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

August 2013

Contents

[Lab 1 – EVM board bring up and out of the box demonstration 1](#_Toc357707340)

[Purpose 1](#_Toc357707341)

[Project Files 1](#_Toc357707342)

[Task 1: Prerequisites 1](#_Toc357707343)

[Task 2: Load and run standard Hello application 7](#_Toc357707344)

[Lab 2 – Build a new ARM program 11](#_Toc357707345)

[Purpose 11](#_Toc357707346)

[Task 1: Modify the file system 11](#_Toc357707347)

[Example simple code 11](#_Toc357707348)

[Unzip and decompress the file system and add the new executable 12](#_Toc357707349)

[Compressed and zip the new file system 13](#_Toc357707350)

[Reboot the EVM and run the new program 13](#_Toc357707351)

[Task 2(optional): Build U-boot, Boot Monitor and Kernel 13](#_Toc357707352)

[builds instructions 15](#_Toc357707353)

[Lab 3 – Boot Using NFS-mounted file system 18](#_Toc357707354)

[Purpose 18](#_Toc357707355)

[Task 1: Build a file system on a Linux host, install, configure and run NSF server 18](#_Toc357707356)

[Task 2: Configure U-BOOT to mount the file server and boot 20](#_Toc357707357)

[Task 3: Build a new C program in the file system, and debug it 20](#_Toc357707358)

[Lab 4 – ARM-DSP Inter Processor Communication (IPC) Using 21](#_Toc357707359)

[Purpose 21](#_Toc357707360)

[Project Details 21](#_Toc357707361)

[Lab Instructions 21](#_Toc357707362)

[Task 1: Import & Examine the Skeleton Project 22](#_Toc357707363)

[Task 2: Build the DSP Project 23](#_Toc357707364)

[Task 3: Examine ARM code 24](#_Toc357707365)

[Task 4: Connect to the EVM 24](#_Toc357707366)

[Task 5: Run the Program 25](#_Toc357707367)

[Lab 5 – Using CCS (on Linux) to build and run ARM code 28](#_Toc357707368)

[Purpose 28](#_Toc357707369)

[Prerequisites 28](#_Toc357707370)

[Task 1: CCS - Start a new cross compiler project and build it 28](#_Toc357707371)

[Task 2: GDB from the tera-terminal 32](#_Toc357707372)

[Move the Debug directory outside of CCS 32](#_Toc357707373)

[Task 3: Connect a SSH terminal into the target 34](#_Toc357707374)

[Task 4: Optional – Use gdb to debug the application 44](#_Toc357707375)

[Task 5: Configure the host proxy 48](#_Toc357707376)

### 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 a 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

# 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 on the router. If DHCP server is not available, static IP addresses will be used. The scheme for assigning static IP addresses is the following:

|  |  |
| --- | --- |
| Device | Static IP |
| Ubuntu Server | 192.168.0.100 |
| Station N EVM | 192.168.0.10+N |
| Station N first Laptop | 192.168.0.40+N |
| Station N second Laptop | 192.168.0.70+N |

### 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.

Modify file path correctly to match your OS:

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

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 will 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/stuentN 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/tftoboot/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-evm.cpio.gz (look at the release for an updated name)

setenv name\_fdt -> the name of the binary device tree, uImage-tci6638-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)

