|  |
| --- |
| Texas Instruments |
| Keystone II Multicore Workshop |
| ARM-based Lab Manual |

August 2013

Contents

[Prerequisites 1](#_Toc371079152)

[Hardware 1](#_Toc371079153)

[Software 1](#_Toc371079154)

[Lab 1 – EVM board bring up and out of the box demonstration 2](#_Toc371079155)

[Purpose 2](#_Toc371079156)

[Workshop network 2](#_Toc371079157)

[Task 1: Prerequisites 3](#_Toc371079158)

[Task 2: Load and run standard Hello application 5](#_Toc371079159)

[Lab 2 – Build a new ARM program 9](#_Toc371079160)

[Projects and source code 9](#_Toc371079161)

[Purpose 9](#_Toc371079162)

[Task 1: Modify the file system 9](#_Toc371079163)

[Example simple code 9](#_Toc371079164)

[Unzip and decompress the file system and add the new executable 11](#_Toc371079165)

[Compressed and zip the new file system 12](#_Toc371079166)

[Reboot the EVM and run the new program 12](#_Toc371079167)

[Task 2(optional): Build U-boot, Boot Monitor and Kernel 12](#_Toc371079168)

[U-BOOT source code extraction and build instructions 13](#_Toc371079169)

[4. Boot Monitor (skern.bin) source code extraction and build instructions 14](#_Toc371079170)

[5. Linux Kernel and device tree source code extraction and build instructions 14](#_Toc371079171)

[Lab 3 – Boot Using NFS-mounted file system 16](#_Toc371079172)

[Purpose 16](#_Toc371079173)

[Task 1: Build a file system on a Linux host, use the NFS server 16](#_Toc371079174)

[Task 2: Configure U-BOOT to mount the file server and boot 16](#_Toc371079175)

[Task 3: Build a new C program in the file system, and debug it 17](#_Toc371079176)

[Lab 4 – Build, run and optimize DSP project using CCS 19](#_Toc371079177)

[Purpose 19](#_Toc371079178)

[II. Project Files 19](#_Toc371079179)

[Task 1: Build and Run the Project 19](#_Toc371079180)

[Task 2: Define the target 25](#_Toc371079181)

[Task 3: Connect to the EVM 26](#_Toc371079182)

[Task 4: Load and Run CASE 1 26](#_Toc371079183)

[Task 5: Use Optimization and disable symbol debug for the fir filter routine 27](#_Toc371079184)

[Task 6: Optimize Software Pipeline 29](#_Toc371079185)

[Task 7: Enable the cache 30](#_Toc371079186)

[Task 8: Running in parallel on multiple cores 31](#_Toc371079187)

[Lab 5 – Using MPM server to load and run DSP code 34](#_Toc371079188)

[Purpose 34](#_Toc371079189)

[II. Project Files 34](#_Toc371079190)

[Task 1: Build and Run the Project 34](#_Toc371079191)

[Task 2: Using MPM to load, run and observe results 36](#_Toc371079192)

[Lab 6 – Using CCS (on Linux) to build and run ARM code 38](#_Toc371079193)

[Purpose 38](#_Toc371079194)

[Prerequisites 38](#_Toc371079195)

[Task 1: CCS - Start a new cross compiler project and build it 38](#_Toc371079196)

[Task 2: GDB from the tera-terminal 42](#_Toc371079197)

[Move the Debug directory outside of CCS 42](#_Toc371079198)

[Task 3: Connect a SSH terminal into the target 44](#_Toc371079199)

[Task 4: Optional – Use gdb to debug the application 54](#_Toc371079200)

[Task 5: Configure the host proxy 58](#_Toc371079201)

### Prerequisites

Depends on the revision of the EVM, updating UCD and BMC is needed

## Hardware

1. Update BMC and UCD on EVMK2H (optional):
   1. The wiki page <http://processors.wiki.ti.com/index.php/EVMK2H_Hardware_Setup> gives instructions on how to detect if the board needs BMC (Baseboard Management Controller) update. It also instructs how to do the update the BCM using CCS.  
      NOTE: A PDF version of the wiki page (KeyStone2\_EVM\_hardwareSetUp.pdf) is also available. Ask your instructor.
   2. The user must also check the UCD Power Management version (see EVMK2H Hardware Setup at link above) and update if necessary.  
      NOTE: Instructions and scripts that show how to update the UCD are provided in the zip file XTCIEVMK2X-UCD-Update.zip (ask your instructor).

## Software

The following software packages must be pre-installed on the student Laptop before the workshop starts. Note, during the workshop the Laptop is attached to local network and has limited access to the Web:

1. Download the MCSDK and CCS
   1. For details regarding the instructions in this section, refer to the [MCSDK User Guide for KeyStone II](http://processors.wiki.ti.com/index.php/MCSDK_User_Guide_for_KeyStone_II).
   2. The latest release of MCSDK is found here:  
      <http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/index_FDS.html>
      1. For this lab you can use the Windows or the Linux version, depends on your laptop. Linux MCSDK was pre-installed on an Ubuntu server that will be used in some of the labs.
   3. From the same download page as the MCSDK, locate and download the latest CCS version and the emupack version that goes with the CCS. Follow the instructions on the page. Note, installing CCS requires licensing from TI.
2. Installing VNC Viewer

Vnc server that supports graphic interface was installed on the Ubuntu server. Each laptop must have a VNC viewer. Texas Instruments and many other corporations purchased global licenses for Real VNC enterprise users and it can be downloaded from internal software download site (EDS). Limited functionality Real VNC viewer is available as freeware from multiple sites.

1. Ftp client

FTP server is installed on the Ubuntu server. Moving files between the student Laptop and the Ubuntu server can be done with the enterprise version of Real VNC or (if the student uses a freeware real VNC) by using ftp client on the laptop. The student must confirm that ftp client is installed on the laptop.

1. For communication between the student PC and the EVM, the FTDI driver is required. As needed, download the 32-bit driver here: <http://www.ftdichip.com/Drivers/D2XX.htm>

# Lab 1 – EVM board bring up and out of the box demonstration

## Purpose

The purpose of this lab is to demonstrate how to boot and run a very basic hello world program using u-boot. Loading the kernel and file server and run a pre-built hello world

### Workshop network



The above chart shows the workshop network environment. There are up to 10 stations. Each station has a single EVM, one Laptop that is connected to the EVM via JTAG cable, called the first laptop in the station, and one laptop that is not connected- the second laptop in the station. Stations are numbered from 1 to 10 and so are the EVMs and the first and second laptops. All EVMs and students laptops are connected to the local network 192.168.0.XX via a switch or a router. All EVMs use wired connection. Depends on the availability of the router, the Laptop may have wire or wireless connection. The Ubuntu server is connected as well. The Ubuntu server (or a router) has access to an external network with a global IP that have access to the Web. The IP address may be given by DHCP server on the Ubuntu or DHCP server on a router.

### Task 1: Prerequisites

1. Loading U-Boot using CCS – Only if U-Boot has not been programmed into the flash. To verify if the U-BOOT was already loaded do the following:
   1. Set SW1 of the EVM to Off, Off, On, Off
   2. Open two terminals into the EVM using a single USB cable connected to the lower USB connection on the board. From serial port setting chose Baud rate of 115200
   3. Power up the EVM. If the U-BOOT is already burned into the flash, the terminal will look like the following:

U-Boot SPL 2013.01 (Apr 05 2013 - 14:12:32)

SF: Detected N25Q128A with page size 64 KiB, total 16 MiB

U-Boot 2013.01 (Apr 05 2013 - 14:12:32)

I2C: ready

DRAM: 1 GiB

WARNING: Caches not enabled

NAND: 512 MiB

Net: Ethernet PHY: 88E1111 @ 0x00

TCI6614-EMAC

Warning: failed to set MAC address

Hit any key to stop autoboot: 0

* 1. Instructions how to load U-Boot with CCS are given in the MCSDK User Guide getting started section starting on page 6. Summary of the instructions are given below:
  2. Optional only if no Uboot has been programmed. Then use CCS to load and run Uboot
  3. Set DIP Switch (SW1) to ARM no boot mode: 1 OFF 2 OFF 3 OFF 4 ON for Rev 1 EVM
  4. Power on the EVM and Launch target configuration: tci6638-evm.ccxml, connect CCS to the CortexA15\_1 target.
  5. Edit the loadlin-evm-uboot.js java script from the <release folder>/host-tools/loadlin folder.

**NOTE – In the following instructions (and in all the following labs), modify the paths correctly to match the installation location of the laptop that you use**

PATH\_LOADLIN = C:/ti/mcsdk\_linux\_3\_00\_00\_10/host-tools/loadlin

var pathUboot = PATH\_LOADLIN + "/u-boot.bin";

var pathSKern = PATH\_LOADLIN + "/skern.bin";

* 1. copy u-boot\*.bin and skern\*.bin from images folder to PATH\_LOADLIN folder and

rename it into u-boot.bin and skern.bin.

CCS, click View ==> scripting console

loadJSFile C:/ti/mcsdk\_linux\_3\_00\_00\_10/host-tools/loadlin/loadlin-evm-uboot.js

or on Linux:

loadJSFile /opt/ti/mcsdk\_linux\_3\_00\_00\_10/host-tools/loadlin/loadlin-evm-uboot.js

This will load, u-boot image to MSMC RAM at 0xc001000 and boot monitor image to MSMC RAM at address 0xc5f0000.

Make sure PC is currently pointing to 0xc001000.

* 1. Click Resume button on the CCS window to run u-boot.
  2. If necessary, program the SPI flash with the U-boot GPH image

1. Program SPI NOR flash with U-boot GPH image (always update Uboot for a new release)

Stop autoboot in SOC UART Console

Copy u-boot-spi.gph from the images in the release 3 to tftp server root directory and rename it to u-boot-spi-keystone-evm.gph.

make sure your tftp server is running. Then issue the following commands to U-Boot console.

setenv serverip -> your tftp address

dhcp 0xc300000 u-boot-spi-keystone-evm.gph

sf probe

sf erase 0 <size of u-boot-spi.gph in hex up rounded to sector boundary of 0x10000>

sf write 0xc300000 0 <size of u-boot-spi.gph image in hex>

Note that size of the image will be displayed as part of the dhcp command.

Note if u-boot-spi.gph was not copied to tftp root directory, then add path,e.g.: dhcp 0xc300000 release/u-boot-spi-keystone-evm.gph

Note when using the training standard server, u-boot-spi.gph should be copied to directory /var/lib/tftpboot/studentN where N is the student number. Then add path to studentN, e.g.: dhcp 0xc300000 studentN/u-boot-spi-keystone-evm.gph

1. The EVM has two (mini) USB ports. One of the ports access JTAG connection and can be used to connect CCS to the board. This USB connector is part of the emulator daughter (mezzanine) card. The second USB port is part of the mother board and can be used to connect two terminals into the EVM. Note that there are other connections that can be used to connect a serial port terminal to the EVM. We will refer to the serial terminal as Tera-Terminal (to distinguish from a window terminal on Ubuntu machines). The tera-terminals are connected using a single USB cable but can be opened as two tera-terminals. One is connected to the ARM terminal (the lower com port) and the second is connected to the FPGA on the board. The user must open the two tera-terminals and set the serial rate to 115200 Baud.
2. VNC into the Ubuntu server. The server IP will be given by the instructor. For static configuration, when DHCP is not available, the server IP is 192.168.0.100. The login instance for student N is :N. That is, for example, for static IP , student 3 wills VNC to address 192.168.0.100:3, while student number 7 will use 192.168.0.100:7. The VNC password for all students is “ vncserve ”

### Task 2: Load and run standard Hello application

1. The base location of the tftp server is defined in the tftp configuration file (/etc/xinetd.d/tftp) as the server\_args. The default setting is /var/lib/tftpboot/studentN where student is the user name (N is 1, 2, 3, …10). In order for the U-BOOT to get files from sub-directory, the subdirectory should be the tftp\_root value. For example, if all the files that are needed for U-BOOT are in directory /var/lib/tftpboot/studentN then (see below) the tftp\_root value of the U\_BOOT is studentN
2. Make a subdirectory /var/lib/tftpboot/studentN if it does not exist already and copy the MCSDK release binary images into this directory. The binary images are located in the following directory on the Ubuntu server:

/opt/ti/MCSDK\_3\_XX/mcsdk\_linux\_3\_00\_00\_xx\images

(XX is the release number, currently 10)

1. You can also move the images from your own release of MCSDK on your laptop using ftp into the same directory, /var/lib/tftpboot/studentN
2. U-BOOT loading and running Linux Kernel using TFTP with ramfs file system

First set the DIP switch (sw1) to ARM SPI boot mode – 1 OFF 2 OFF 3 ON 4 OFF (for Rev 1 EVM)

Power off EVM

disconnect CCS from EVM (if already connected)

Power up EVM , look at the ARM tera-terminal

* 1. It is very important to set the environment variables correctly for U-BOOT. Here are the instructions how to do so:

After power on, press ESC to stop autoboot in the ARM tera-terminal and set environment properly

env default -f -a

If the sizeof file system is larger than 9M bytes, update the value to match it, note the limitation is: 80 MBytes:

*setenv args\_ramfs 'setenv bootargs ${bootargs} earlyprintk rdinit=/sbin/init rw root=/dev/ram0 initrd=0x802000000,80M'*

*setenv name\_fs*  the name of the compressed file system to load, for example*, tisdk-rootfs.cpio.gz* (look at the release for an updated name)

*setenv name\_fdt*  the name of the binary device tree, uImage-k2hk-evm.dtb (look at the release for an updated name)

*setenv name\_kern*  the name of the kernel file, uImage-keystone-evm.bin (look at the release for an updated name)

*setenv name\_mon* the name of the kernel monitoring file, skern-keystone-evm.bin (look at the release for an updated name)

setenv serverip IP address of the TFTP server (The Ubuntu server, default ip for workshop is 192.168.0.100)

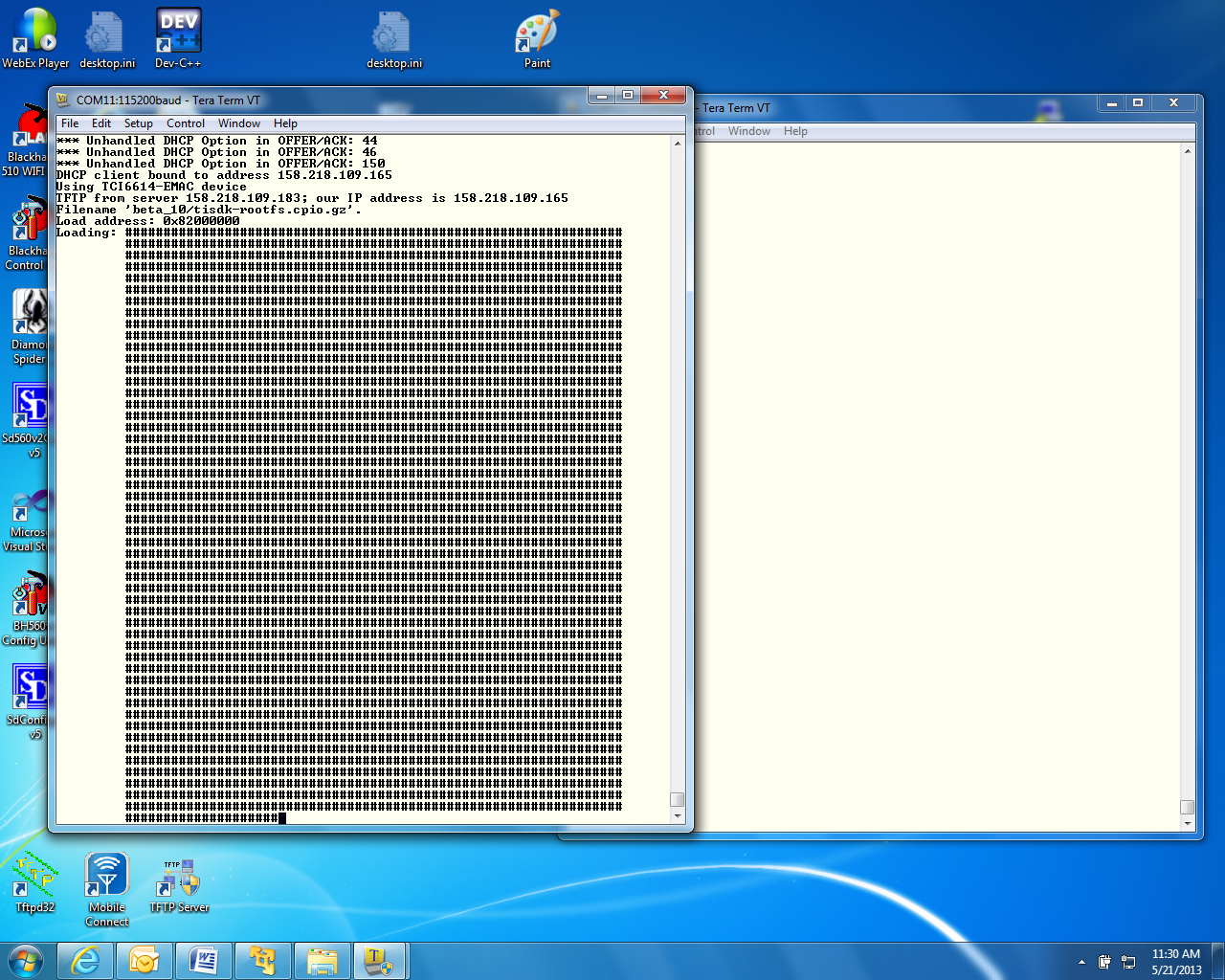
setenv boot ramfs

setenv tftp\_root studentN

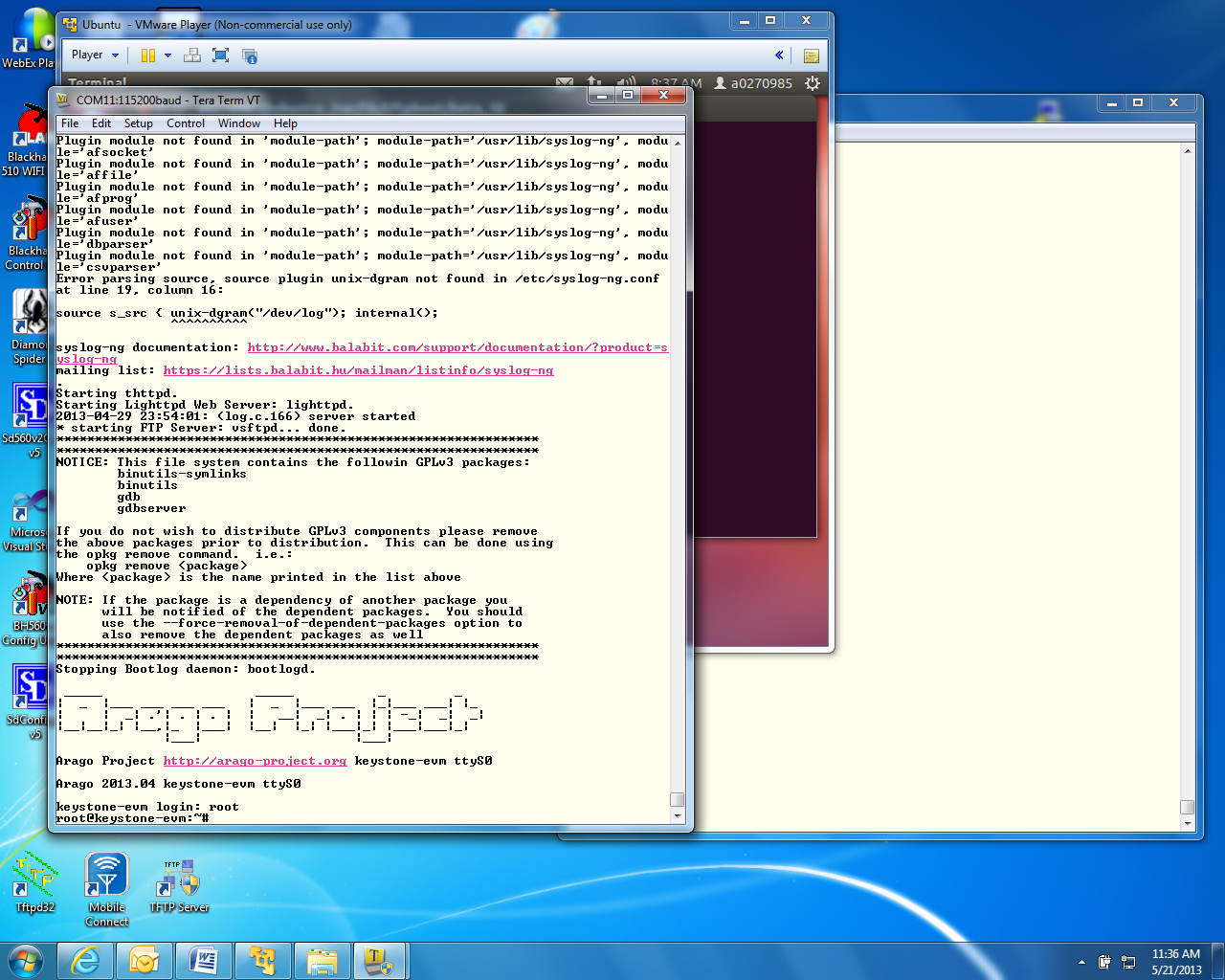
saveenv

boot (or start hardware or software reboot. Hardware reboot = power cycle, software reboot, write fullrst in the BMC terminal window.

The tera-terminal will start as follows:



1. When booting ends, log as root (no password)



1. Go to directory /usr/bin cd /usr/bin
2. Run the hello program ./hello
3. The program should print hello world on the tera-terminal

# Lab 2 – Build a new ARM program

## Projects and source code

Unless instructed by the instructor otherwise, all projects and source code are available on the server. Directory /usr/local/projects has two sub-directories, ARM and DSP. The source for this Lab and the next Lab is in the ARM sub-directory. DSP projects are in the DSP sub-directory. When the DSP projects are built using CCS on the student Laptop, the projects should be ftp from the server to the student laptop.

## Purpose

The purpose of this lab is to demonstrate how to build and run a simple ARM program, using all the development tools on Ubuntu system, load the new file server to the EVM and executes the code

### Task 1: Modify the file system

The first task of this Lab involved modifying the file system that was loaded into the EVM in the previous step. A complete build of all other images that are part of the release is an optional exercise. The instructions how to build the U-Boot, boot monitor and the kernel images are given in task 2.

Modifying the file system involves three steps. First a new main function is developed and using the cross compilers tools on the Ubuntu, the function is compiled and an executable is built.

Next the current compressed file system is unzipped and de-compressed into a temporary directory, and the new executable that was built in the previous step is added

Last the new file system is compressed, zipped, and moved to the tftp directory. The EVM is boot, and the new program is executed and produces the expected results.

### Example simple code

The instructor will provide you with a simple c program that does elementary calculations and print out some comments and the results of the calculations. For standard TI workshop, the example code is located on the server in /usr/local/projects/ARM directory. Assume that the example file name is example1.c. Copy this file to the student directory /home/studentN/temp

**Note – if the example code name is different, you have to substitute the correct name for example1 throughout the document**

If the temp directory does not exist, create it by using

cd ~/ Takes you to the home directory

sudo mkdir temp

Note – when sudo is used, the system may ask you for a password. (WsN where N is the student number)

/usr/local/studentStartScript.sh is a script that defines all the paths and exports for each individual user. The user must call this script for each new terminal:

source /usr/local/studentStartScript.sh

The Linaro toolchain and all other shared software are installed on the Ubuntu server ahead of time in directory /usr/local/. A path to the Linaro tool chain is defined in the script studentStartScript.sh.

To use the cross compiler to build the executable, cd the terminal to the directory where example1.c was stored and use the following command:

arm-linux-gnueabihf-gcc -o example1 –g example1.c

The cross compiler tools will compile the file and build an executable called example1 in the same directory where the terminal is.

To verify that the compilation was done for the ARM processor and not for the native Intel (or other) processors do the following:

sudo file example1

The results should show ARM architecture:

“example1: ELF 32-bit LSB executable, **ARM**, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.31, BuildID[sha1]=0x953dac672e7159d481d5a6d3bbb5356e5f870d21, not stripped”

### Unzip and decompress the file system and add the new executable

The compress file system has a cpio.gz extension (the current release name for the file system is tisdk-rootfs.cpio.gz)

1. Create a new directory (if it does not exist already) /opt/filesys/studentN

sudo mkdir /opt/filesys/studentN

cd /opt/filesys/studentN

1. Copy the current compressed file system to the new directory

sudo cp /var/lib/tftpboot/studentN/tisdk-rootfs.cpio.gz .

1. Unzip the compressed file system

sudo gzip –d tisdk-rootfs.cpio.gz (or the name of the file system that you use)

1. Uncompress the file system from the cpio file. This operation builds the complete file system. Note that the compressed file tisdk-rootfs.cpio is still in the directory

sudo cpio –i –F tisdk-rootfs.cpio

1. Remove tisdk-rootfs.cpio

sudo rm tisdk-rootfs.cpio “

1. Copy the executable that was built in the previous paragraph (example1) into usr/bin directory in the file system. The complete path is /opt/filesys/studentN/usr/bin.

sudo cp example1DirectoryLocation/example1 /opt/filesys/studentN/usr/bin/.

### Compressed and zip the new file system

1. The next step is to compress the file system back into a new file system. This is done by piping all the directories and the files in the file system into the cpio. The resulted compressed file system will be copied to one directory above –

cd /opt/filesys/studentN

sudo chmod -R 777 \* ( This will give read write execution permission to all files and subdirectories)

sudo find . | sudo cpio –H newc –o –O new.cpio

Where new.cpio is the new compressed file system

1. To zip the new file system the user does the following

sudo gzip new.cpio (this will generate a file new.cpio.gz)

sudo cp new.cpio.gz /var/lib/tftpboot/studentN/.

