|  |
| --- |
| Texas Instruments |
| Keystone Multicore Workshop |
| Lab Manual |

Contents

[Lab 1 – SRIO Loopback Direct IO 1](#_Toc330541551)

[Purpose 1](#_Toc330541552)

[Project Files 1](#_Toc330541553)

[Task 1: Import the Example Project 1](#_Toc330541554)

[Task 2: Set/Verify the Project Properties 2](#_Toc330541555)

[Task 3: Build the Project 2](#_Toc330541556)

[Task 4: Connect to the Target EVM 3](#_Toc330541557)

[Task 5: Load and Run the Program 4](#_Toc330541558)

[Lab 2 – Hyperlink 5](#_Toc330541559)

[Purpose 5](#_Toc330541560)

[Project Files 5](#_Toc330541561)

[Task 1: Import the Example Project 5](#_Toc330541562)

[Task 2: Set the Project Properties 6](#_Toc330541563)

[Task 3: Build the Project 7](#_Toc330541564)

[Task 4: Connect to the EVM 7](#_Toc330541565)

[Task 5: Load and Run the Program 8](#_Toc330541566)

[Task 6: Increase the Transfer Rate 8](#_Toc330541567)

[Task 7 (Optional): Board-to-board Hyperlink Example 9](#_Toc330541568)

[Lab 3 – SRIO Type 11 10](#_Toc330541569)

[Purpose 10](#_Toc330541570)

[Project Files 10](#_Toc330541571)

[Task 1: Load the Project 11](#_Toc330541572)

[Task 2: Build the Application 13](#_Toc330541573)

[Task 3: Launch the Debugger 14](#_Toc330541574)

[Task 4: Load and Run 16](#_Toc330541575)

[Task 5 (Optional): Debug the Project 17](#_Toc330541576)

[Lab 4 – Optimization Exercise 18](#_Toc330541577)

[Purpose 18](#_Toc330541578)

[Project Files 18](#_Toc330541579)

[Task 1: Build and Run the Project 18](#_Toc330541580)

[Task 2: Compiler Optimization 19](#_Toc330541581)

[Task 3: Enable Software Pipelining 19](#_Toc330541582)

[Task 4: Align the Data 20](#_Toc330541583)

[Task 5: Cache Considerations 20](#_Toc330541584)

# Lab 1 – SRIO Loopback Direct IO

## Purpose

The purpose of this lab is to demonstrate how to build and run a very basic Code Composer Studio v5 project on the C6678 EVM using the Direct IO Loopback example delivered with the MCSDK.

## Project Files

The exact location of the project files will depend on where the MCSDK was installed and which version you are using. They can be found here:

<MCSDK\_DIR>\pdk\_C6678\_<your pdk version>\packages\ti\drv\exampleProjects\DRIO\_LoopbackDioIsrexampleproject

### Task 1: Import the Example Project

1. Open CCS.
2. Set the Perspective to CCS Edit.
3. Import the project.
   * Project | Import Existing CCS/CCE Eclipse Project
   * Select search directory to

<MCSDK\_DIR>\pdk\_C6678\_<your pdk version>\packages\ti\drv\exampleProjects

* + From the list of Discovered projects, choose SRIO\_LoopbackDioIsrexampleproject and then click Finish.

1. SRIO\_LoopbackDioIsrexampleproject should now appear in your Project Explorer.

### Task 2: Set/Verify the Project Properties

1. Select the SRIO\_LoopbackDioIsrexampleproject.
2. Right click and select Properties.
3. Select General and choose the Main Tab.
4. Set the following Device Properties.
   * Device Family = C6000
   * Variant = Generic C66x Device
5. Under Build/C6000 Compiler, select Basic Options and set the following compiler debug properties:
   * Target processor version = 6600
   * Debugging model = Full symbolic debug
   * Optimization level = 0
   * Optimize for code size = 0
6. Click OK.

### Task 3: Build the Project

1. Select SRIO\_LoopbackDioIsrexampleproject.
2. Build the project.
   * Project | Build Project  
     OR
   * Right Click and select Build Project
3. Verify that the build was successful.

Was the file SRIO\_LoopbackDioIsrexampleproject.out generated?

1. From the CCS Edit perspective, check the Binaries or Debug directory. From the CCS Debug perspective, check the Console.

### Task 4: Connect to the Target EVM

1. Set the Perspective to CCS Debug.
2. Create a new User-Defined Target:
   * View | Target Configurations
   * Select User Defined
   * Click the New Target button or Right-click and select New Target Configuration
3. Define the C6678L/LE EVM as a new target:
   * File name = EVM6678L or EVM6678LE
   * Location = <local>\ti\CCSTargetConfigurations
   * Select the emulator type in the connection drop-down menu
     1. If you are using the on-board XDS100, you should choose TI XDS100v1
     2. If you are using a Mezzanine Emulator, you should choose Blackhawk XDS560v2 Mezzanine Card
   * Specify the Board or Device by checking the appropriate box (TMS320C6678)
   * Click the “Advanced” tab at the bottom of the screen and add the appropriate GEL file for core 0 by selecting Core 0 and choosing the initialization file.
     1. The GEL file is located here:

<CCS\_INSTALL\_DIR>\ccsv5\ccs\_base\emulation\boards\evm6678l\gel\evmc6678.gel

* + Click Finish

1. Make sure the EVM is powered ON and connect your PC/laptop to the emulator port on the EVM using the provided USB cable.
2. Launch the target configuration (e.g., EVM6678LE.ccxml).
   * Select the target.
   * Right click and select Launch Selected Configuration.
3. Select Core 0, right click, and select Connect Target.

