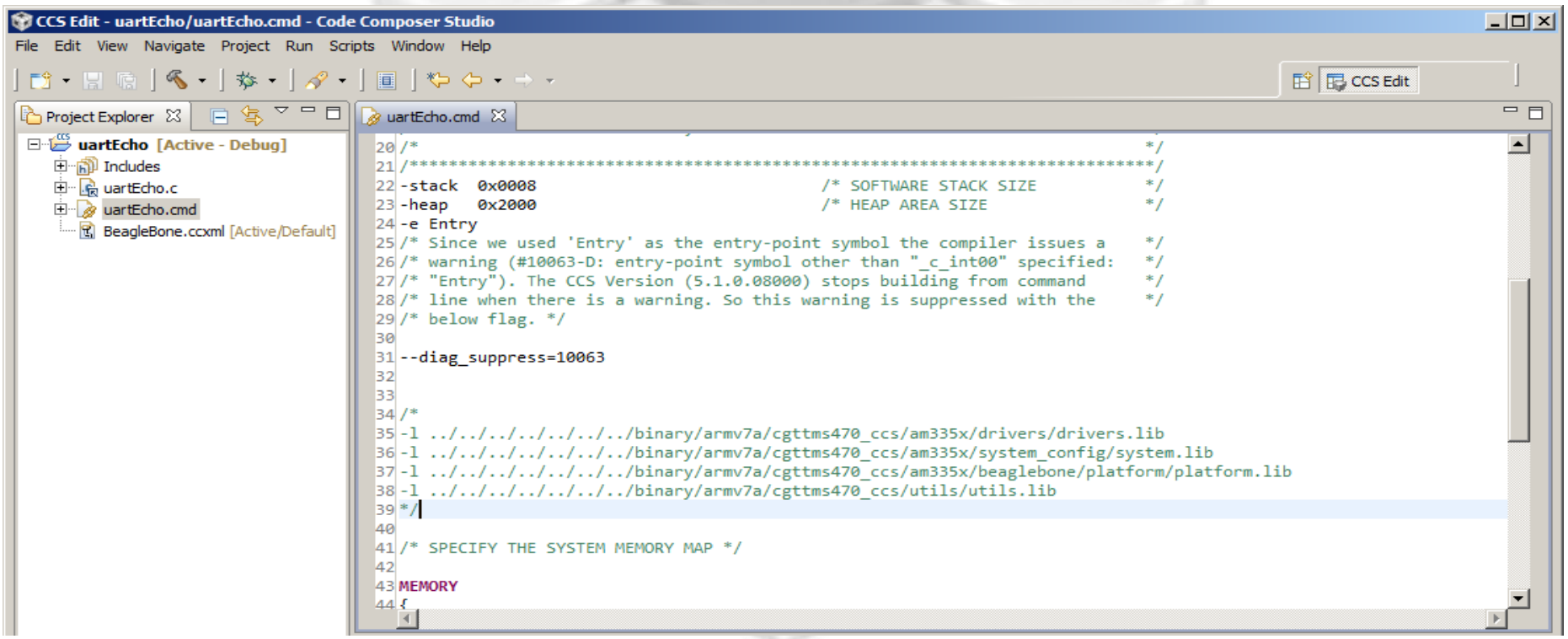
'${<BUILD VARIABLE>}' is the syntax to use a Build Variable in the project properties

Linked Resource Path Variables are only used when linking source files to a project. They cannot be used for build options. Use Build Variables when modifying build options