setenv boot ramfs

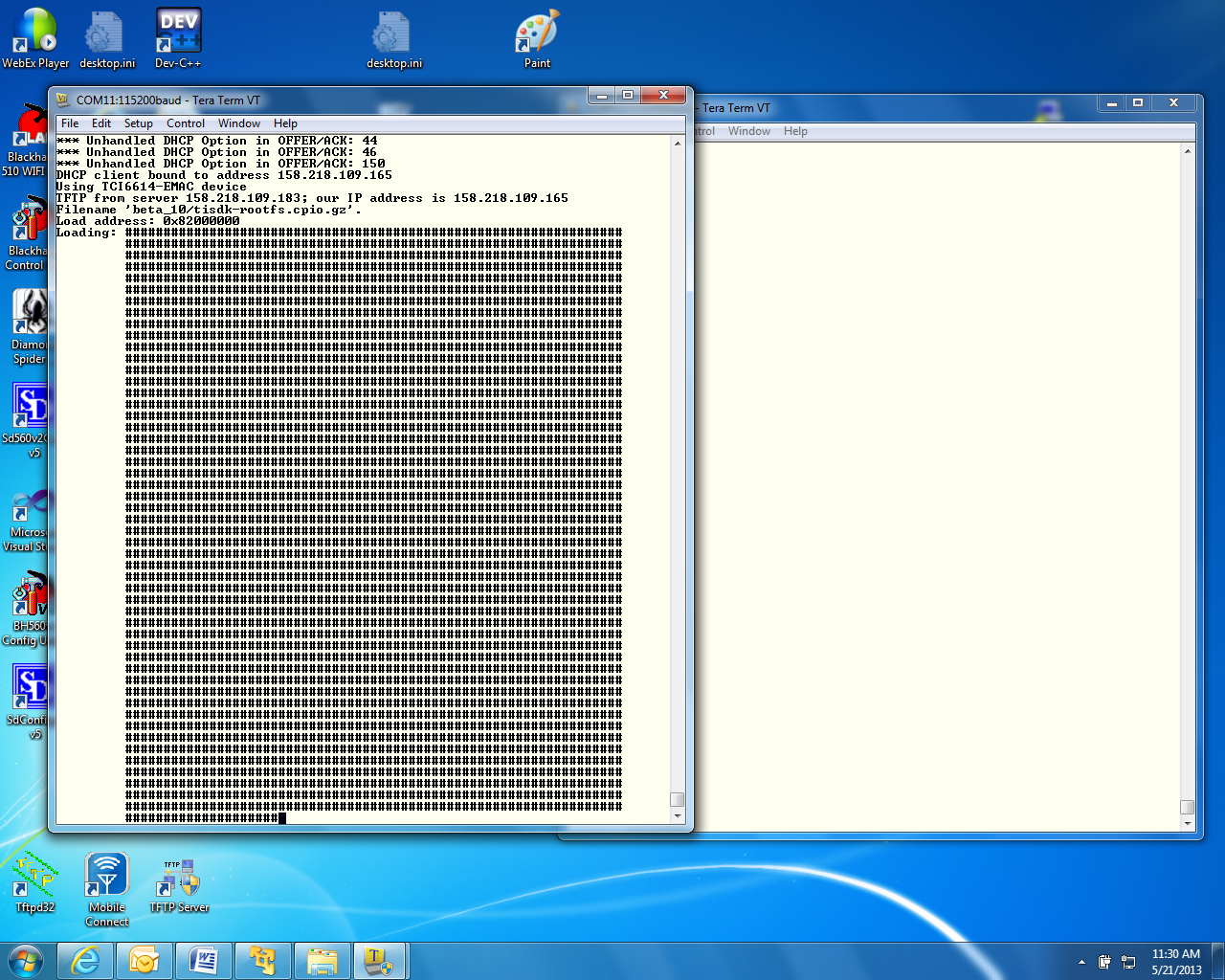
setenv tftp\_root studentN

saveenv

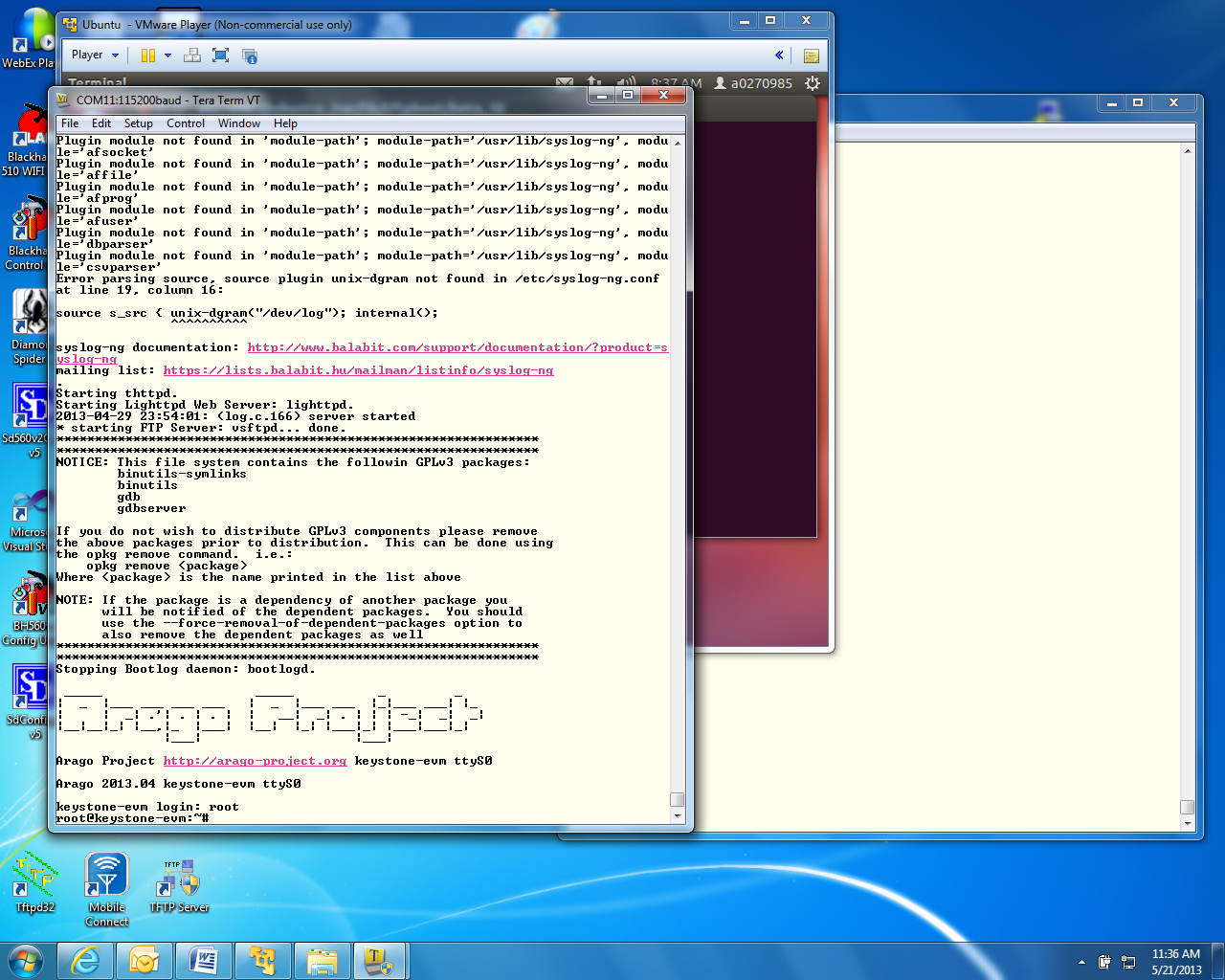
boot (or start hardware or software reboot. Hardware reboot = power cycle, software reboot, write fullrst in the BMC terminal window.

**Very important. If static IP address is used, you must make several changes to the environment. Appendix 1 shows what changes are needed**

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

## 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.

Note: Part of the boot process ARM keeps the DSP in reset using the file /usr/bin/mpmsrv, so CCS cannot connects to any of the DSP cores. See note 2 Lab 4 task 4. In order to get DSP out of reset this file should be eliminated from the file system. This can be easily done using NFS boot (Lab 3) or by building a new file system without the file /usr/bin/mpmsrv. To do so, change the name of the mpmsrv file to something like mpmsrv1 before rebuilding the file system.