In this point, studentN has two file systems. The user can change the name\_fs in the EVM U-BOOT to new.cpio.gz, or the user can delete the previous tisdk-rootfs.cpio.gz, and then change the name of new.cpio.gz to tisdk-rootfs.cpio.gz, or the user can change the name of new.cpio.gz to any other name (with the .cpio.gz extension) like tisdk-rootfs-example1.cpio.gz, and change the name\_fs in the EVM U-BOOT to the new name.

## Reboot the EVM and run the new program

Reboot the EVM with the new file system. After boot, login as a root. Go to /usr/bin and run example1. Observe the results.

### Task 2(optional): Build U-boot, Boot Monitor and Kernel

Linux distribution enables users to download all source code and build the Linux components. This optional task gives instructions how to do so.

TI uses the Arago distribution on a public server to store all source code. The user can download all the sources and make files to build all images in the release. The Ubuntu server must be connected to the network to facilitate downloading of the source code. If the server is behind a firewall, proxies must be set. These proxies are defined already in the /usr/local/studentStartScript.sh script.

### U-BOOT source code extraction and build instructions

Before getting the source, the user must be in his home directory. Cloning the source code will generate sub-directories.

*cd ~/* back into the home directory /home/studentN

*git clone git://arago-project.org/git/projects/u-boot-keystone.git*  This instruction will clone the repository into your local directory ~/u-boot-keystone

*cd u-boot-keystone*  change directory to where the sources are

*git reset --hard label* Where label is the label that is attached to the current release. To find the head label the user should look at the release notes. For MCSDK 3\_14 the label is ***K2\_UBOOT\_2013-01\_13.09\_01*** so the reset instruction for the current release will be the following

*git reset –hard K2\_UBOOT\_2013-01\_13.09\_01*

Different versions of the U-BOOT can be built, depends how the code is loaded. U-BOOT can be loaded from code composer studio, or from boot methods and the format can either be u-boot.bin, u-boot.img, u-boot-spl.bin or u-boot-spi.gph (The different formats are discussed in the boot loader presentation)

option a: if using CCS to load (u-boot.bin)

*make tci6638\_evm\_config*

*make*

option b: if using the two stage SPI NOR boot

*make tci6638\_evm\_config*

*make spl/u-boot-spl.bin*

*make tci6638\_evm\_config*

*make u-boot.img*

*make tci6638\_evm\_config*

*make u-boot-spi.gph*

### 4. Boot Monitor (skern.bin) source code extraction and build instructions

Before getting the source, the user must be in his home directory. Cloning the source code will generate sub-directories.

*cd ~/* back into the home directory /home/studentN

*git clone git://arago-project.org/git/projects/boot-monitor.git* This instruction will clone the repository into your local directory ~/boot-monitor

*cd boot-monitor* change directory to where the sources are

*git reset --hard label* Where label is the label that is attached to the current release. To find the head label the user should look at the release notes. For MCSDK 3\_14 the label is K2\_BM\_13.08 so the reset instruction for the current release will be the following

*git reset –hard K2\_BM\_13.08*

make clean

make

### 5. Linux Kernel and device tree source code extraction and build instructions

Before getting the source, the user must be in his home directory. Cloning the source code will generate sub-directories.

*cd ~/* back into the home directory /home/studentN

*git clone git://arago-project.org/git/projects/* *linux-keystone.git* This instruction will clone the repository into your local directory ~/boot-monitor

*cd linux-keystone* change directory to where the sources are

*git reset --hard label* Where label is the label that is attached to the current release. To find the head label the user should look at the release notes. For MCSDK 3\_14 the label is *K2\_LINUX\_03.08.04\_13.09\_01* so the reset instruction for the current release will be the following

*git reset –hard K2\_LINUX\_03.08.04\_13.09\_01*

*make keystone2\_defconfig*

*make uImage*

*make keystone-sim.dtb*

*make tci6638-evm.dtb*

The built files:

vmlinux is in /linux-keystone folder:

uImage & \*.dtb are in /linux-keystone/arch/arm/boot folder

# Lab 3 – Boot Using NFS-mounted file system

## Purpose

The purpose of this lab is to demonstrate how to boot the EVM when the file system resides on a different server that is mounted on the EVM, then develop a code on the Linux host and move it to the file system. The executable will be available to the ARM on the EVM. A debug session using gdb will be performed from the serial port terminal.

### Task 1: Build a file system on a Linux host, use the NFS server

The NFS server is installed on the Ubuntu server in the directory /opt/filesys. Each student has a sub-directory where he or she builds the file server, and the Uboot is configured to reach this directory for each student.

1. Next the file system to be mounted should be built on the local Ubuntu machine.
   1. Create a directory where the file system resides; say /opt/filesys/studentN (where N is the student number. **Note, this directory should be created already**)
   2. Copy a tar version of the compressed file system tisdk-rootfs.tar.gz (part of the release in the images directory) into /opt/filesys/studentN
   3. Untar the file system -> “sudo tar zxf tisdk-rootfs.tar.gz “
   4. Delete the original compress file -> “sudo rm tisdk-rootfs.tar.gz “
   5. Add the file system directory to the exports list, open the file /etc/exports and add the following line to it. **Note: just verify that this was already done.**

/opt/filesys \*(rw,subtree\_check,no\_root\_squash,no\_all\_squash,sync)

The file /etc/exports looks like the following:



1. The instructor will start the NFS server

### Task 2: Configure U-BOOT to mount the file server and boot

1. Power cycle the EVM, in the ARM tera-terminal stop the autoboot
2. Change the following environment variable
   1. Change the boot to be from the network :  
      setenv boot net
   2. Add the nfs server ip  
       setenv nfs\_serverip xxx.xxx.xxx.xxx   
      where xxx.xxx.xxx.xxx is the IP address of the Ubuntu server on which the file system resides. For standard workshop it is 192.168.0.100
   3. Define the file system root directory:  
      setenv nfs\_root /opt/filesys/studentN
   4. Configure the arguments for the boot:   
      setenv args\_net 'setenv bootargs ${bootargs} rootfstype=nfs root=/dev/nfs rw nfsroot=${nfs\_serverip}:${nfs\_root},${nfs\_options} ip=dhcp'
   5. Save the new environment variables:  
      saveenv
3. Boot
   1. Note, if the DHCP does not supply an IP address to the EVM, the EVM will use its default IP address. This default IP address is define in the environment -> “ printenv” as ipaddr. If this does not exist the user can configure it -> “setenv ipaddr yyy.yyy.yyy.yyy “

### Task 3: Build a new C program in the file system, and debug it

1. In a local Ubuntu terminal go to /opt/filesys/studentN and look at the file system
2. Follow the example simple code section of Lab 2, copy example1.c into one of the directories of the file system, for example into /opt/filesys/studentN/bin. You may need to modify permissions of the bin directory as such:  
   sudo chmod 777 /opt/filesys/studentN/bin
3. Set the terminal in the bin directory -> “cd /opt/filesys/student/bin “
4. Compile and build the application similar to what you did in Lab2, but add the debug flag (-g) to the command – that is

arm-linux-gnueabi-gcc **–g** -o example1 example1.c

1. Back to the tera-terminal, navigate to /bin -> “cd /bin “
   1. Make sure that example1.c and example1 are both in the bin directory -> “ls –ltr example1\*
   2. Start a debug session -> “ gdb example1 “
   3. Use the list command to see the source, use b to set a break point, use r to run to the break point
   4. Other simple gdb command s to step, n for next (step over), c to run to the next breakpoint, and finish to end
   5. There are many gdb quick guides on the Web. Here is a URL to one of them:

<http://condor.depaul.edu/glancast/373class/docs/gdb.html>

# Lab 4 – Build, run and optimize DSP project using CCS

## Purpose

In lab 2 and 3 ARM program was developed and debugged. The purpose of this Lab is to develop and debug multicore C66 program using CCS IDE. This lab has the following parts:

1. Using CCS, build a simple FIR project that runs on a single core
2. Optimize the code by achieving software pipeline, understand what can prevent the compiler from generating software pipeline code
3. Optimize execution by enabling cache
4. Perform parallel processing of the code and observe multi-cores processing speedup

CCS IDE is used to execute the lab. Before starting, the EVM should be configured to no-boot mode. To do so, Set SW1 of the EVM to Off, Off, Off, On

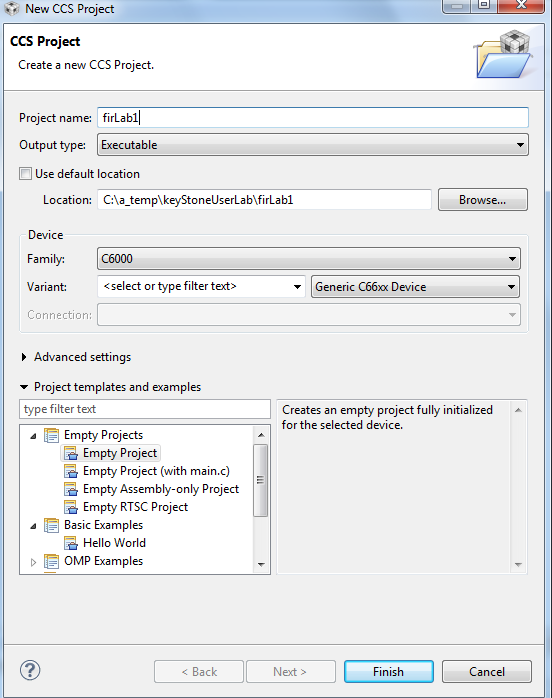
## II. Project Files

The following files are used in this lab:

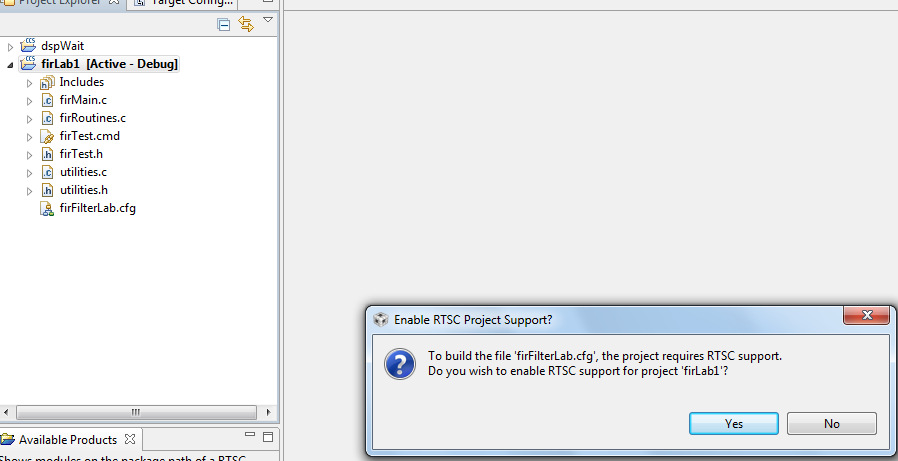
1. firMain.c
2. firRoutines.c.c
3. firTest.cmd
4. firTest.h
5. utilities.c
6. Utilities.h
7. firFilterLab.cfg

### Task 1: Build and Run the Project

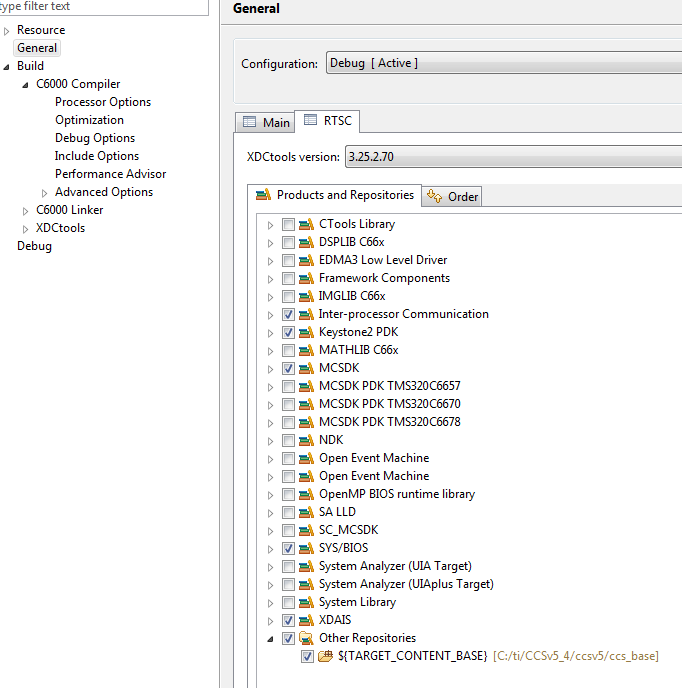
1. ftp into the Ubuntu server and get all the files that are in directory /usr/local/projects/DSP/firLab into a local directory on your Laptop c;\ti\labs\firFilter. If this directory does not exist, create it
2. Open CCS.
3. Create new project through the CCS menu item *File* 🡪 *New* 🡪 *CCS Project*.
4. Enter *firLab1* as a *Project Name*.
5. Click the check box to *Use default location.*
6. Set the *Family to C6000* and *Variant* to *Generic C66xxx Device*
7. Then press *Finish* to create the new project. See the screen shot below. Note, you will use the default location and not the location in the screen shot.