Modifying Project Properties

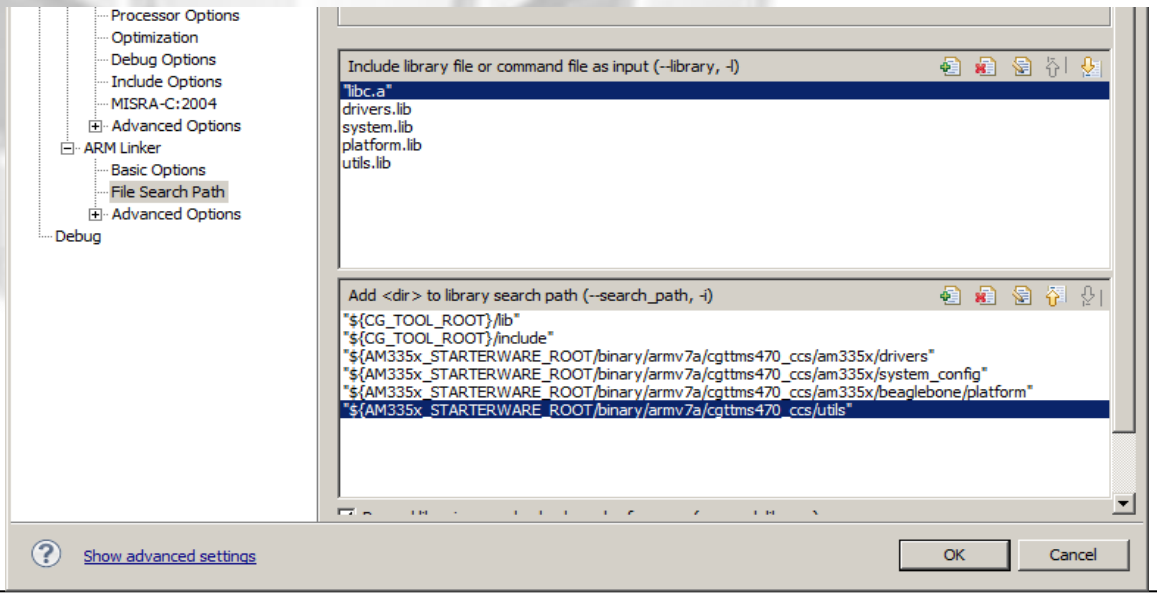
1. Double-click on the file <uartEcho.cmd> to open it in the editor
2. Comment out lines 68 through 71 (that specify the libraries to link in)
3. Save the file



Modifying Project Properties

Here we are adding paths to libraries using the Build Variable

1. **Right-click on the project and select *Properties***
2. **In the Linker *File Search Path*, add the following entries under *--search_path* option:**
   - ${AM335x_STARTERWARE_ROOT}/binary/armv7a/cgt_ccs/am335x/drivers
   - ${AM335x_STARTERWARE_ROOT}/binary/armv7a/cgt_ccs/am335x/system_config
   - ${AM335x_STARTERWARE_ROOT}/binary/armv7a/cgt_ccs/am335x/beaglebone/platform
   - ${AM335x_STARTERWARE_ROOT}/binary/armv7a/cgt_ccs/utils
3. **In the Linker *File Search Path*, add the following entries under *--library* option:**
   - drivers.lib
   - system.lib
   - platform.lib
   - utils.lib



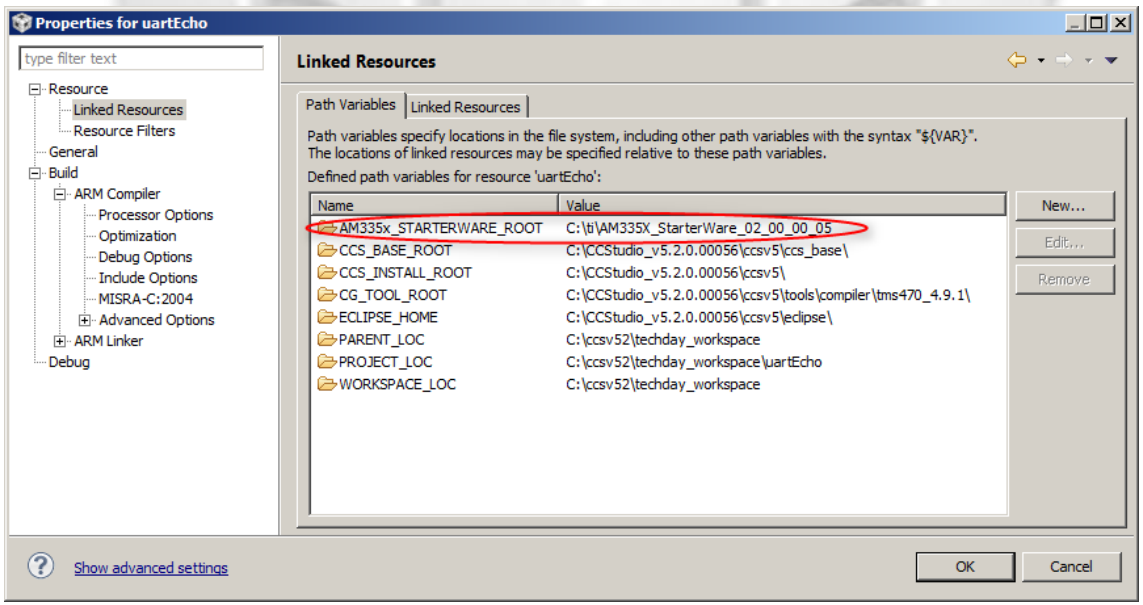
Project Properties

1. **Go to *Resource* → *Linked Resources* to see all the Linked Resource Path Variables that are available to the project**