### 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. Assume that the example file name is example1.c. save example1.c in a temp directory.

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:

sudo arm-linux-gnueabihf-gcc -o example1 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 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 a directory in the file system, for example, into /usr/bin

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

To eliminate the mpmsrv file (see comment above):

sudo mv /opt/filesys/studentN /usr/bin/mpmsrv /opt/filesys/studentN /usr/bin/mpmsrv1

### 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 –

sudo 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.

.

Updating the document up to here

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

In order to build these elements we need to extract the components from the Arago distribution using the git utility. To be able to access the distribution, several steps are needed to ensure the correct proxy. There are multiple ways to configure the proxy. The following is a list of steps that I did in order to access the distribution:

* 1. Configure the network proxy:
     1. Refer to "Proxy Setup" under Exploring the MCSDK in the [MCSDK User Guide for KeyStone II](http://processors.wiki.ti.com/index.php/MCSDK_User_Guide_for_KeyStone_II).
     2. Within TI network, type in the following:

export http\_proxy="http://wwwgate.ti.com:80"

export ftp\_proxy="http:// wwwgate.ti.com:80"

export https\_proxy="http:// wwwgate.ti.com:80"

export no\_proxy=".ti.com"

NOTE: The above proxy setting is valid only for the terminal window in which it is created. It is suggested to create a proxyConfig.sh file containing the lines above and then use this file (source proxyConfig.sh) for each new terminal that is opened.

* + 1. Create a file git-proxy\_ubuntu.sh under ~/Documents folder:

cd documents

sudo vi gtt git-proxy\_ubuntu.sh

* + 1. Enter the following lines in the git-proxy file:

#!/bin/bash

exec /usr/bin/corkscrew webproxy.ext.ti.com 80 $\*

* + 1. Exit from the editor to the terminal and add execution permissions to the mod:

sudo chmod 777 git-proxy\_ubuntu.sh

* + 1. In my system I had to repeat step iii, iv and v in a bin subdirectory of home, that is

cd ~/

sudo mkdir bin

sudo vi git-proxy (and write the following two lines)

#!/bin/bash

exec /usr/bin/corkscrew webproxy.ext.ti.com 80 $\*

exit from the editor

sudo chmod 777 git-proxy

cd ..

* + 1. Export the git-proxy file you created to GIT\_PROXY\_COMMAND

export GIT\_PROXY\_COMMAND=$HOME/Documents/git-proxy\_ubuntu.sh

* + 1. To verify that the proxy is set correctly do

printenv | grep proxy

The results should look like the following:

http\_proxy=http://wwwgate.ti.com:80

ftp\_proxy=http://wwwgate.ti.com:80

GIT\_PROXY\_COMMAND=/home/a0270985/Documents/git-proxy\_ubuntu.sh

https\_proxy=http://wwwgate.ti.com:80

no\_proxy=.ti.com

Not that the https\_proxy port must be 80 and not 81

* + 1. Add the following under the [global] section of ~/.subversion/servers:  
       NOTE: If the subversion directory does not exist, create it using mkdir. Copy the servers file from /etc/subversion/

sudo mkdir ~/.subversion

sudo /etc/subversion/servers ~/.subversion/.

And add the following lines at the bottom of the file after the [global] label

http-proxy-exceptions = \*.ti.com, \*.nsc.com

http-proxy-host = webproxy.ext.ti.com

http-proxy-port = 80

**Note – on my VM Ubuntu I was able to access the Arago distribution ONLY from the home directory**. Follow the instructions into the home directory and then you can move the built files to a new subdirectory

### builds instructions

git clone git://arago-project.org/git/projects/u-boot-keystone.git

cd u-boot-keystone

git reset --hard DEV.MCSDK-2013-01.10

option a: if using CC 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 build instructions (skern.bin)

cd ..

git clone git://arago-project.org/git/projects/boot-monitor.git

cd boot-monitor

git reset --hard DEV.MCSDK-03.00.00.10

make clean

make

5. Linux kernel & device tree blob build instructions

5.1: to get the code

cd ..

git clone git://arago-project.org/git/projects/linux-keystone.git

cd linux-keystone

git reset --hard DEV.MCSDK-03.08.04.10

5.2 to build kernel and DTB

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

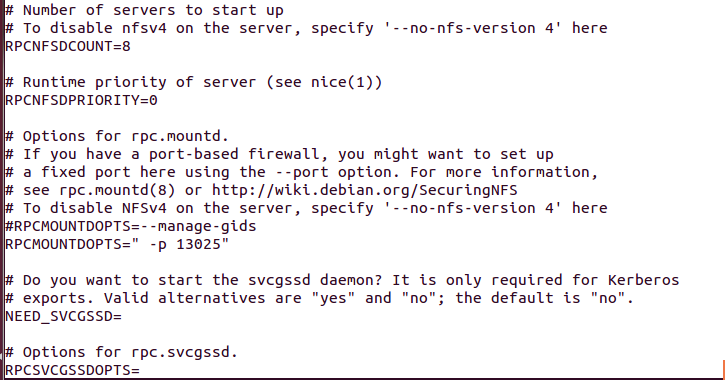
The purpose of this lab is to demonstrate how to boot the EVM when the file system resides on a different server that is mounted on the EVM, then develop a code on the Linux host and move it to the file system. The executable will be available to the ARM on the EVM. A debug session using gdb will be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

### Task 1: Build a file system on a Linux host, install, configure and run NSF server

1. Install NFS server

To install a NFS server and configure it:

* 1. Install the NFS server -> “sudo apt-get install nfs-kernel-server “
  2. Configure the nfs server, open the file /etc/default/nfs-kernel-server The file looks like the following:



Note that when the file is generated, the default value of RPCMOUNTDOPTS is

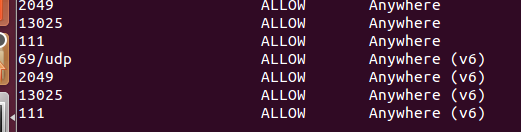
–manage-gids. The user should change this value and assign a port. In the above example the port 13025 was assigned. Any available port can be used as well.