### Task 5: Load and Run the Program

1. Select Core 0 and load the .out file created earlier in the lab.
   * Run | Load | Load Program
   * Click Browse Project
   * Select SRIO\_LoopbackDioIsrexampleproject.out and Click OK.
   * Click OK to load the application to the target (Core 0)
2. Run the application.

Did the application execute successfully?

1. Check the console.

# Lab 2 – Hyperlink

## Purpose

The purpose of this lab is to demonstrate how to build and run a HyperLink loopback application on the C6678 EVM using the example code as delivered with MCSDK. In addition, you will make modifications to the application parameters to vary the transfer rate. Optionally, you will run the application on two boards and modify the transfer rate to determine the maximum throughput allowed with this example configuration.

## Project Files

The exact location of the project files will depend on where the MCSDK was installed and which version you are using. They can be found here:

<MCSDK\_DIR>\pdk\_C6678\_<your pdk version>\packages\ti\drv\exampleProjects\hyplnk\_exampleproject

### Task 1: Import the Example Project

1. Open CCS.
2. Set the Perspective to CCS Edit.
3. Import the project.
   * Project | Import Existing CCS/CCE Eclipse Project
   * Select search\_directory: \pdk\_C6678\_1\_0\_0\_19\packages\ti\drv\exampleProjects
   * From the list of Discovered projects, choose hyplnk\_exampleproject and then click Finish.
4. hyplnk\_exampleproject should now appear in your Project Explorer.

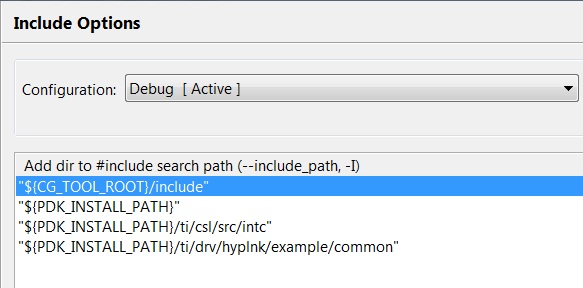
How many lanes are configured?

What is the baud rate?

1. Refer to hyplnkLLDCfg.h

### Task 2: Set the Project Properties

1. Select the hyplnk\_exampleproject.
2. Right click and select Properties.
3. Select General and choose the Main Tab.
4. Set the following Device Properties.
   * Device Family = C6000
   * Variant = Generic C66x Device
5. Under Build/C6000 Compiler, select Basic Options and set the following compiler debug properties:
   * Target processor verison = 6600
   * Debugging model = Full symbolic debug
   * Optimization level = 0
   * Optimize for code size = 0
6. Click OK.
7. Under Build/C6000 Compiler, select Include Options and verify the following paths:



### Task 3: Build the Project

1. Select hyplnk\_exampleproject.
2. Build the project.
   * Project | Build Project  
     OR
   * Right Click and select Build Project
3. Verify that the build was successful.

Was the file hyplnk\_exampleproject.out generated?

1. From the CCS Edit perspective, check the Binaries or Debug directory. From the CCS Debug perspective, check the Console.

### Task 4: Connect to the EVM

1. Set the Perspective to CCS Debug.
2. Create a new User-Defined Target:
   * View | Target Configurations
   * Select User Defined
   * Click the New Target button or Right-click and select New Target Configuration
3. Define the C6678L/LE EVM as a new target:
   * File name = EVM6678L or EVM6678LE
   * Location = <local>\ti\CCSTargetConfigurations
   * Click Finish
4. Make sure the EVM is powered ON and connect your PC/laptop to the emulator port on the EVM using the provided USB cable.
5. Launch the target configuration (e.g., EVM6678LE.ccxml).
   * Select the target.
   * Right click and select Launch Selected Configuration.
6. Select Core 0, right click, and select Connect Target.

### Task 5: Load and Run the Program

1. Load the .out file created earlier in the lab.
   * Run | Load | Load Program
   * Click Browse Project
   * Select hyplnk\_exampleproject.out and Click OK.
   * Click OK to load the application to the target.
2. Run the application.

Did the application execute successfully?

1. Check the console.

### Task 6: Increase the Transfer Rate

1. Set the Perspective to CCS Edit.
2. Modify the example code for hyplnk\_exampleproject
   * Open the readme.txt and determine which file contains parameters that define the reference clock, the serial data rate, the number of lanes, and loopback.
   * Open the file within the project.
   * Change the Baud Rate to 12.5 Gbaud
3. Build the code, load to Core 0, and run.