ⓘ This will show all variables created at the project level and workspace level

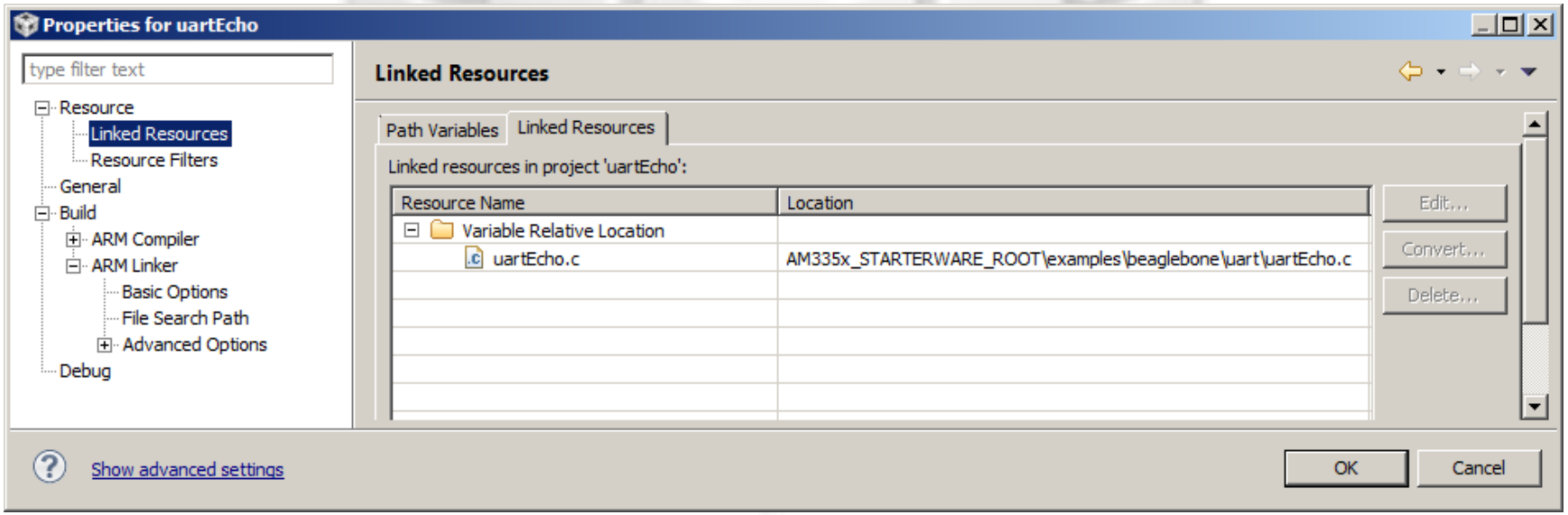
2. **Check if the workspace level Linked Resource Path Variable that was created appears in the list**

ⓘ Variables may be edited here but changes will only be recorded at the project level (stored in the project files)