* 1. Next the firewall should be configured to allow the NFS server. Assume the user set the port number above to 13025 then the following commands should be executed in an Ubuntu terminal

“sudo ufw allow from any to any port 111”

“sudo ufw allow from any to any port 2049”

“sudo ufw allow from any to any port 13025”

* 1. To verify that these ports are indeed open, call “sudo ufw status” you should see something like
  2. 

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
   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 directory
   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

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

The file /etc/exports looks like the following:



1. Start the nfs server -> “sudo /etc/init.d/nfs-kernel-server restart “

### 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
   3. Define the file system root directory -> “setenv nfs\_root /opt/filesys “
   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 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/bin
3. Set the terminal in the bin directory -> “cd /opt/filesys/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

~/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux/bin/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 – ARM-DSP Inter Processor Communication (IPC) Using **Msgcom**

## Purpose

The goal of this lab is to become familiar with how to utilize the *Msgcom* APIs in order to communicate between applications running between an ARM and DSP controller on a Keystone 2 EVM. You will build a project that will ultimately send messages both to and from the ARM and DSP core 0.

## Project Details

This project consists of three .out files. Two of these will be run on the ARM core and one will run on DSP core 0. The first program run on the ARM, msgrouter.out, initializes control channels between the ARM and DSP. The second program run on the ARM, test\_msgcom.out, and the program to be run on the DSP each create an *Msgcom* channel for the other to write to. These two programs work together, starting with the ARM sending a message to DSP core 0. DSP core 0 then receives and validates this message before sending a message of its own back to the ARM core. For each iteration (one message sent both ways), the size of the message sent doubles. This process continues for a total of ten messages sent both directions, utilizing the *Msgcom* APIs throughout the process. At this point, both the ARM and DSP core 0 are in charge of deleting and freeing resources they created and syncing this information with the other through the *Resource Manager* and *Agent* modules.

## Lab Instructions

Important Note

In order to build the project, several packages were used. The environment variables are not standard. Building the project involves getting various releases of software and defining lots of hard coded environment variables. Until MCSDK3 GA is released, we will not try to build the DSP or ARM projects. The source code will be used to look at the code. To run the code we will use the binaries that already were built.

## Task 1: Import & Examine the Skeleton Project

1. In the CCS Edit perspective, click on the CCS menu option *Project* 🡪 *Import Existing CCS Eclipse Project.*
2. In the *Select search-directory* box browse to ‘path-to-project-location’
3. Select the *Core0\_msgCom\_tmdxevm6648lxe\_UnittestProject\_little* project from the list of *Discovered projects*.
4. The "Copy projects into workspace" is checked by default. CCS will de-compress the project into the workspace
5. In the Project Explorer, open the file *test\_core0.cfg* as follows:
   1. Right-click on the file in the CCS Project Explorer.
   2. Select *Open With* and *XDCScript Editor.*
6. Examine the following lines in the test\_core0.cfg file that are necessary for leveraging *Msgcom:*
   1. The *MultiProc* module handles the management of the various processor IDs. The *Ipc* module is used to initialize the subsystems of IPC.

**var** Ipc = xdc.useModule('ti.sdo.ipc.Ipc');

**var** MultiProc = xdc.useModule('ti.sdo.utils.MultiProc');

* 1. The following line defines which DSP processors will be used. In this case, we will be using just DSP core 0.

MultiProc.setConfig(**null**, ["CORE0"]);

* 1. These lines include the *SharedRegion* module, which manages the shared memory allocation across processors and defines the specific location of the shared memory. Then shared memory base addresses and sizes are defined.

**var** SharedRegion = xdc.useModule('ti.sdo.ipc.SharedRegion');

SharedRegion.translate = **false**;

SharedRegion.setEntryMeta(0,

{ base: 0x0C010000,

len: 0x00070000,

ownerProcId: 0,

isValid: **true**,

name: "MSMCMem",

});

SharedRegion.setEntryMeta(1,

{ base: 0xA0100000,

len: 0x00070000,

ownerProcId: 0,

isValid: **true**,

name: "DDR3Mem",

});

* 1. *Pktlib* is the module used for allocating the messages that are passed between processors. The *ResMgr* (*Resource Manager*) is used to monitor and sync resources such as channels between processors. *Agent* is responsible for sending control messages, typically things like syncing creation and deletion of channels with the *Resource Manager* and other processors. Lastly, *Josh* (*Job Scheduler*) allows function calls made on one processing element to be executed on another. However, user does not directly exercise any *Josh* APIs.

**var** Pktlib = xdc.loadPackage('ti.runtime.pktlib');

**var** ResMgr = xdc.loadPackage('ti.runtime.resmgr');

**var** Josh = xdc.loadPackage('ti.runtime.josh');

**var** Agent = xdc.loadPackage('ti.runtime.agent');

1. Open and examine *main\_core0.c*. There are several functions/tasks to take note of:
   1. *Main()*

This function dynamically creates the initialization task (*Test\_sysInitTask*), calls *Ipc\_start()* to synchronize processors, and then calls *BIOS\_start()*

* 1. *Test\_sysInitTask()*

This task calls *system\_init()* which initializes things such as *cppi*, *qmss*, and heap in shared memory. This task also *creates AgentRxTask* which handles agent messages received on this processor. Lastly, this task creates *dspReadWriteControlTask* which performs the message passing work.