Did the application execute successfully at the new rate?

1. Check the console.

### Task 7 (Optional): Board-to-board Hyperlink Example

Modify the example to run the HyperLink application on two EVMs.

Hardware requirements:

* + Two C66x EVMs
  + One HyperLink cable
  + Connector Board

1. Modify the example code for hyplnk\_exampleproject
   * Open hyplnkLLDCfg.h
   * Search for “#define hyplnk\_EXAMPLE\_LOOPBACK”
   * Comment out this command.
   * Change the Baud Rate back to 6.25 Gbaud
2. Build the code, load to both targets, and run only on Core 0.

Did the application execute successfully?

1. Check the console.
2. Modify the example code for hyplnk\_exampleproject
   * Open hyplnkLLDCfg.h
   * Change the Baud Rate to a higher rate.
3. Build the code, load to both targets, and run only on Core 0.

Did the application execute successfully?

What is the highest transfer rate that can be achieved using this example?

# Lab 3 – SRIO Type 11

## Purpose

The purpose of this lab is to demonstrate how to use Type 11 SRIO in an application.

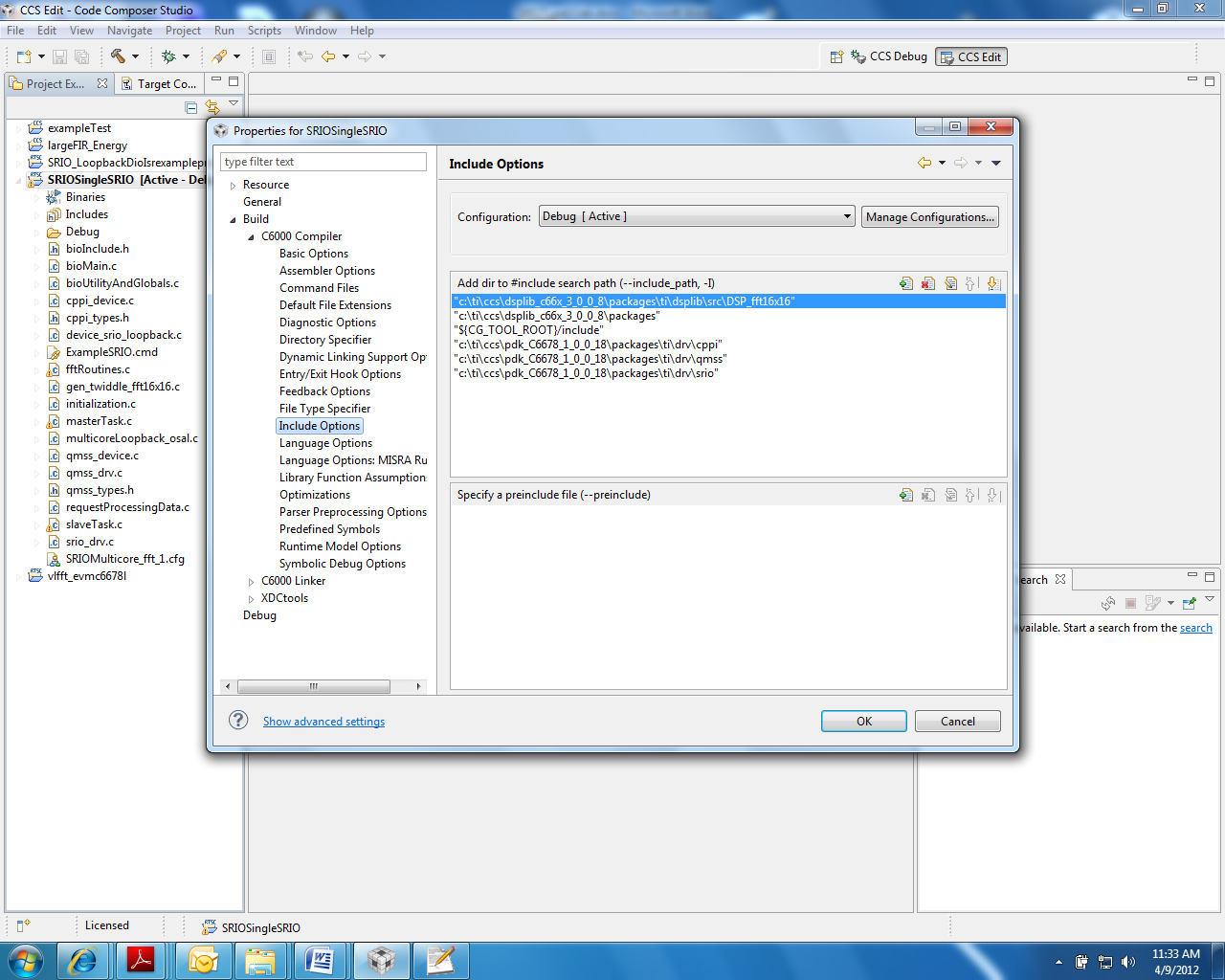
## Project Files

The following files are used in this lab

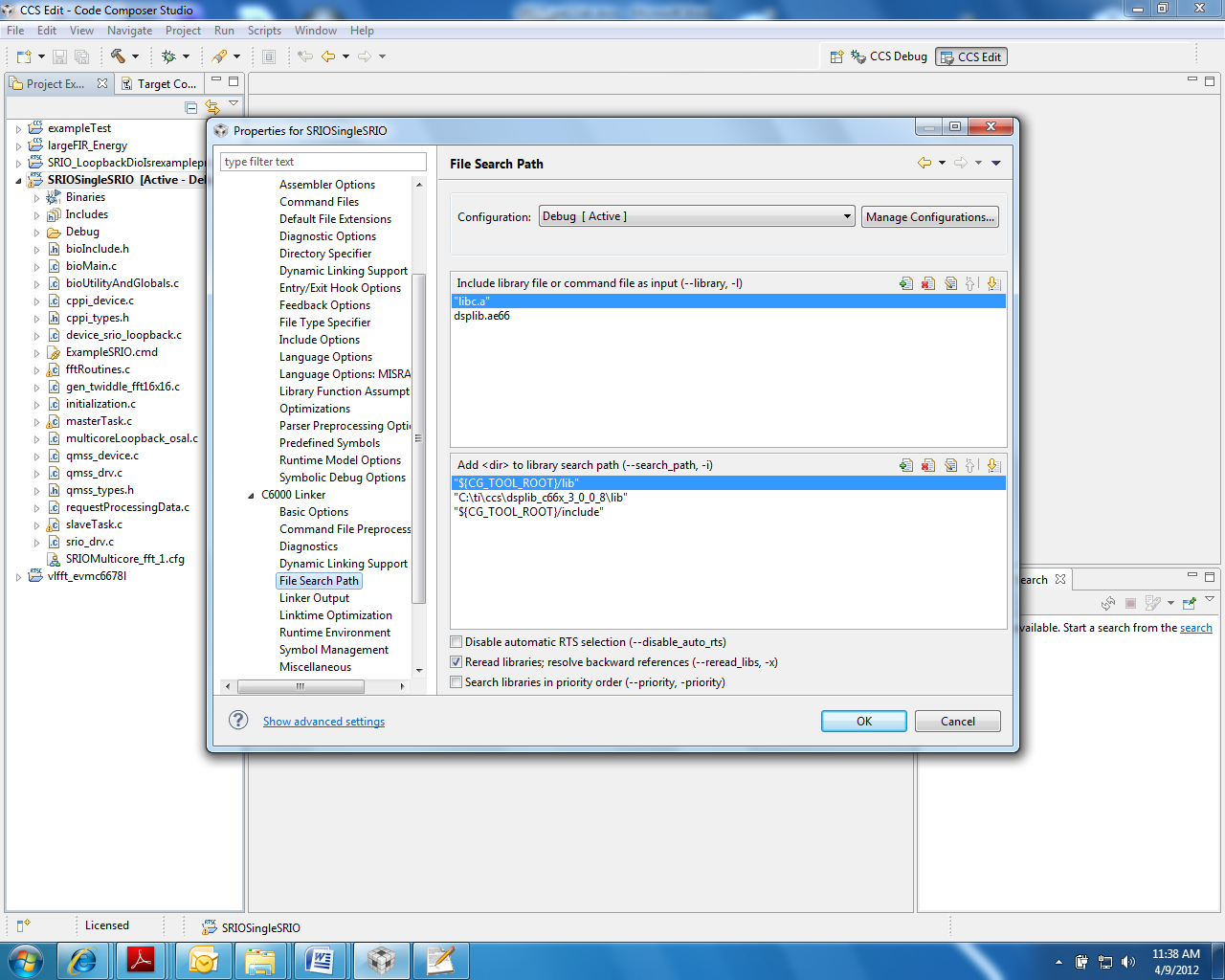
* bioInclude.h
* bioMain.c
* bioUtilityAndGlobals.c
* cppi\_device.c
* cppi\_types.h
* device\_srio\_loopback.c
* ExampleSRIO.cmd
* fftRoutines.c
* gen\_twiddle\_fft16x16.c
* initialization.c
* masterTask.c
* multicoreLoopback\_osal.c
* qmss\_device.c
* qmss\_drv.c
* qmss\_types.h
* requestProcessingData.c
* slaveTask.c
* srio\_drv.c
* SRIOMulticore\_fft\_1.cfg

## Task 1: Load the Project

1. Copy the project folder to your local development environment. The instructor will pass around a USB drive or point you to a location where the folder can be downloaded.
2. Start CCS and import the project.
3. Update the include path in the Project Properties. Refer to the path as defined below. You must modify all paths to either an absolute address where you put your tools, or refer to a relative address. You can use {PDK\_INSTALL\_PATH} or similar if it is defined.