1. Then in the *Project Explorer* view, right-click on the newly-created *firLab1* project, and click on *Add Files…*
2. Browse to ‘C:\ti\labs\firFilter,’ select all the files in this directory, and click *Open*. When prompted how files should be imported into the project, leave it as default of *Copy File.* If you defined the new project with main.cremove the main.c file that may be created.
3. As soon as the file firFilter.cfg imported into the project, CCS will ask you to enable RTSC support, see the screen shot below. Select yes.

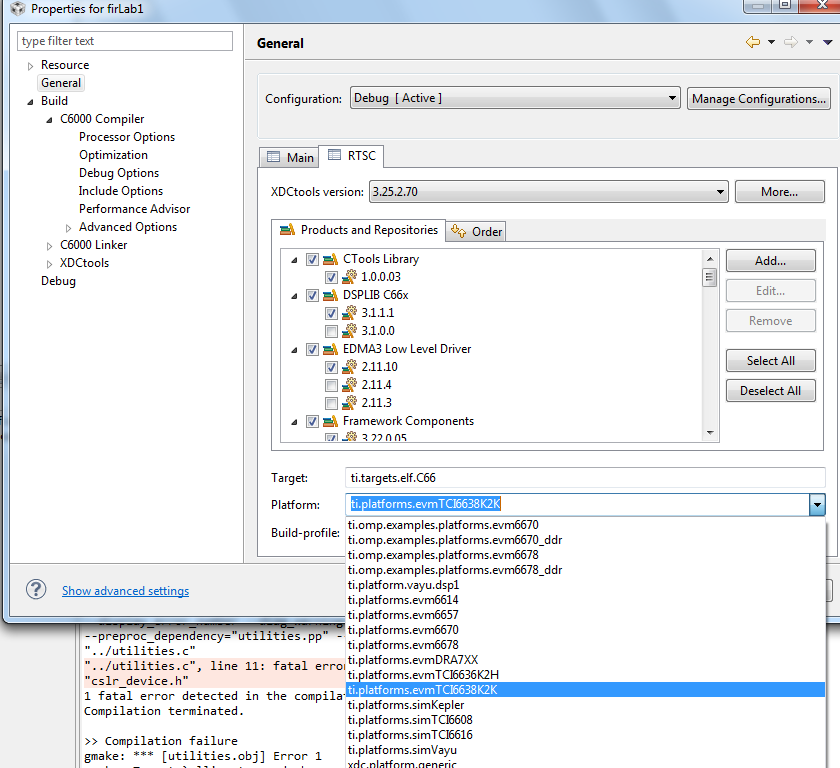


1. Open project properties and select general->RTSC. Look at the RTSC modules that are selected in the screen shot below and make sure that you select ONLY the same RTSC modules (or packages). When a project starts, RTSC attempt to include all the modules in the release, so unselect any module that is not in the screen shot. Note, the location of the TARGET CONTENT BASE should reflect the location of CCS in your system.



1. Click on the platform tab and select ***ti.platform.evmTCI6638K2K*** platform as shown in the next screen shot

Note – RTSC projects require the user to select three type of information. The device family in the CCS create page determines what core is used and thus what version of the compiler should be used (different cores have different intrinsic functions). The platform that is defined here determines the memory configuration of the core. To build the correct RTSC drivers, the device name should be defined. This is done by adding a predefine symbol with the device name. More about it later.



1. If you try to build the project now you will see a long list of errors that look like the following: C:/ti/MCSDK\_3\_14/pdk\_keystone2\_3\_00\_02\_14/packages/ti/csl/csl\_cacheAux.h", line 86: error #20: identifier "CSL\_C66X\_COREPAC\_REG\_BASE\_ADDRESS\_REGS" is undefined. This is because the device that we use is not defined yet, so RTSC does not know how to build the drivers for the device that is used. Next step describes how to configure the device

1. Currently the file cslr\_device.h refers to two devices, K2K and K2H. The EVM uses K2K device. (cslr\_device.h has the following lines:

#if defined(DEVICE\_K2K)

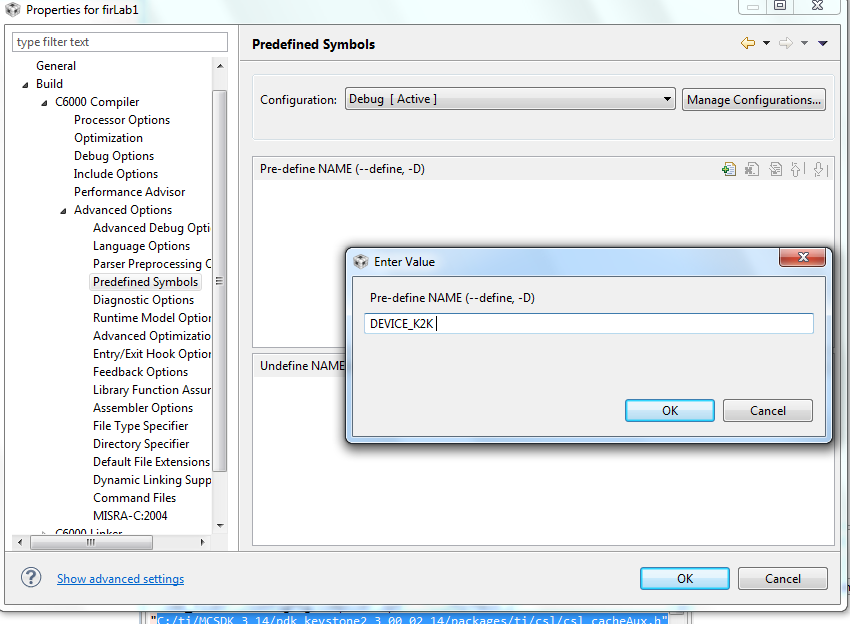
#include <ti/csl/device/k2k/src/cslr\_device.h>

#elif defined(DEVICE\_K2H)

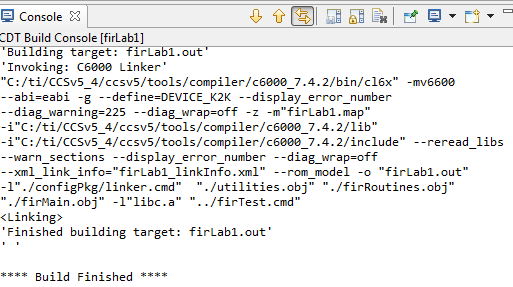
#include <ti/csl/device/k2h/src/cslr\_device.h>

#endif /\* DEVICE\_XXXXX \*/)

1. To configure the device open the project properties, build -> C6000 compiler ->advanced Options -> predefined symbols and enter DEVICE\_K2K as the next screen shot shows. Click OK and then OK to the properties page.



1. Right click on the project name and select rebuild. If the build goes correctly you will see the following in the consol. Note, look at the debug directory to ensure that the file firLab1.out is there. Ignore any warning.



1. Examine the code in ‘firMain.c’. There are 5 cases but only case 1 is not commented out. DSP 0 generates input data (inputData) and a set of filter coefficients (filterCoef) , and then, depends on the case, a set of fir filters is applied to the data and the results are written to the out file (outputFilter). A set of timer registers (TSCL and TSCH)measures the execution time of the fir filter. The standard printf function prints the results on the console.

### Task 2: Define the target

In this lab we run the DSP code from no-boot mode. The non-boot mode requires setting SW1 of the EVM to Off, Off, Off, ON. Since no-boot mode is chosen, the device configuration (DDR configuration, PLL configuration and so on) must be done in a gel file.

#### Create a new target in CCS

1. Create a new target configuration:
   1. Select the CCS menu option *View 🡪 Target Configurations*.
   2. Select *User Defined*.
   3. Right-click and select *New Target Configuration*.
2. Enter the name of the new target configuration in the *File Name:* text box.
   1. Set the File name based on the EVM model, *<model>.ccxml*  
      For example, ‘TCI6638.ccxml’
   2. Leave the *Location* the default value:  
      “C:\Documents and Settings\student\ti\CCSTargetConfigurations”
   3. Click the *Finish* button. The .ccxml file will now open in a GUI-based view with the *Basic* tab active.
3. First step to define the new target configuration is to select the connection type in the *Basic* Tab.
4. The *Connection* drop-down menu identifies the emulator type, as shown in the table above. For example, ‘Texas Instruments XDS2xx USB Emulator”
   1. *Board or Device* identifies the TI processor device, set it to 6638 and select TCI6638
   2. Under *Save Configuration*, click the *Save* button.
5. Second step is to configure setup in *Advance* Tab
   1. Click the *Advanced* tab at the bottom of the screen.
   2. Select Core 0 on the target device:
      * *TCI6638\_0* 🡪 *IcePick\_C\_0* 🡪 *Subpath\_1* 🡪 *C66xx\_0*
   3. You will now see a sub-window called *Cpu Properties* that allows you to choose an *initialization script*.
   4. Locate the appropriate GEL file, then click *Open*:
      * Select: C:\ti\CCSv5\_4\ccsv5\ccs\_base\emulation\boards\evmtci6638k2k\gel\evmtci6638k2k.gel
      * Repeat the process for all C66, that is *C66xx\_1, C66xx\_2, … C66xx\_7*

Click the *Save* button

### Task 3: Connect to the EVM

1. Click the *Open Perspective* (available right top corner of the *CCS*).
2. Switch to the Debug Perspective by selecting the CCS menu option *Window* 🡪 *Open Perspective* 🡪 *CCS Debug*.
3. Select the CCS menu option *View* 🡪 *Target Configurations*. Select the target configuration you created
4. Launch the target configuration as follows:
   1. Select the target configuration .ccxml file.
   2. Right click and select *Launch Selected Configuration*.
5. This will bring up the *Debug* window. (this may take some times, but you will see all the device cores)
   1. Select all C66 cores (select + Ctrl)
   2. Right click and choose group cores
   3. Select the group, right click and select connect Target

### Task 4: Load and Run CASE 1

1. Select the core group and load the .out file created earlier in the lab.
   1. Select the CCS menu option *Run* 🡪 *Load* 🡪 *Load Program*
   2. Click *Browse project…*
   3. Select *firLab1.out* by unwrapping the *firLab1->Debug* and click *OK.*
   4. Click *OK* to load the application to the target (all Cores).
2. Run the application by selecting the CCS menu option *Run* 🡪 *Resume*.
3. A successful run should produce a console output as shown below. Record the cycles time:

[C66xx\_0] start generating input data

finish generating input data

case 1 -> time consumed By core -> 0 610749952.000000

**QUESTION:**

Look at the function CACHE\_disableCaching ((Uint8) 144) – It disable cache-ability for memory region. What memory region is it?

1. Look at C66 core User Guide (<http://www.ti.com/lit/ug/sprugw0c/sprugw0c.pdf>) at table 4-20 .

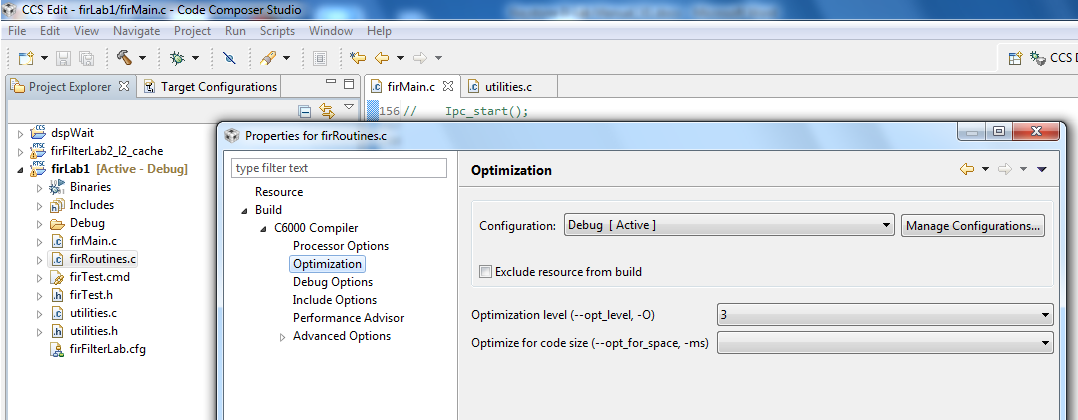
Look at the User Guide, the code and the map file and fill the table based

|  |  |  |
| --- | --- | --- |
| Command | Memory region | Variables in this region |
| CACHE\_disableCaching ((Uint8) 128) |  |  |
| CACHE\_disableCaching ((Uint8) 136) |  |  |
| CACHE\_disableCaching ((Uint8) 144 |  |  |
| CACHE\_disableCaching ((Uint8) 145) |  |  |
| CACHE\_disableCaching ((Uint8) 146) |  |  |
| CACHE\_disableCaching ((Uint8) 147) |  |  |

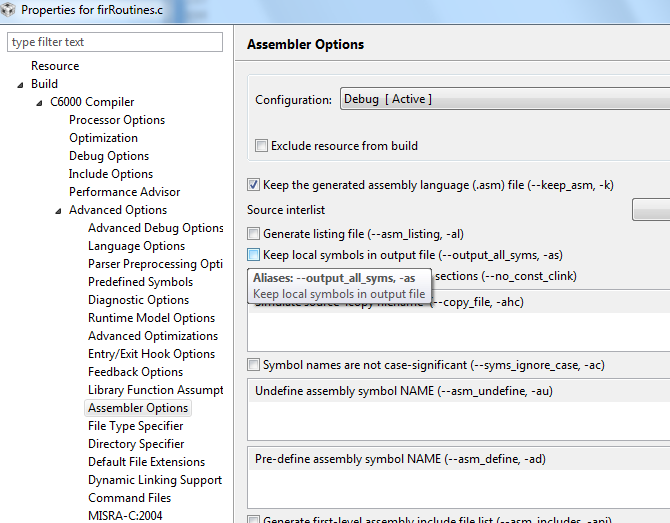
### Task 5: Use Optimization and disable symbol debug for the fir filter routine

As the project is still in development/debug state, there is no optimization and full debug support. The next step is to optimize the fir filter and disable the debug information. However, we would like to leave the other parts of the project without optimization and with full debug support. The properties for the file firRoutines.c will be changed. No other file will be effected.