* 1. *AgentRxTask()*

Initializes an *Agen*t then creates an *Msgcom* control channel (note: this is NOT a channel used for the message passing visible on console). It then checks to see if *Agent* is up and running on the ARM side. If so, it polls waiting for *Agent* messages.

* 1. *dspReadWriteControlTask()*

This is where the magic happens. A channel is created and then synced with ARM by using the *Agent* and *Resource Manager*. Then, this DSP core searches for the channel created on ARM by name. After this is found, this DSP waits for a message and upon receiving one, validates the contents. Then a new message is generated and sent on the other channel to the ARM core. After this procedure is followed for a total of ten times, the channels are deleted and synced with ARM via *Agent* and *Resource Manager*.

## Task 2: Build the DSP Project

At this point we skip the building process and use the pre-built out file

1. Right-click on the project and select *Build Project*
2. The project should build without errors or warnings. If it doesn’t build properly, attempt to figure out why. Otherwise, ask the instructor.

## Task 3: Examine ARM code

1. Open *main.c* from within Msgcom ARM Code/msgcom/test directory in your favorite text editor.
2. There are several functions/tasks to take note of:
   1. *Main()*

The *main()* code on ARM does not call a system initialization task, instead it takes care of much of the initialization within the *main()* function. Several of the obvious differences are that there is no need to call *Ipc\_start()*, *BIOS\_start()*, and that *udma* is used for memory allocation and packet creation unlike *Pktlib* on the DSP side.

* 1. *agentInitTask()*

Initializes the *Agent* and configures it to communicate with DSP core 0. This step was not required on the DSP side.

* 1. *agentRxTask()*

Near identical to that of the DSP. It polls waiting for an *Agent* message to be received.

* 1. *armReadWriteControlTask()*

This is the magic on the arm side. First the code tries to sync with the channel created by the DSP. Once this is accomplished it creates a channel for the DSP to write to. These are both synced with the *Resource Manager* through *Agent*. The code then creates a message using *udma* (instead of *Pktlib* on DSP) and sends it over the channel created by the DSP. Afterwards, it waits until it receives a message from the DSP. Once this process is repeated for a total of ten times, the channels are deleted and once again synced with DSP via *Agent* and *Resource Manager*.

## Task 4: Connect to the EVM

**Note1- For this test we run the mounted file system. The location of the file system is in /opt/filesys (if the user obeyed the default setting of Lab 3**

**Note 2 – The binary file mpmsrv in directory /usr/bin may keep the DSP in reset so CCS cannot be connected to the DSP cores. To overcome this problem, change the name of the file from mpmsrv to mpmsrv1 (or any other name) so the ARM will release the DSP cores**

**Note 3- The binary files to load into the ARM file system will be given to the use:**

1. **The compressed file MsgcomArmCode.tar.gz will be available on ftp site that will be given by the instructor. Load it to a temporary directory ~/temp**
2. **Next the file is untar and a directory is build**
3. **Change the directory name to testMsgCom (or any other name) eliminate blanks**
4. **Copy the two out files into the /opt/filesys/usr/bin directory. These files will be part of the EVM file system**
5. Open a ARM tera-terminal session
   1. Under the category *Session*, select *Serial* and chose the appropriate *Serial line*.
   2. Set *Speed* to 115200
   3. Click on the *Serial* category and ensure the correct serial line, the speed is 115200, data bits is set to 8, stop bits is set to 1 and both parity and flow control are set to none.
   4. Go back to the *Session* category and hit open.
6. When the window prompts for username, enter “root”. There is no password. At this point, you are connected to the ARM portion.
7. Go back to the CCS window.
8. Switch to the Debug CCS Perspective by selecting the CCS menu option *Window* 🡪 *Open Perspective* 🡪 *CCS Debug*.
9. If you have previously created a *Target Configuration* .ccxml file, then please skip to **Task 5**.
10. 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*.
11. Define the new target.
    1. Set the File name based on the EVM model, *<model>.ccxml*. For example, ‘TCI6638.ccxml’ or ‘keplerEVM.ccxml’
    2. Leave the *Location* the default value:

“C:\Documents and Settings\student\ti\CCSTargetConfigurations”

* 1. Click the *Finish* button. The .ccxml file will now open in a GUI-based view with the Basic tab active.

*Basic Tab*

* 1. The *Connection* drop-down menu identifies the emulator type. In this case, select “Texas Instruments XDS2xx USB Emulator”
  2. *Board or Device* identifies the TI processor device. In this case, use TCI6638
  3. Under *Save Configuration*, click the *Save* button

## Task 5: Run the Program

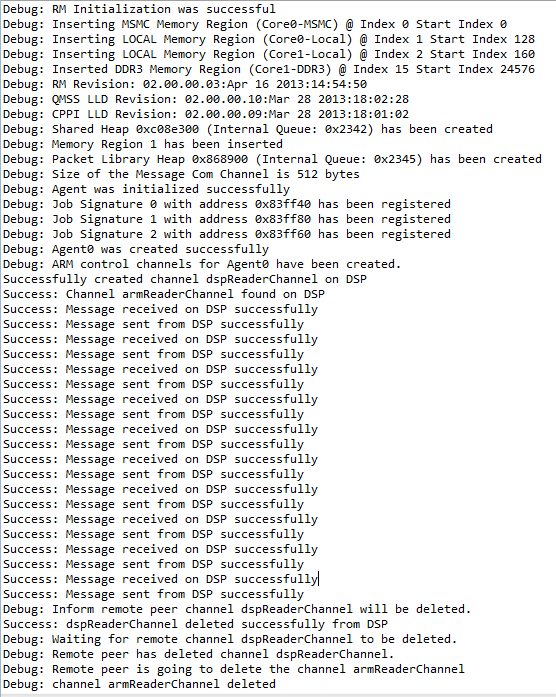
1. In the tera-terminal ARM session, enter the following command:

/usr/bin/msgrouter.out - n 4 - d 10 &

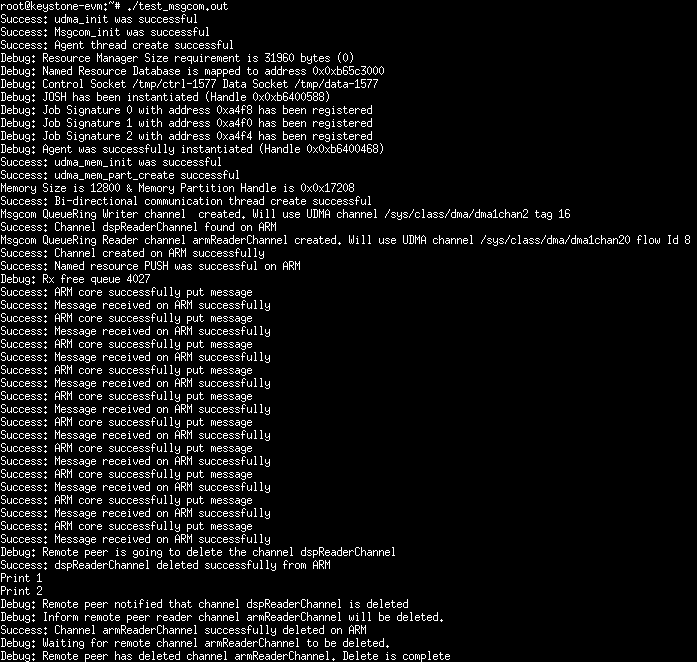
1. Hit enter.
2. Enter the following command:

/usr/bin/test\_msgcom.out

1. In your CCS window, select *View* 🡪 *Target Configurations.* Your newly-created .ccxml target configuration file should be available under *User Defined* target configurations.
2. Right-click on the target configuration .ccxml file that was created and select *Launch Selected Configuration*
3. This will bring up the *Debug* window.
   1. Select Core 0
   2. Right-click and select *Connect Target*
4. Select Core 0 and load the .out file associated with this project.
   1. Select the CCS menu option *Run* 🡪 *Load* 🡪 *Load Program*
   2. Click *Browse project…*
   3. Select *Core0\_msgCom\_tmdxevm6638lxe\_UnittestProject\_little.out* and click OK
   4. Click OK to load the application to the target (Core 0)
5. Now run the application. To do this, select the CCS menu option *Run* 🡪 *Resume*
6. A successful run should produce the following output on the console:



1. In your the tera-terminal ARM session, a successful run is accompanied by the following output:



1. Ensure that the application behaves as expected by checking the following items:
   1. Ensure that there are ten “Success: Message received on \_\_ successfully” and “Success: Message sent from \_\_ successfully” prints on both the Putty console and CCS console.
   2. Ensure neither the tera-terminal ARM console or CCS console printed a line beginning with “Error:”
   3. Ensure the prints mentioning channel deletion match those seen in the above screenshots.

# Lab 5 – 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

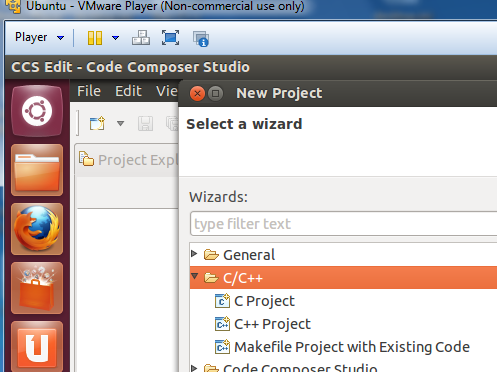
1. To start CCS go to CCS eclipse directory (in my setting it is in /home/a0270985/ti/CCS\_5\_4/ccsv5/eclipse ) and do

sudo ./ccstudio

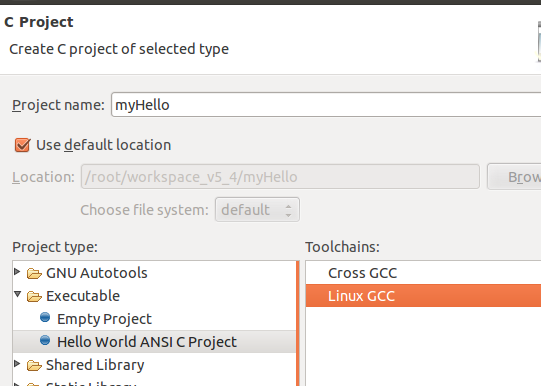
1. After CCS start, and in the edit prospective start a new (not CCS) project:

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

C/C++ and then C project



From the C project choose Hello World ANCI C project, Toolchians 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) ;

}

}

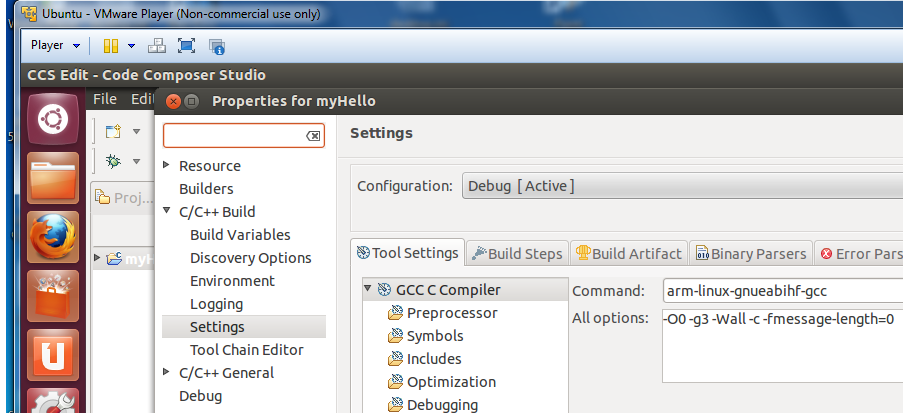
**printf**(" END ->>> i= %d k = %d j = %d l = %d \n", i,k,j,l) ;