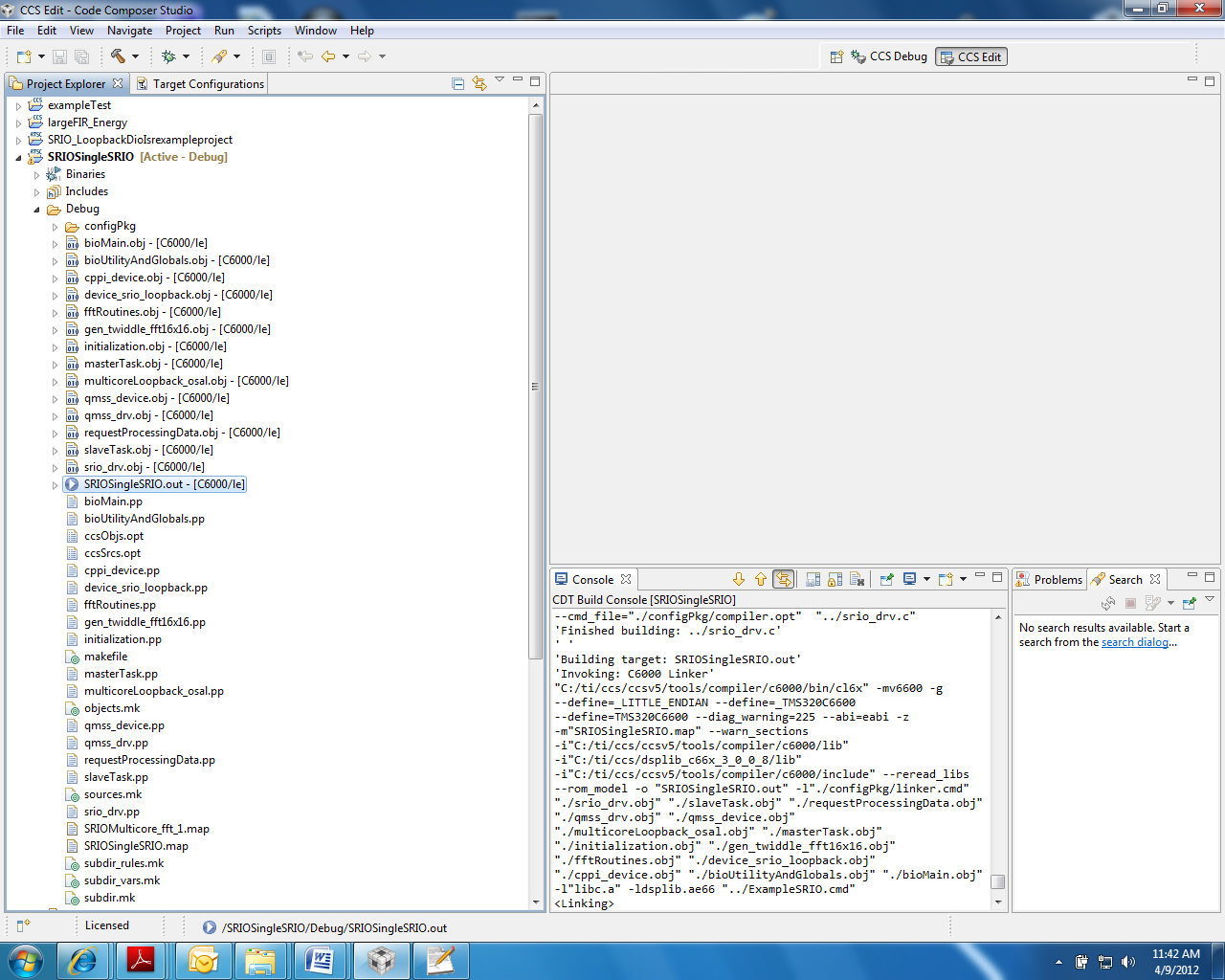


1. Do the same for the Linker File Search Path, as shown below.



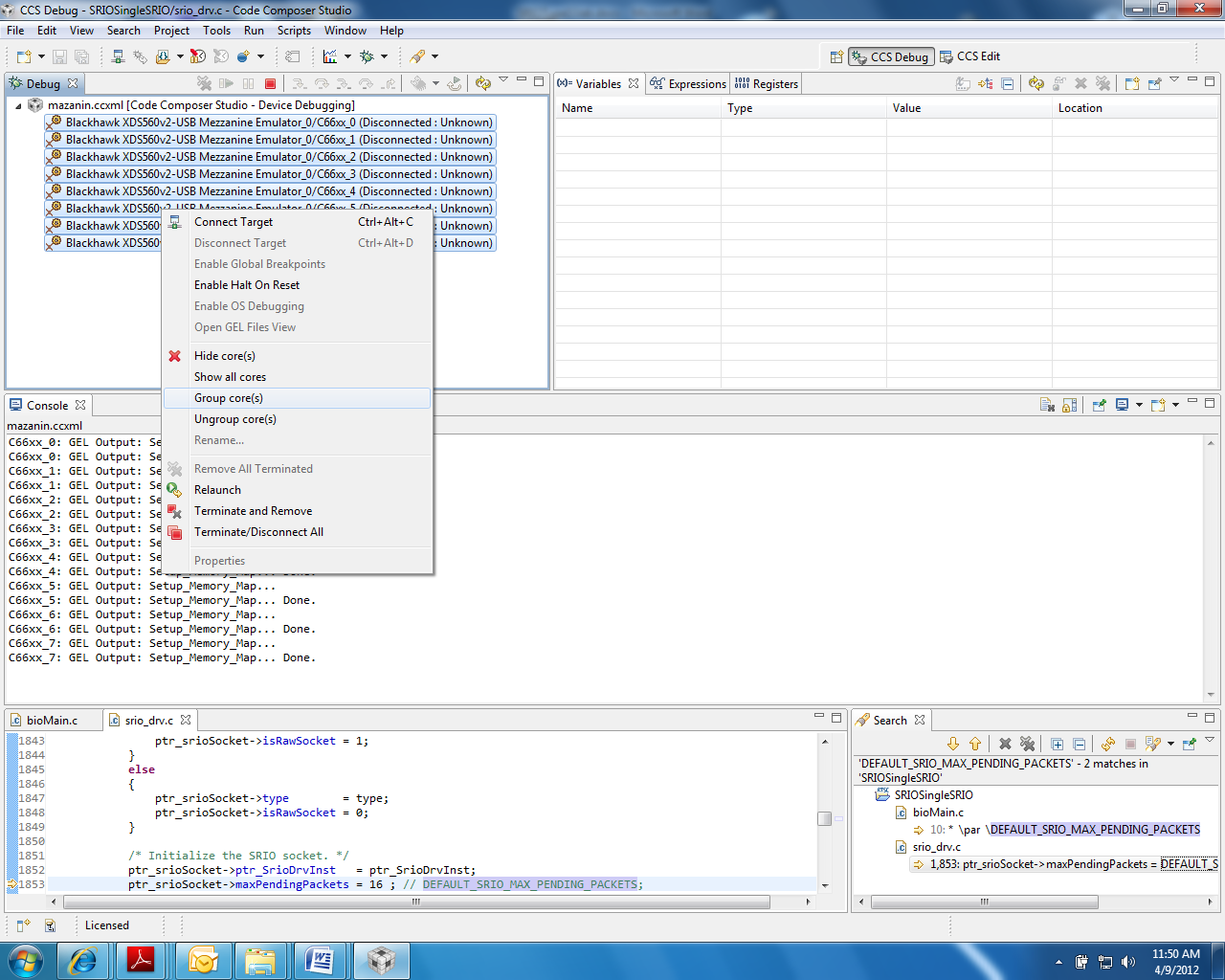
### Task 2: Build the Application

1. Clean the build.
2. Build the project.
3. Verify that the executable (.out) was built by looking at the debug directory (assuming the build configuration is debug configuration).