1. In the project explorer, select the file firRoutines.c and right click. Open the properties dialogue window – see screen shot below
2. Select build->optimization. In the dialogue window set optimization to 3
3. From the debug options dialogue select “suppress all symbolic debug generation” from the pull down menu



1. From the build ->C6000 compiler -> Advanced Debug select Assembly options and check “Keep the generated assembly language (.asm) file as seen in the screen shot below.



1. Click OK and rebuild the project. Load and run
2. A successful run should produce a console output as shown below. Record the cycles time:

start generating input data

finish generating input data

case 1 -> time consumed By core -> 0 500579008.000000

**QUESTION:**

Is the code really optimized? Only 15% improvements

1. Look at the assembly file firRoutines.asm in the debug directory and search for the function firRealFilter. Look for the loop and see if the compiler could get software pipeline

What is the reason that the loop does not qualify for software pipeline?

### Task 6: Optimize Software Pipeline

The reason why fir filter loop is not qualified for software pipeline is because it calls myMultiply. Next step is to inline this function. myMultiply is an artificial function (no one will develop this function in real code) so it is easy to “inline” it. Look at the definition of myMultiply in the utilities.c file and inline it.

1. Change the function firRealFilter by inline myMultiply function
2. Save and build the project. Load and run
3. A successful run should produce a cosole output as shown below. Record the cycle time:

start generating input data

finish generating input data

case 1 -> time consumed By core -> 0 273086080.000000

1. Next step is telling the compiler what is the minimum times that each loop will be executed. The filter size in this program is 8. Assume that the filter size will always be more than 4 and divided by 4, so adding a pragma( **#pragma** MUST\_ITERATE(4,,4); ) will tell the compiler that the inner loop must be performed at least 4 times and the number of iterations is divided by 4
2. The outer loop presents the size of the output vector. The number of elements is 16K, but eventually we would like to run it on all 8 cores, so each core will have about 2K element. It is enough if we tell the compiler that the number of elements is more than, say 64. However, if you look carefully, you will notice that the number of output results is 16K – filter size + 1, so this is an odd number. You can tell the compiler that the number of elements is more than 64. In that case use something like (**#pragma** MUST\_ITERATE(64,,1); ) or, if you agree to ignore the last fake result, you can tell the compiler (**#pragma** MUST\_ITERATE(4,,2); ))
3. Add the two pragma directives before the two loops (internal and external) in the function save and build.
4. If the external loop is **pragma** MUST\_ITERATE(64,,1)

start generating input data

finish generating input data

case 1 -> time consumed By core -> 0 221407008.000000

1. If the external loop is **pragma** MUST\_ITERATE(64,,2);

start generating input data

finish generating input data

case 1 -> time consumed By core -> 0 221306848.000000

**QUESTION: To summarize the code optimization section, fill the following table**

|  |  |  |
| --- | --- | --- |
| Optimization Technique | Cycles | Improvements compare with previous line |
| No Optimization |  |  |
| Compiler optimization 3, no symbolic debug |  |  |
| Software Pipeline |  |  |
| Adding pragma must iterate |  |  |
|  |  |  |

### Task 7: Enable the cache

Enabling the cache is done in CASE 2. Un-comment the line #define CASE\_2 above the main() in firMain.c

**QUESTION:**

What instruction(s) enable the cache

1. The function CACHE\_enableCaching ((Uint8) 128) ; was discussed in task 4. The function CACHE\_setL2Size ((CACHE\_L2Size) 4); is part of the file csl\_cachAux.h in the \MCSDK\_3\_14\pdk\_keystone2\_3\_00\_02\_14\packages\ti\csl directory. Note, version number and location of MCSDK may be different for your setting.
2. Un-comment the line #define CASE\_2 in firMain.c
3. Save, build, load and run. The results will be looked like the following:

start generating input data

finish generating input data

case 1 -> time consumed By core -> 0 221223008.000000

case 2 -> time consumed By core -> 0 7491409.000000

**QUESTION: Complete the table**

|  |  |  |
| --- | --- | --- |
| Optimization Technique | Cycles | Improvements compare with previous line |
| No Optimization |  |  |
| Compiler optimization 3, no symbolic debug |  |  |
| Software Pipeline |  |  |
| Adding pragma must iterate |  |  |
| Enabling cache |  |  |

**QUESTION: What are the most important steps to optimize code running on a single core?**

### Task 8: Running in parallel on multiple cores

Multiple cores are enables in CASE 3 (2 cores), CASE 4 (4 cores) and CASE 5 (8 cores). Un-comment the lines #define CASE\_3 #define CASE\_4 and #define CASE\_5 above the main() in firMain.c

1. Un-comment the line #define CASE\_3 #define CASE\_4 #define CASE\_5 in firMain.c
2. Save, build, load and run. The results will be looked like the following:

finish generating input data

case 1 -> time consumed By core -> 0 288423616.000000

case 2 -> time consumed By core -> 0 7493824.000000

case 3 -> time consumed By core -> 0 3680093.000000

[C66xx\_1] case 3 -> time consumed By core -> 1 3678251.000000

[C66xx\_0] case 4 -> time consumed By core -> 0 1839643.000000

[C66xx\_1] case 4 -> time consumed By core -> 1 1838608.000000

[C66xx\_2] case 4 -> time consumed By core -> 2 1839438.000000

[C66xx\_3] case 4 -> time consumed By core -> 3 1836440.000000

[C66xx\_0] case 5 -> time consumed By core -> 0 918711.000000

[C66xx\_1] case 5 -> time consumed By core -> 1 921884.000000

[C66xx\_2] case 5 -> time consumed By core -> 2 921973.000000

[C66xx\_3] case 5 -> time consumed By core -> 3 920785.000000

[C66xx\_6] case 5 -> time consumed By core -> 6 922374.000000

[C66xx\_4] case 5 -> time consumed By core -> 4 923078.000000

[C66xx\_5] case 5 -> time consumed By core -> 5 921646.000000

[C66xx\_7] case 5 -> time consumed By core -> 7 920075.000000

For each case, the total time that is consumed to perform the FIR filter is the maximum time of all the cores.

**QUESTION: Complete the table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case** | **Cycles per core** | **Execution cycles (this is the cycles of the core with the highest cycles count)** | **Accumulate execution time for all the cores** | **Penalty of the accumulation execution time compare to single core (CASE 2)** |
| Case 2 – single core |  |  |  |  |
| Case 3 – 2 cores |  |  |  |  |
|  |
| Case 4 – 4 cores |  |  |  |  |
|  |
|  |
|  |
| Case 5 – 8 cores |  |  |  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**QUESTIONS:**

1. What is the purpose of the function waitBarrier(barrier\_1, coreNum, jointNumber) , what would happen if the function is commented out
2. Try to comment out the function (3 places) and see what happen.
3. What is the purpose of the function waitAboutNSeconds(10) inside the function waitBarrie, what would happen if the function is commented out. Do you understand why?
4. Try to comment out the function and see what happen. Think about timing between cores.
5. Can you think about other (better) methods to synchronize the execution of all the cores?
6. Semaphores?, QMSS queues based solution?, openMP? .

# Lab 5 – Using MPM server to load and run DSP code

In this lab you build a DSP project similar to Lab 4. For screen shots refer to Lab 4. Before you start you should change the boot mode of the EVM from no boot of Lab 4, CCS IDE is used to execute the lab. Before starting, the EVM should be configured to no-boot mode. To do so, Set SW1 of the EVM back to Off, Off, On, Off. Read Lab3 instructions to ensure that the EVM boots using NFS-mounted file system.

## Purpose

Building a DSP code that is managed by the ARM. The ARM will reset C66 core 0, load it with executable, run it and retrieve the results

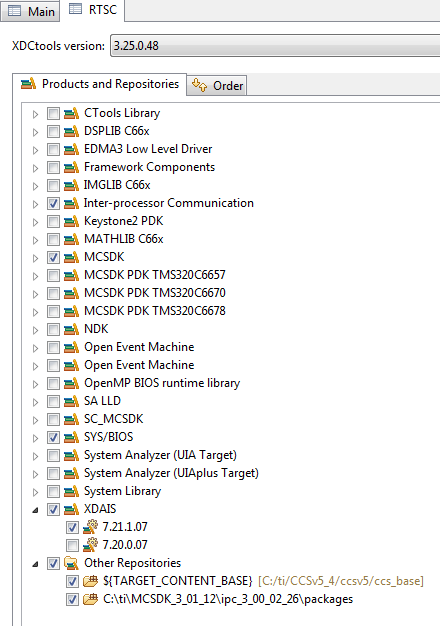
## II. Project Files

The following files are used in this lab:

1. Main.c
2. mpmsrv\_keystone2\_example1.cfg

### Task 1: Build and Run the Project

1. ftp into the Ubuntu server and get all the files that are in directory /usr/local/projects/DSP/mpm into a local directory on your Laptop c;\ti\labs\mpm\_example. If this directory does not exist, create it
2. Open CCS.
3. Create new project through the CCS menu item *File* 🡪 *New* 🡪 *CCS Project*.
4. Enter *mpm\_example* as a *Project Name*.
5. Click the check box to *Use default location.*
6. Set the *Family to C6000* and *Variant* to *Generic C66xxx Device*
7. Then press *Finish* to create the new project
8. Then in the *Project Explorer* view, right-click on the newly-created *mpm\_example* project, and click on *Add Files…*
9. Browse to ‘C:\ti\labs\mpm\_example,’ select all the files in this directory, and click *Open*. When prompted how files should be imported into the project, leave it as default of *Copy File.* If you defined the new project with main.cremove the main.c file that may be created.
10. As soon as the file mpmsrv\_keystone2\_example1.cfg imported into the project, CCS will ask you to enable RTSC support, Select yes.
11. Open project properties and select general->RTSC. Look at the RTSC modules that are selected in the screen shot below and make sure that you select ONLY the same RTSC modules (or packages). When a project starts, RTSC attempt to include all the modules in the release, so unselect any module that is not in the screen shot. Note, the location of the TARGET CONTENT BASE should reflect the location of CCS in your system.



1. Click on the platform tab and select ***ti.platform.evmTCI6638K2K*** platform

Note – RTSC projects require the user to select three type of information. The device family in the CCS create page determines what core is used and thus what version of the compiler should be used (different cores have different intrinsic functions). The platform that is defined here determines the memory configuration of the core. To build the correct RTSC drivers, the device name should be defined. This is done by adding a predefine symbol with the device name. More about it later.

1. Right click on the project name and select rebuild. If the build goes correctly you will see the following in the consol. Note, look at the debug directory to ensure that the file mpm\_example.out is there. Ignore any warning.

'Building target: MPM\_example.out'

'Invoking: C6000 Linker'

"C:/ti/CCSv5\_4/ccsv5/tools/compiler/c6000\_7.4.2/bin/cl6x" -mv6600 --abi=eabi -g --display\_error\_number --diag\_warning=225 --diag\_wrap=off -z -m"MPM\_example.map" -i"C:/ti/CCSv5\_4/ccsv5/tools/compiler/c6000\_7.4.2/lib" -i"C:/ti/CCSv5\_4/ccsv5/tools/compiler/c6000\_7.4.2/include" --reread\_libs --warn\_sections --display\_error\_number --diag\_wrap=off --xml\_link\_info="MPM\_example\_linkInfo.xml" --rom\_model -o "MPM\_example.out" -l"./configPkg/linker.cmd" "./main.obj" -l"libc.a"

<Linking>

'Finished building target: MPM\_example.out'

' '

\*\*\*\* Build Finished \*\*\*\*

### Linux and DSP simple memory management

The above project uses private L2 memory for program and data. What should be done for more complex DSP projects that use DDR? How does the system manage the DDR resources between the DSP and the ARM?