**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.

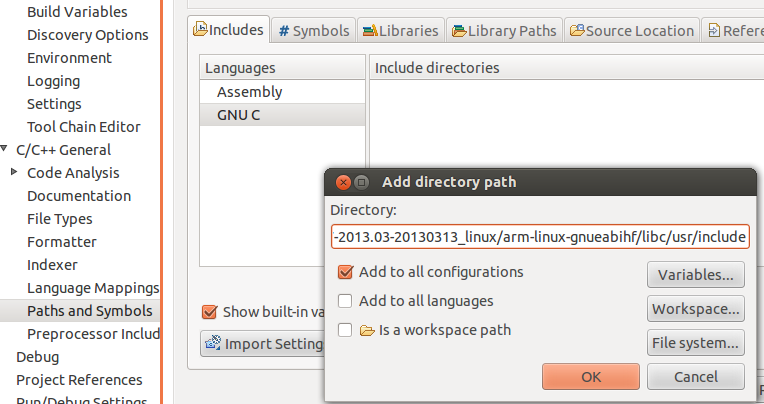
Next the include path to the tools should be configured. From project properties select

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…

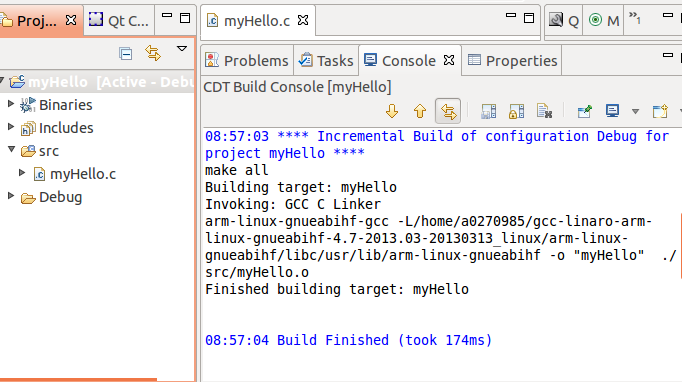
Enter Name: **PATH**, ***Add to all configurations***

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

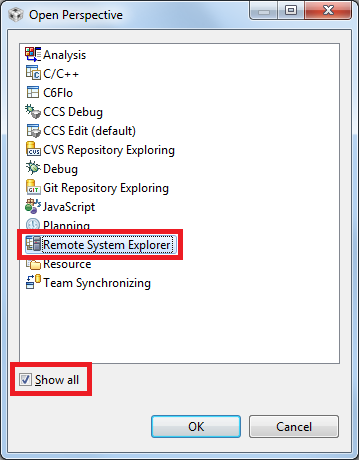
r

c

You can play with other gdb instructions

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

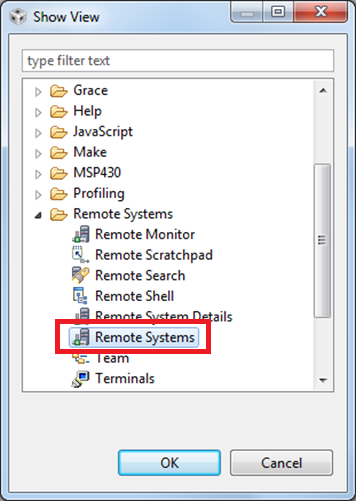
1. After reboot and login the target from the tera-terminal, do ifconfig eth0 and see what is the IP address of the target. Note, the IP address is assigned by the DHCP server.
2. Start CCS; Select ***Window🡪Open Perspective🡪Other***  
   Open Perspective: Check ***Show All***, Select ***Remote System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



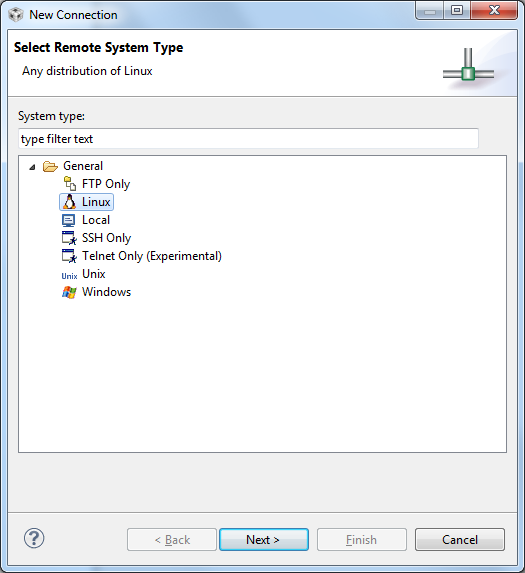
1. Select ***CCS Edit*** perspective



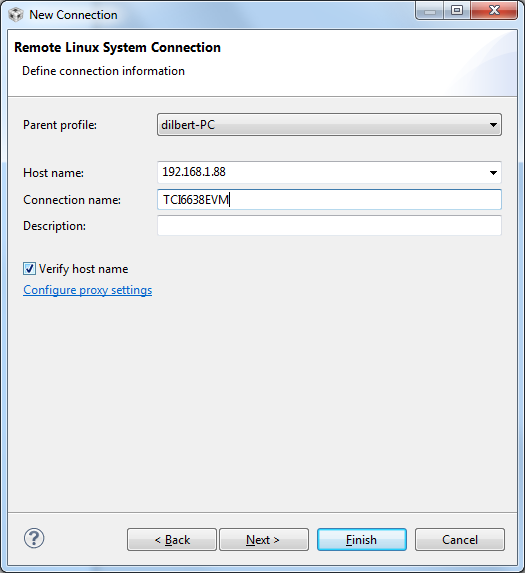
1. Select ***Window🡪Show View🡪Other***  
   Expand Remote Systems, Select ***Remote Systems***  
   Optionally drag Remote Systems to different location  
   (next to Project Explorer tab is a nice spot…)