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**



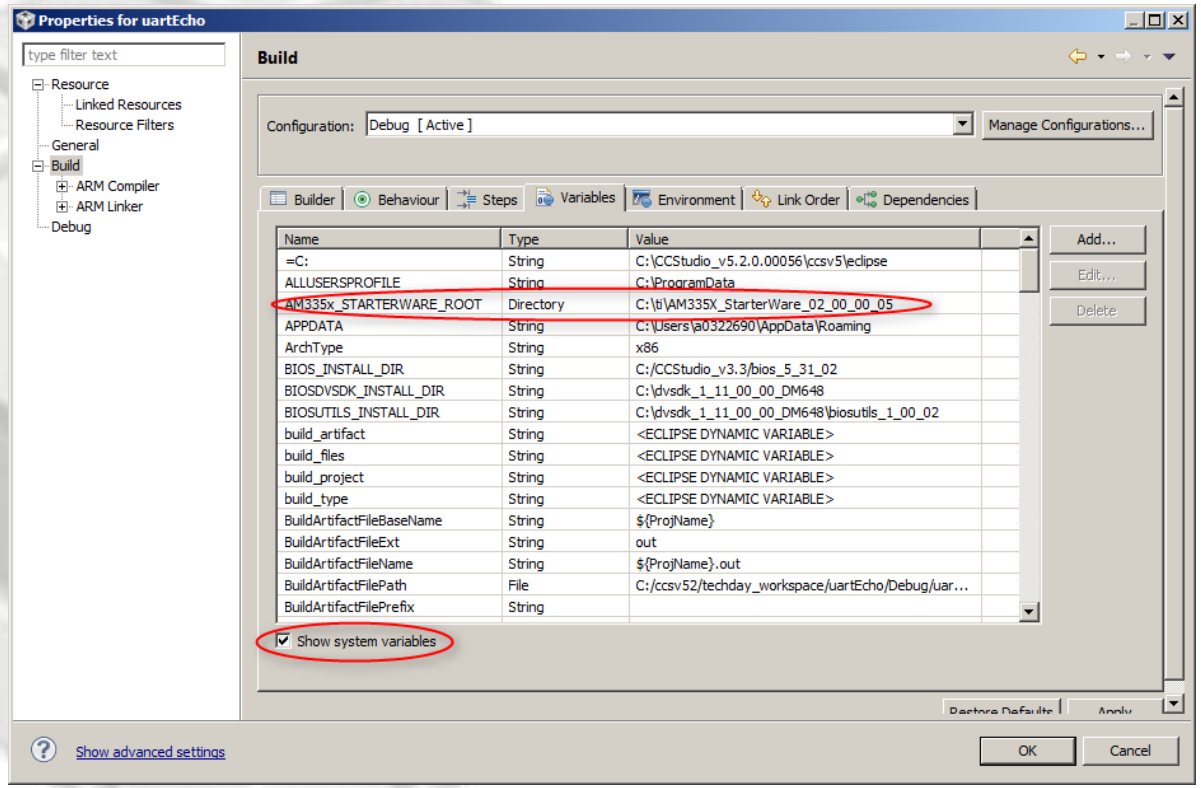
Project Properties

1. **Go to *Build* → tab *Variables* to see all the Build Variables that are available to the project**

   Only project level variables will be listed by default

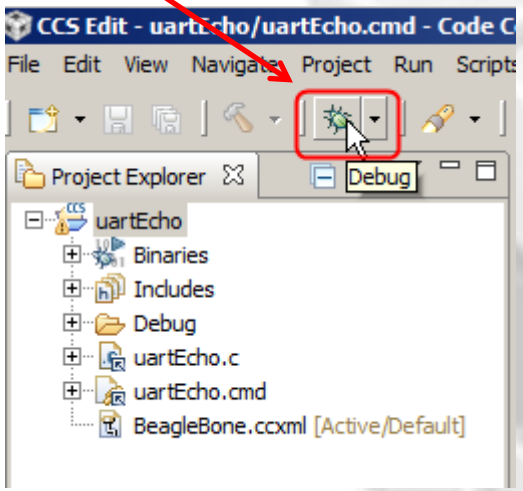
2. **Enable the *Show system variables* checkbox to see variables set at the workspace and system level**