The Linux uses part of the DDR, so if a DSP program uses some of the DDR, it must tell the Linux. This is done in the U-BOOT environment.

To do it correctly, the user must follow the following steps:

1. Stop autoboot and look at the messages from the U-BOOT. It looks like the following:

I2C: ready

Detected SO-DIMM []

**DRAM: 1 GiB**

NAND: 512 MiB

Net: TCI6638\_EMAC, TCI6638\_EMAC1

Hit any key to stop autoboot: 0

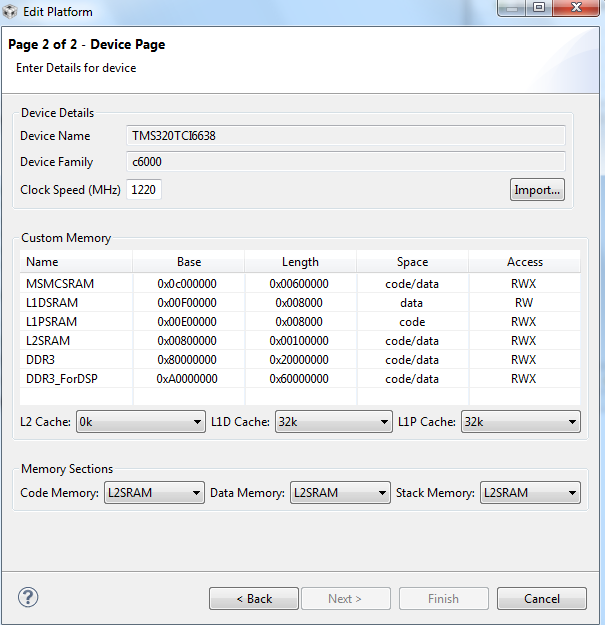
The size of the DRAM is 1 GiB in this case. It can be 2G or 8 G depends on the EVM revision

1. Now determine how much DRAM the DSP needs. Obviously it must be less than the total DRAM in the EVM. Assume that the LINUX uses 512M and the DSP will use 512M. The user must tell the U-BOOT that 512M is reserved for the DSP.
2. After stopping the autoboot, configure the memory that is assigned to the DSP (if it is not configured already)

setenv mem\_reserve 512M (For EVM with 2G DRAM it can be 1536M or even larger if the DRAM is more than 2G)

saveenv and boot

1. The memory that is reserved for the DSP is located at the end of the available memory. For the 1G DRAM case, available memory is between 0x80000000 and 0xbfffffff, so the 512M reserved for the DSP start at address 0xa0000000 to address 0xbfffffff
2. Building the DSP code – Next we need to build the DSP code and ensure that it uses only the assigned DDR. One way to do it when using RTSC is to re-define the platform. The release platform defines DDR starting at 0x80000000 and with size 0x80000000. One way is to modify this platform
3. To modify the platform go to the debug prospective, tools -> RTSC Tools ->platform -> edit/view. Make sure that the repository is the location of XDCTools/packages (in MCSDK\_3\_14 setting it is in *“MCSDK\_3\_14\xdctools\_3\_25\_02\_70\packages”* and click on package Name. Chose the package that corresponds to the EVM, (the current EVM is *ti.platforms.evmtci6638K2K*) CPU core is CPU and click NEXT. Change the platform as illustrated in the following screen shot



1. In a link command file XXX.cmd make sure that all DDR sections are assigned to the memory DDR3\_ForDSP (in the case above). Verify the correct address in the map file after rebuilding the project
2. As was mention before, the DSP in this Lab does not use the DDR, so for this lab, the above procedure is not necessary (But can be used as reference for more complex projects)

### Task 2: Using MPM to load, run and observe results

In this part, we assume that the EVM is boot in net mode, that is, the file system is on the server and it is mounted to the EVM as used in Lab3

1. ftp the out file into the server to /opt/filesys/studentN/bin where N is the student number
2. Reboot the EVM using NFS
3. From the terminal login as root, cd to /bin
4. Use MPM to reset, load and run core 0 with MPM\_example.out by using the following MPM commands:
   1. mpmcl reset dsp0
   2. mpmcl load dsp0 MPM\_example.out
   3. mpmcl run dsp0
5. After the end of run look at the trace buffer printing by using the following command

cat /debug/remoteproc/remoteproc0/trace0

1. Change the main.c file as you wish, build it again, ftp to the file system (step 7) load the code to a different dsp (use N here) and run it:
   1. mpmcl reset dspN
   2. mpmcl load dspN MPM\_example.out
   3. mpmcl run dspN
   4. After the end of run look at the trace buffer printing by using the following command

cat /debug/remoteproc/remoteprocN/trace0

# Lab 6 – Using CCS (on Linux) to build and run ARM code

## Purpose

The goal of this lab is to demonstrate some basic operations using CCS on ARM code as preparation for EVM debug from CCS.

## Prerequisites

CCS must be installed on the Linux machine

## Task 1: CCS - Start a new cross compiler project and build it

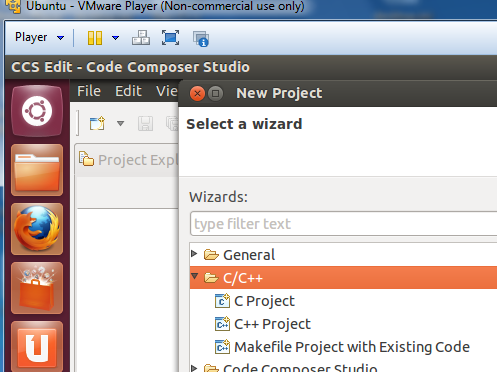
1. To start CCS go to CCS eclipse directory (in my setting it is in /home/a0270985/ti/CCS\_5\_4/ccsv5/eclipse ) and do

sudo ./ccstudio

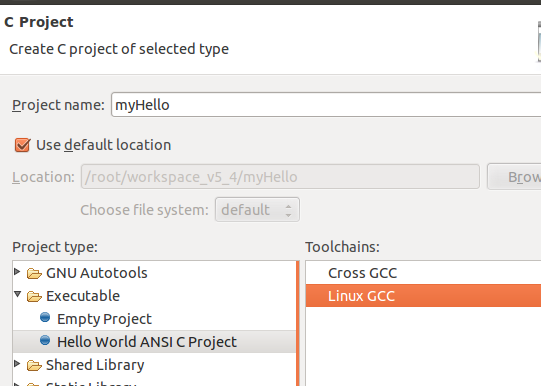
1. After CCS start, and in the edit prospective start a new (not CCS) project:

File -> New -> Project (Not CCS Project)

C/C++ and then C project



From the C project choose Hello World ANCI C project, Toolchains Linux GCC and give the project a name (myHello) and finish.



Before we continue, to make the Lab a little more interesting, we can change the source code using CCS editor. The following is my version of myHello.

**int** **main**(**void**)

{

**int** i, j, k, l ;

**printf**("printf command start the program \n") ;

**puts** (" puts command start the program \n") ;

l = 12 ;

**for** (i=0; i<1000; i++)

{

j = l + i ;

k = 2 \* j ;

l = k + j ;

**if** (l > 10000)

{

**printf**(" inside loop, l > 10000 i= %d k = %d j = %d l = %d \n", i,k,j,l) ;

l = l - 10000 ;

**printf**(" after the normalization i= %d k = %d j = %d l = %d \n", i,k,j,l) ;

}

}

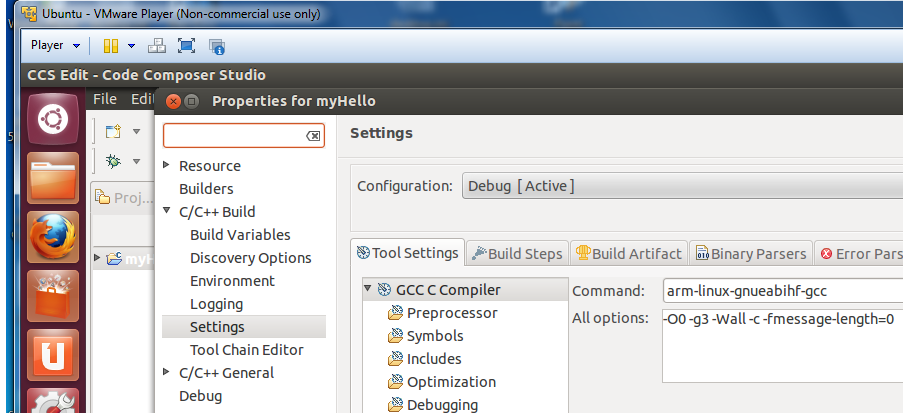
**printf**(" END ->>> i= %d k = %d j = %d l = %d \n", i,k,j,l) ;

**puts**("!!!Hello World!!!"); /\* prints !!!Hello World!!! \*/

**return** 1;

}

Next configure the project properties. Right click on the project and start properties. Open C/C++ build and open setting. In the dialogue window select GCC C compiler and enter arm-linux-gnueabihf-gcc. The same tool will be set for the linker as well.



For assembly, choose the arm-linux-gnueabihf-as assembler.

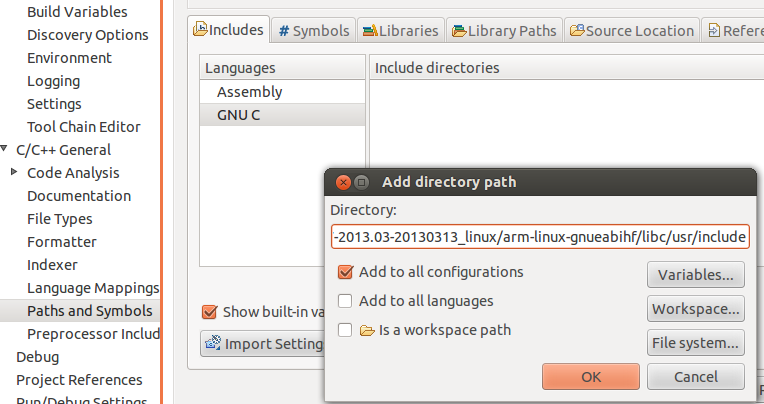
Next the include path to the tools should be configured. From project properties select

C/C++ General->Path and Symbols

Note: Depending on project requirements, more paths may be needed  
  
 From ***Includes*** tab Add GNU C Includes. The easier way is by going to ADD and then file system and browse:  
 DirectoryWhereTheToolsAre/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux/arm-linux-gnueabihf/libc/usr/include  
 DirectoryWhereTheToolsAre /gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux/arm-linux-gnueabihf/libc/usr/include/arm-linux-gnueabihf  
 - ***Add to all configurations***

From ***Library Paths*** tab Add Library Paths

DirectoryWhereTheToolsAre /gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux/arm-linux-gnueabihf/libc/usr/lib/arm-linux-gnueabihf  
 - ***Add to all configurations***



Last the path to the tools should be configured. From project properties select

1. Right-click Project, select ***Properties***
2. Select ***C/C++ Build***🡪***Environment***
3. Click Add…

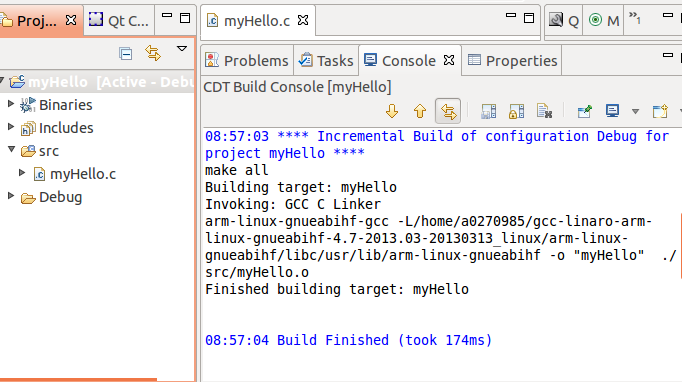
Enter Name: **PATH**, ***Add to all configurations***

Press ***OK***  (ccs will automatically populate with internal path)

1. Select **PATH**, click ***Edit…***
2. Prefix string with path to toolchain binaries  
    (e.g. /opt/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux/bin)
3. Path must be delimited with a colon ‘:’

Do clean build and build

The consol should look like the following:



## Task 2: GDB from the tera-terminal

There are multiple ways to use gdb to run and debug myHello. It can be done outside of CCS or within CCS.

### Move the Debug directory outside of CCS

First allocate where the program is located in the linux host system. From property -> resources you will see something like:

/root/workspace\_v5\_4/myHello

Where workspace\_v5\_4 is the default workspace for CCS.

An alternative method is to search for the myHello project:

cd / takes us to the start of file system

sudo find . –name myHello –print (look for all files with the name myHello in them)

Go to the myHello directory and change the permissions on the Debug sub-directory

Sudo chmod 777 Debug

move the Debug directory to the EVM file system (if the default setting was used, it is in /opt/filesys)

cd /opt/filesys

cd bin

sudo cp –R /root/workspace\_v5\_4/myHello/Debug .