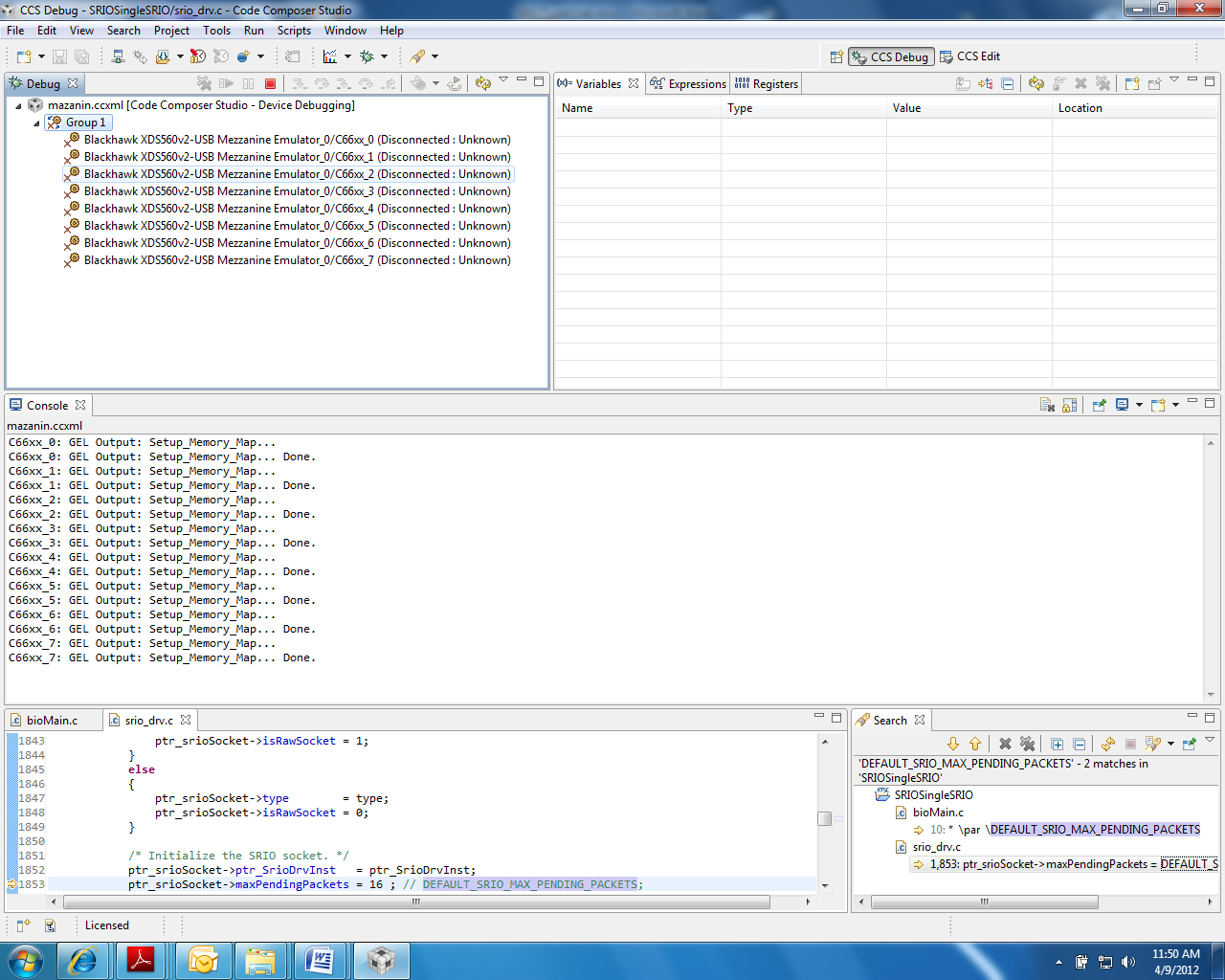


### Task 3: Launch the Debugger

1. Power on the EVM, connect the USB cable to the emulator, wait for the EVM to finish boot (the red light is ON).