   See how the workspace level Build Variable that was created appears in the list



Build and Load the Program

1. Make sure the ‘uartEcho’ project is in focus. Then use the *Debug* button to build and load the code

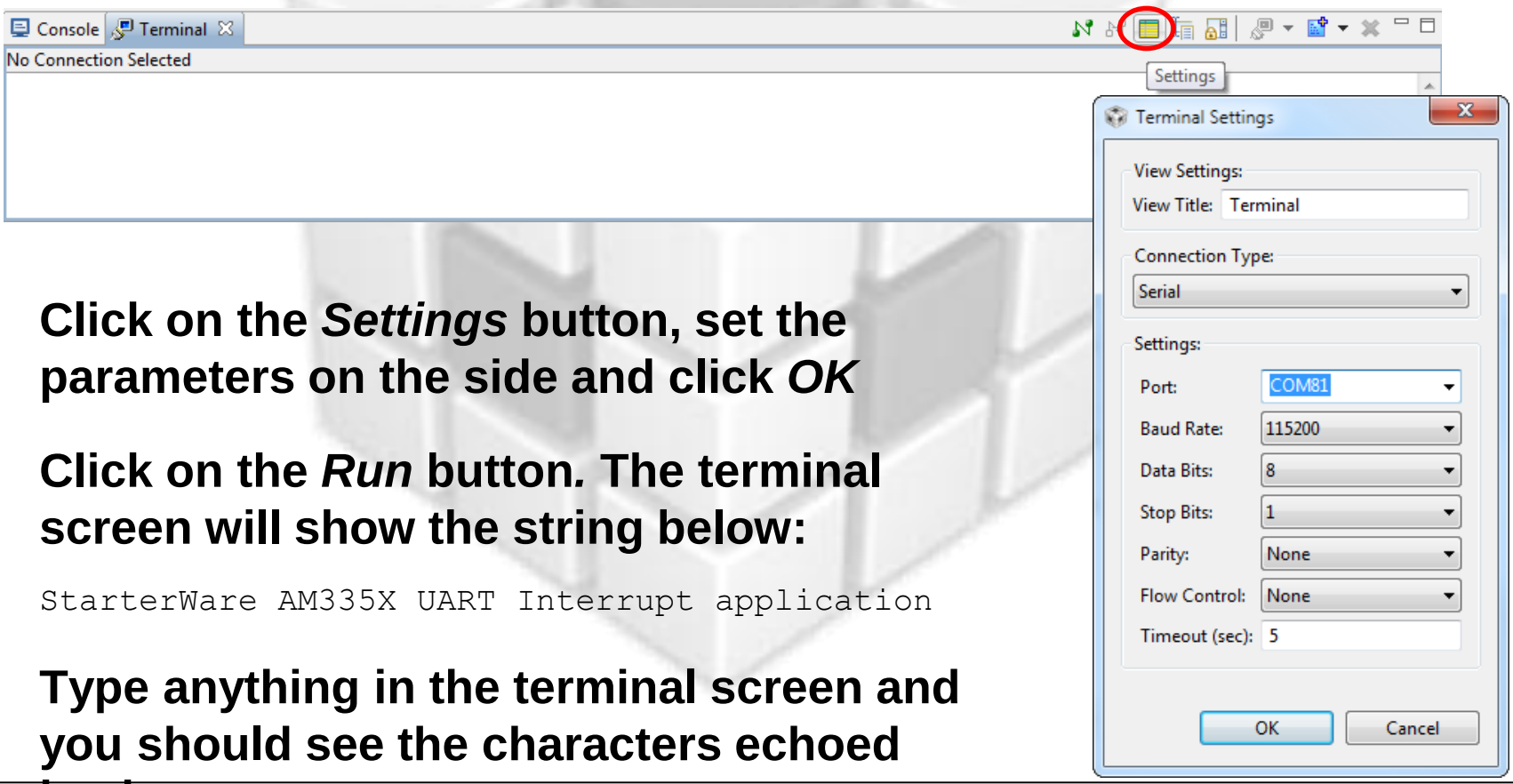


2. When the Debug Session is launching, CCS asks which cores to load. De-select **M3_wakeupSS_0**

This happens because BeagleBone has two cores compatible with the project (ARM cores)

Seeing the output on the terminal plug-in

1. **Go to menu** *Window* → *Show View* → *Other*
2. **Type 'Terminal' in the text box, select the** *Terminal* **entry and click** *OK*



3. **Click on the *Settings* button, set the parameters on the side and click *OK***
4. **Click on the *Run* button. The terminal screen will show the string below:**

```
StarterWare AM335X UART Interrupt application
```

5. **Type anything in the terminal screen and you should see the characters echoed back.**

UART Echo Example: Exercise Summary

- **Congratulations! You finished all the labs.**
- **At this point you experimented the following concepts:**
  - Portable Projects
  - Linked resources
  - Linked resource path variables
  - Build variables
- **Please don't forget to answer the evaluation form before leaving; it is very important for us to improve this session.**
- **Also, make sure to save any files and projects you want to take home.**
- **Thank you, enjoy the rest of your day and have a safe trip back.**