From the tera-terminal go to the bin subdirectory and look at the Debug subdirectory. The src sun-directory should have the source code.

Start the gdb from the tera-terminal

gdb myHello

display the source code

list

Put a breakpoint after the printf

b 17

run the code and observe the printing on the terminal

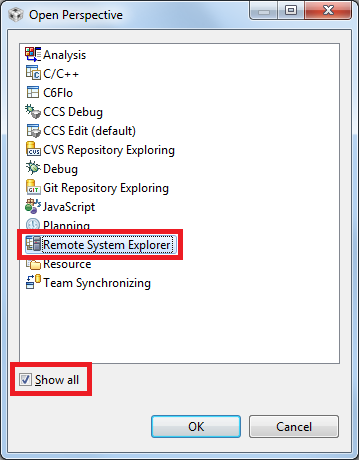
r

c

You can play with other gdb instructions

## Task 3: Connect a SSH terminal into the target

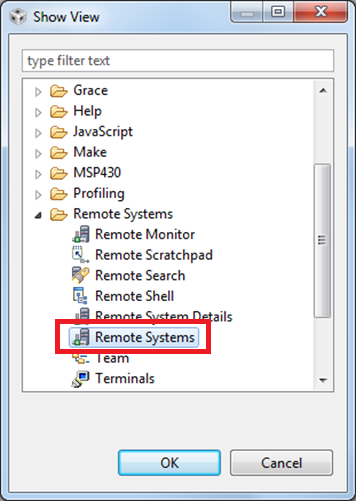
1. After reboot and login the target from the tera-terminal, do ifconfig eth0 and see what the IP address of the target is. Note, the IP address is assigned by the DHCP server.
2. Start CCS; Select ***Window🡪Open Perspective🡪Other***  
   Open Perspective: Check ***Show All***, Select ***Remote System Explorer***  
   Allow enablement of “Remote System Explorer Tools”



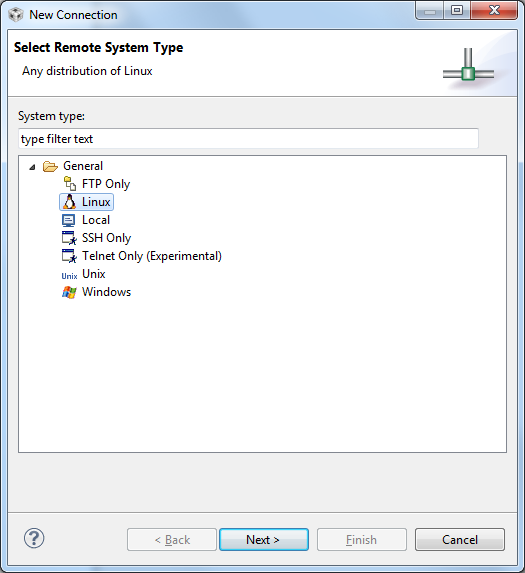
1. Select ***CCS Edit*** perspective



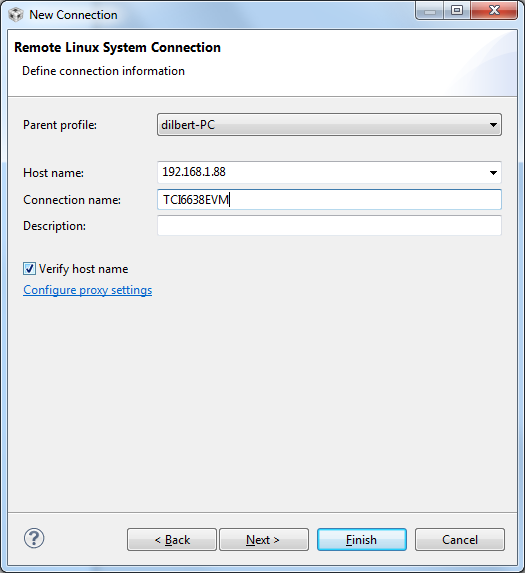
1. Select ***Window🡪Show View🡪Other***  
   Expand Remote Systems, Select ***Remote Systems***  
   Optionally drag Remote Systems to different location  
   (next to Project Explorer tab is a nice spot…)



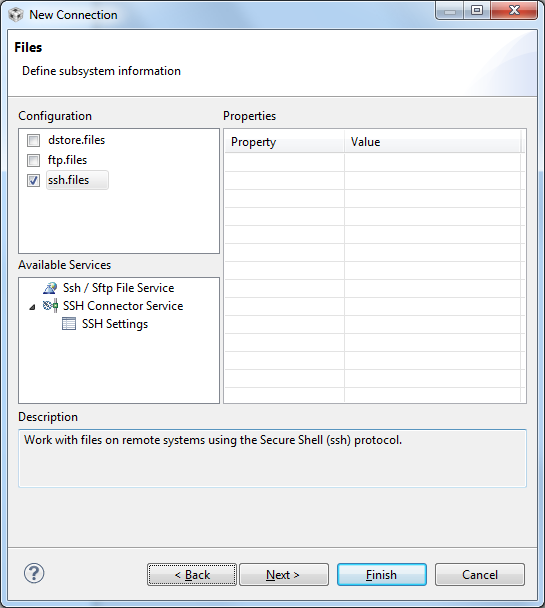
1. From Remote Systems window, Right-Click🡪***New🡪Connection…*** in the window, select linux and click next



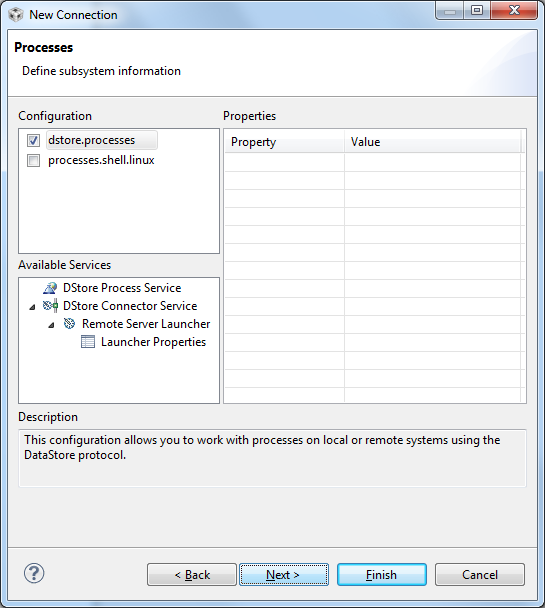
1. Configure the remote linux connection, enter the IP address as the host name, give connection name and profile name:



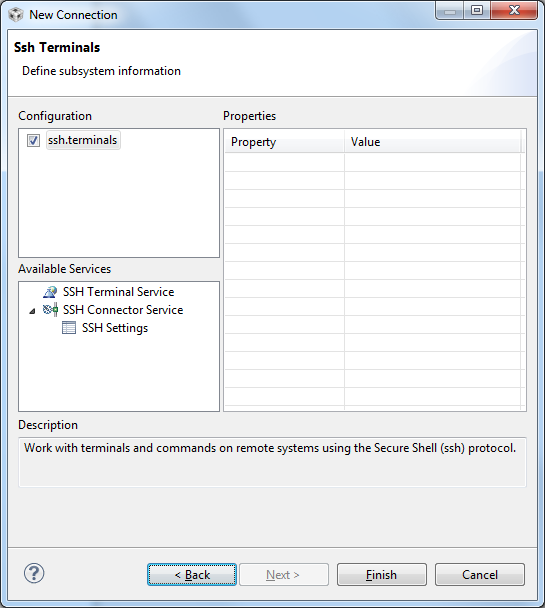
1. Files, select ssh.files



1. next
2. Select processes.shell.linux

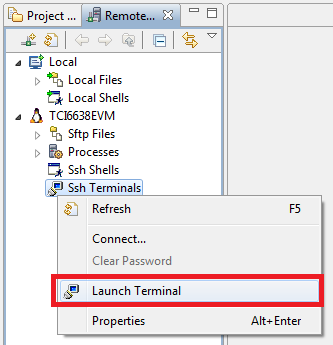


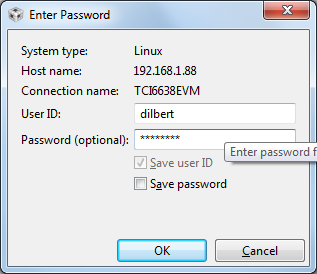
1. Next
2. Select ssh.shells
3. Next

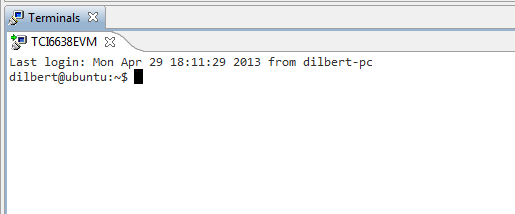


1. Select ssh.terminals
2. Finish
3. Opening a terminal:  
    Right-Click, Select ***Launch Terminal***  
    Enter password (for root user name leave the password blank)

After warnings a terminal will be opened into the target ARM



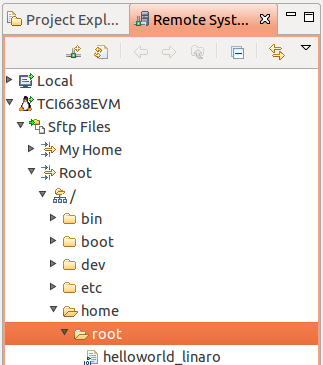
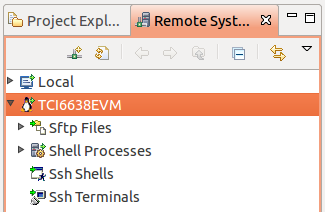
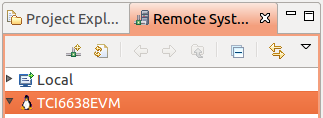
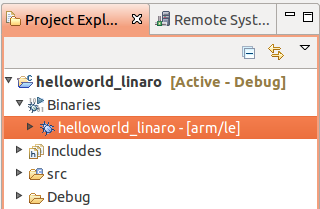




Note – the green arrow indicates that the connection is good. If there is not green arrow, and a proxy server is involved, there may be an additional step required to configure the proxy. Instructions how to configure the proxy are at the end of the Lab. (Task 5)

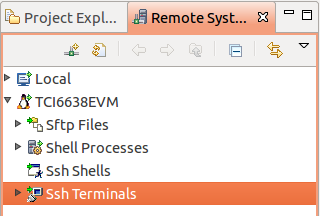
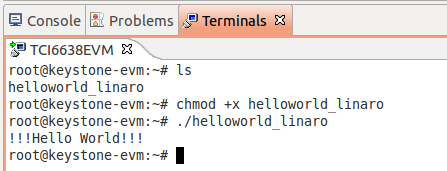
1. Copy and paste binaries onto the target

Select binary file and the source file from project  
Press ***Ctrl-C*** (or right-click, Copy)  
 Click on ***Remote Systems*** tab  
Navigate through the ftp Files tree to desired path  
Press ***Ctrl-V*** (or right-click, Paste)



1. Executing binary on target from CCS RSE Terminal window

1. From the ***Remote Systems*** tab, right-click ***Ssh Terminals***  
 Select ***Launch Terminal***  
2. For newly transferred files, give them executable  
 permissions:  
 chmod 777 ./myHello  
  
3. Execute application  
 ./myHello



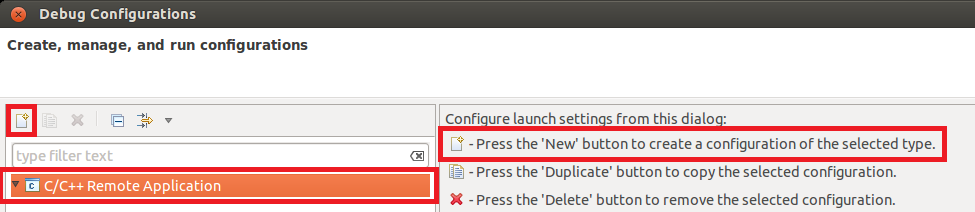
## Task 4: Optional – Use gdb to debug the application