2. Launch your debugger.
3. Group all cores into one group as follows:



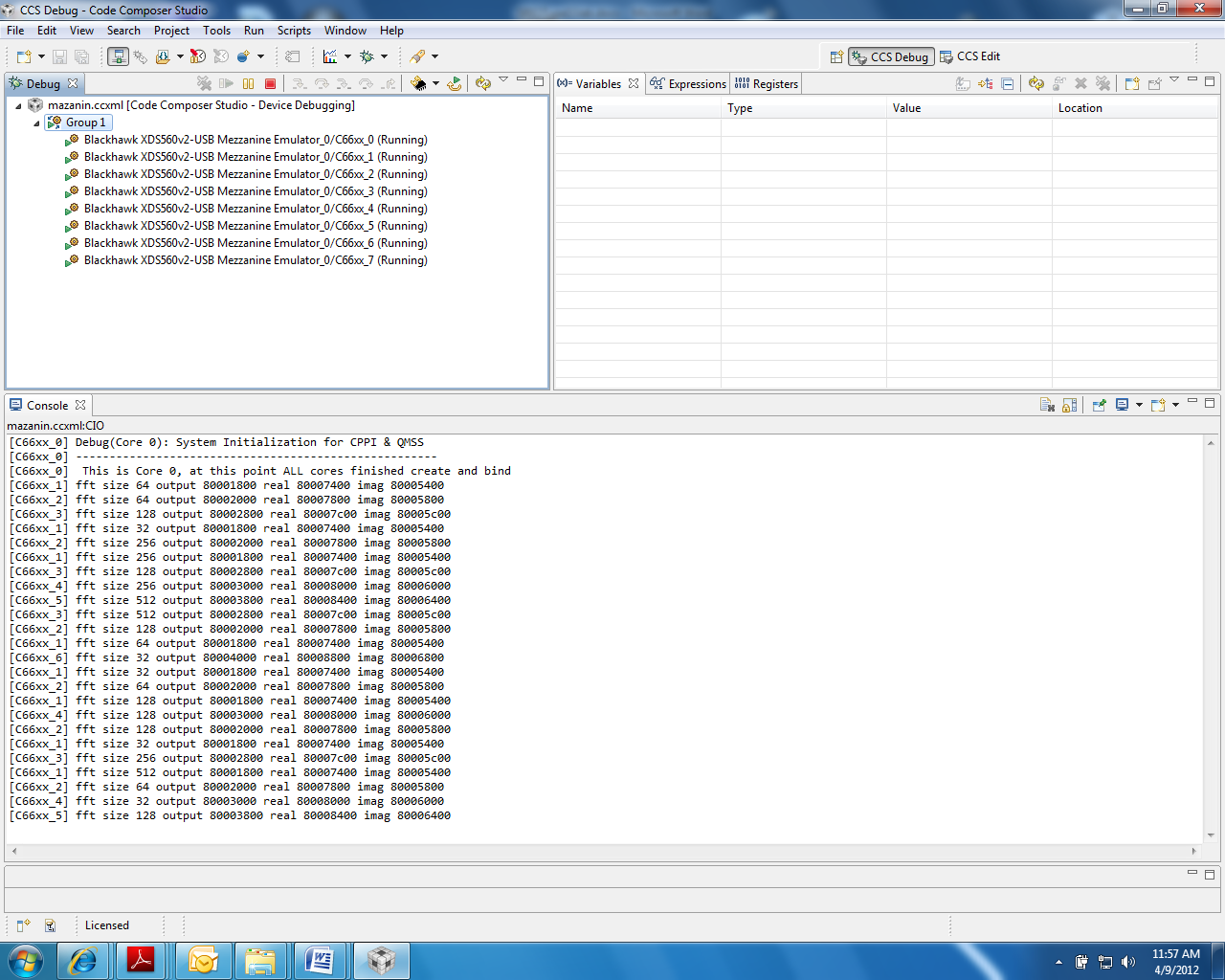
After grouping, Group 1 is defined and displayed as shown below:



### Task 4: Load and Run

1. Select Group1, and connect all cores in the group by one of the three ways:
   * From the RUN menu, select Connect Target.
   * Right click on the group name and choose Connect Target.
   * Click the Connect Target icon.
2. Load the code to all cores in the group:
   * From the RUN menu, select Load.  
     OR
   * Click the Load icon.
3. Run the code in one of the following ways:
   * Press F8.
   * From the RUN menu, select Resume.
   * Click on the Resume icon (green arrow).

The output results appear as follows:



1. Observe the results, then suspend the run:
   * From the RUN menu, choose Suspend  
     OR
   * Click on the Suspend icon (the yellow “pause” lines)

### Task 5 (Optional): Debug the Project

When you have finished the previous task, you can optionally download the same project with some embedded bugs and then debug the code. Your instructor will point you to the location to download the buggy project.

1. Load the project to your development environment:
2. Start CCS and import the project.

Challenge Question: Can you debug the code and make it build and run properly?

1. There are two bugs. One bug is in the build process. Something is missing from the .cfg file.  
   The second bug is in the run. The code was originally developed for 6670 and then converted to 6678.

# Lab 4 – Optimization Exercise

## Purpose

The goal of this lab is to demonstrate some basic optimization techniques.

This lab executes on an EVM board, or can be used with the simulator in conjunction with the estimated cycle count.

## Project Files

The following files are used in this lab:

1. firMain.c
2. naturalCFilter.c
3. intrinsicCFilter.c
4. utilities.c
5. test.h
6. linkerNoRTSC\_L2.cmd

### Task 1: Build and Run the Project

1. Open CCS
2. Create new project
3. Create new project file location
4. Delete default main.c
5. Copy project files to new file. (Your instructor will tell you where to get the files from)
6. Look at the source code. Understand what the main code is doing.
7. Open Properties and set the debug model to full and optimization levels to minimum.
8. Set the include path to point to the PDK.
9. Start a debug project. Compile and link. Generate an out file
10. If the target is not defined yet, define a target. Assign gel file to core 0.
11. Launch the target debugger (emulation), connect and load the program into core 0
12. From the run menu, enable the clock
13. Run the code and record the cycles time for natural C function and for intrinsic function

### Task 2: Compiler Optimization

1. Change the project build option. Suppress all debug features and enable the highest time optimization.
2. Re-build and re-run.
3. Record the optimized project cycles time for natural C function and for intrinsic function.

How much improvement is noted for the natural C code?

How much improvement is noted for the intrinsic code?

What issues exist within the code, if any?

1. Do intrinsic functions better utilize the processor?

### Task 3: Enable Software Pipelining

1. Keep the assembly file. In the project properties build compiler tab go to assembly option and check the appropriate tab
2. Rebuild the code. Find the assembly file. Hint, if the build configuration is debug, the assembly files are in the debug directory.
3. Open the interisicCFilter.asm file.

Was the compiler able to schedule the software pipeline?

What are the general reasons that the compiler might not schedule the software pipeline?

1. Think about cases that can cause randomness in the execution timing.
2. Open the interisicCFilter.asm file
3. Analyze and answer the following question.

What reason can you see that the compiler might not be able to schedule the software pipeline?

1. Think about inline function
3. Substitute the intrinsic function instead of the regular function in all the loops
4. Re-build and re-run. Look at the intrinsicCFilter.asm. Did the compiler schedule software pipeline?
5. Record the optimized project cycles time for natural C function and for intrinsic function with software pipeline

### Task 4: Align the Data

1. In the intrinsicCFilter.c code, the data is read from the memory.

Challenge Question? What is the alignment of the input data? What is the alignment of the filter coefficients (in the stack)?

1. Find Pragma that align the data. What other ways there to align the data on 64 bit boundary?
2. Change the code to tell the compiler that the data is loaded from aligned memory
3. Re-build and re-run
4. Record the optimized project cycles time for natural C function and for intrinsic function with software pipeline and aligned load.

### Task 5: Cache Considerations

1. In test.h, change the number of elements to 4K, 8K and 16K
2. Record the cycle counts for each case

Challenge Question? Why is the non-linear jump in the performances

1. Think about cache trashing
2. Change the code to take full advantage to the cache
3. Break the data into chunks, and call each routine multiple times. Make sure to keep the sum between calls
4. Re-build and re-run
5. Record the final optimization cycle count.
6. Do you have any ideas how to further reduce execution time?