In the edit prospective, select the project myHello

Select ***Run***🡪***Debug configurations*** from file menu

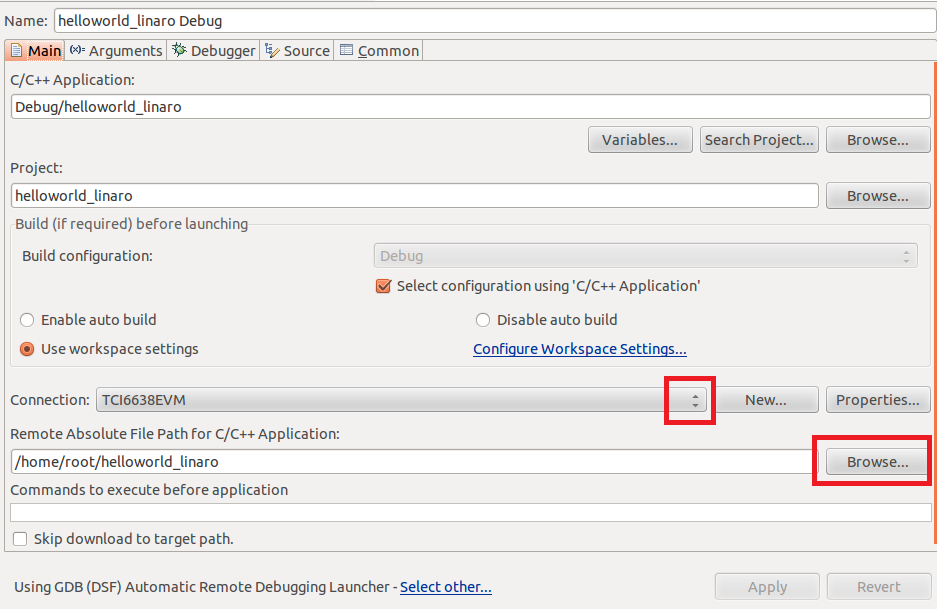
Select ***C/C++ Remote Application***

Click ***New*** button



Main tab:

* + Connection: select EVM [this is configured in WS1]
  + Specify Remote Absolute File Path for C/C++ Application (for example /bin/myHello if you pasted myHello in the bin directory)



. ***Debugger*** tab:

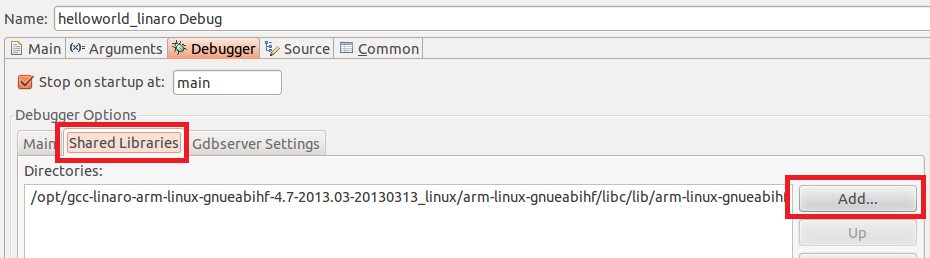
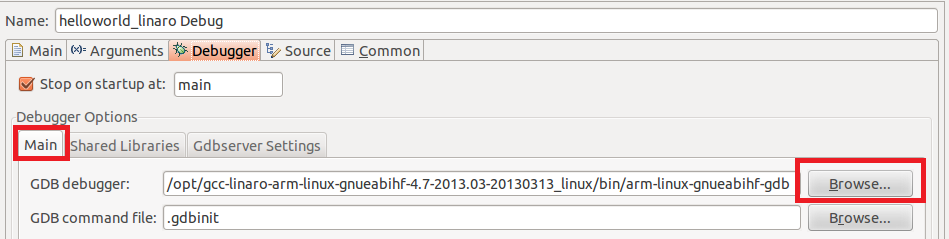
***Main*** sub-tab:

* + GDB debugger🡪***Browse…***:  
     opt/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux/bin/arm-linux-gnueabihf-gdb

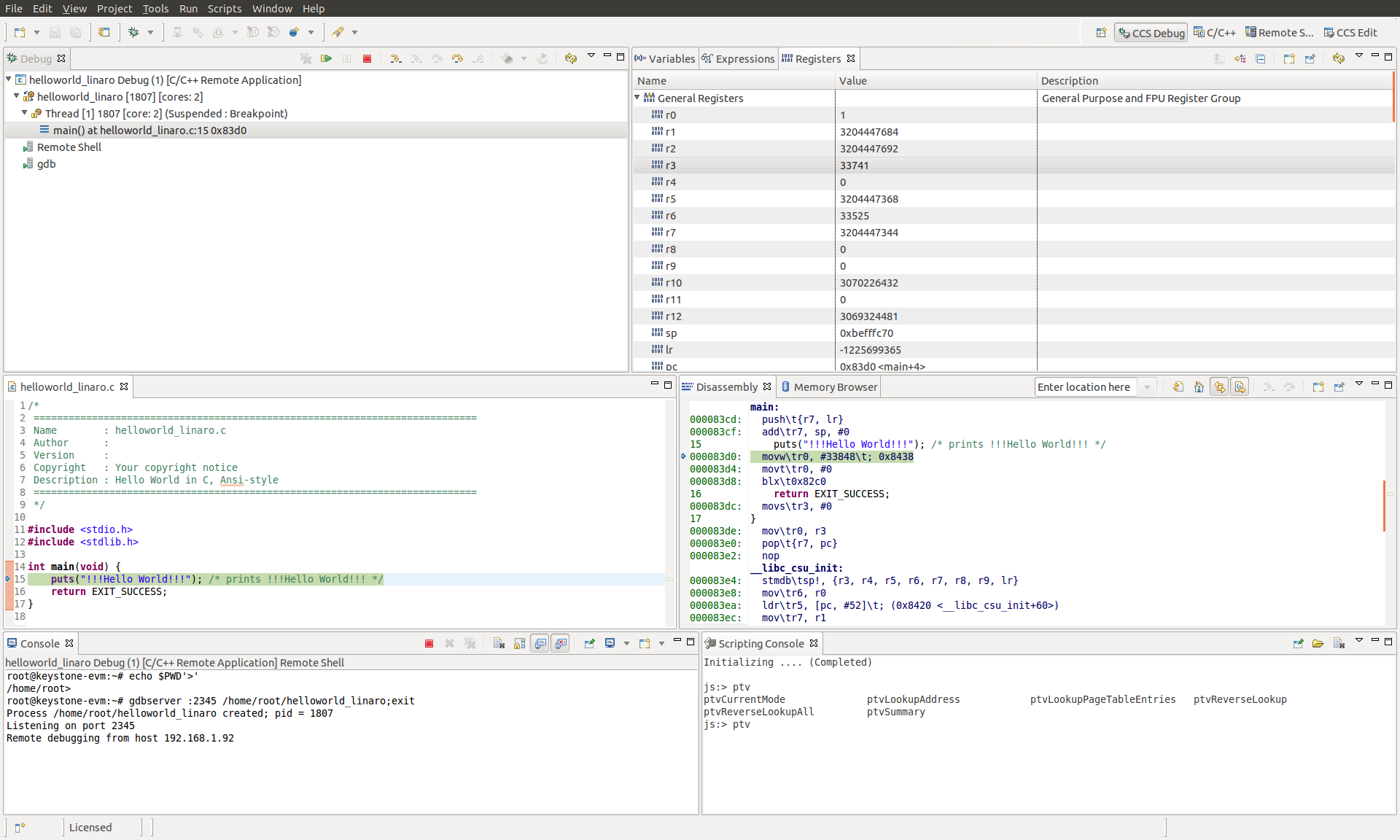
***Shared Libraries*** sub-tab:

* + ***Add…***  
    opt/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux/arm-linux-gnueabihf/libc/lib/arm-linux-gnueabihf

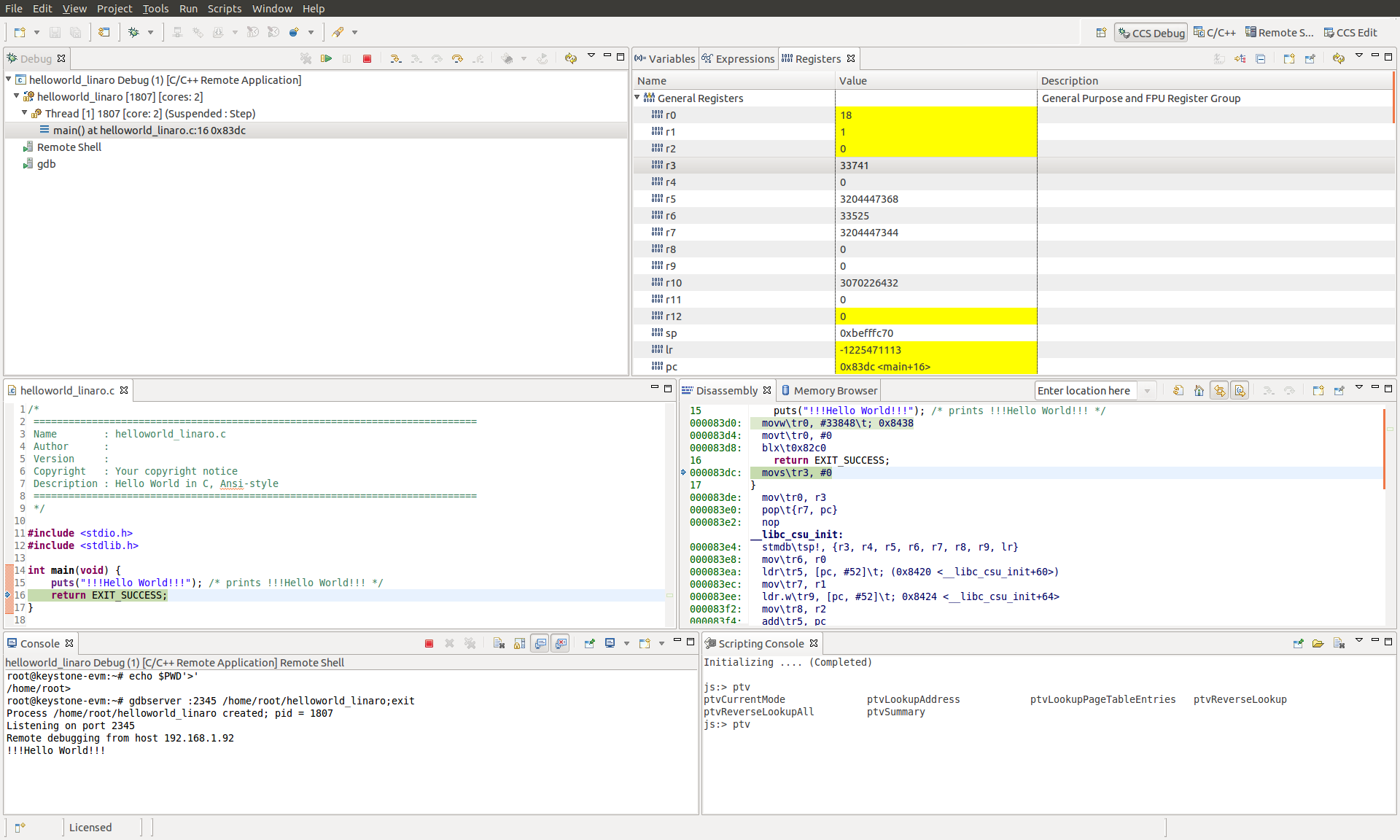
Press ***Apply***, ***Debug***



Using the debug in the debug prospective:

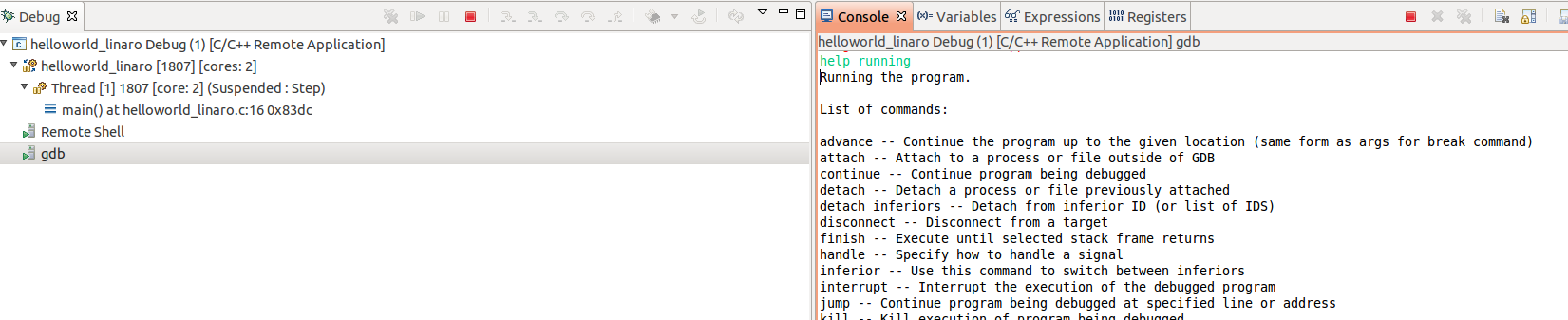


Using the debugger – one step into program:



Using the debugger – gdb console

Selecting gdb from the Debug panel allows direct control of gdb



Select the gdb and from the consol you can use almost all gdb commands, breakpoint, continue and so on.

Note – I was unable to move the pc backwards, or do reset or reload. If I want to go backwards I had to close the remote debug terminal and open it again. Obviously the r (run) command does not work (run can start execution from the beginning of the execution)

## Task 5: Configure the host proxy

## 

CCS defaults to using the system proxy settings (should not be needed if the target and the host are on the same subnet).

Right-click on the connection, select ***Properties***

From the left pane, select ***Host.*** Click ***Configure proxy settings***

From ***Network Connections,*** set ***Active Provider*** to ***Direct,*** click ***Apply*** and ***OK***