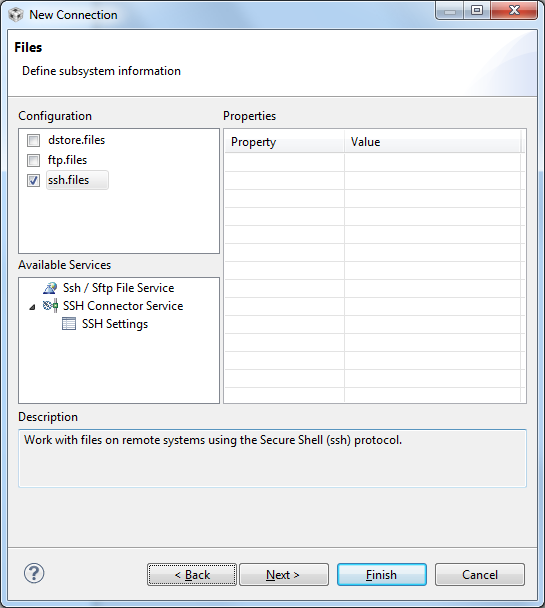
1. From Remote Systems window, Right-Click🡪***New🡪Connection…*** in the window, select linux and click next



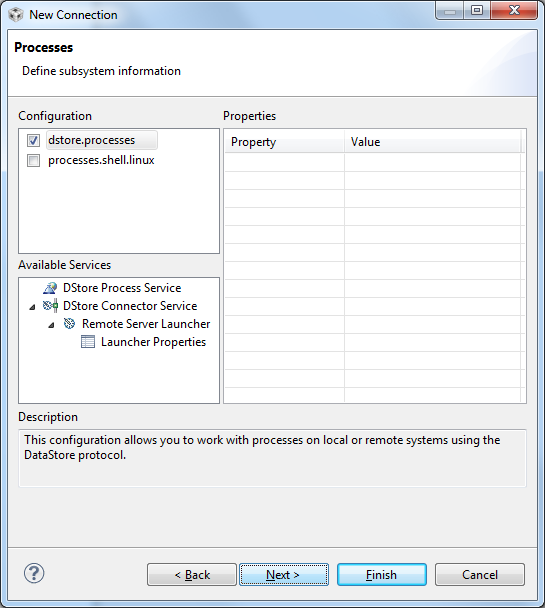
1. Configure the remote linux connection, enter the IP address as the host name, give connection name and profile name:



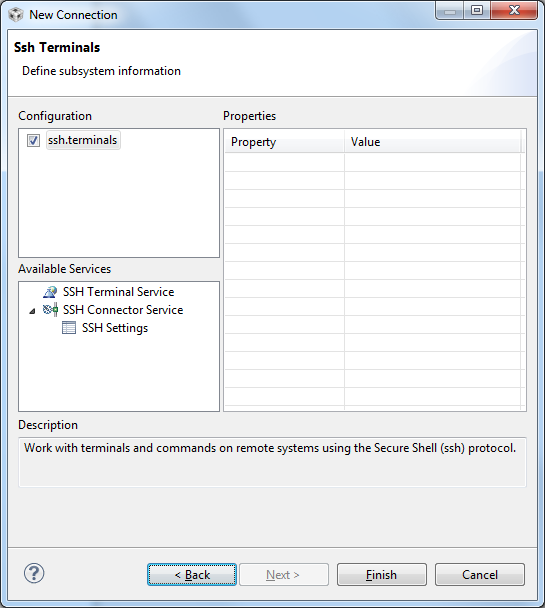
1. Files, select ssh.files



1. next
2. Select processes.shell.linux

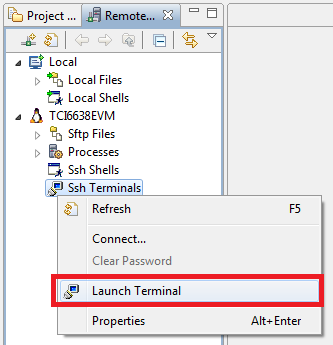


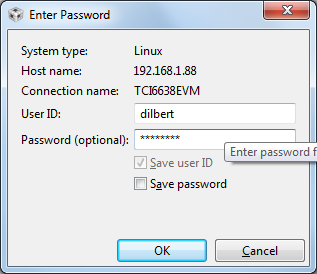
1. Next
2. Select ssh.shells
3. Next

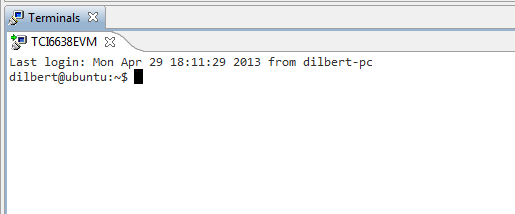


1. Select ssh.terminals
2. Finish
3. Opening a terminal:  
    Right-Click, Select ***Launch Terminal***  
    Enter password (for root user name leave the password blank)

After warnings a terminal will be opened into the target ARM



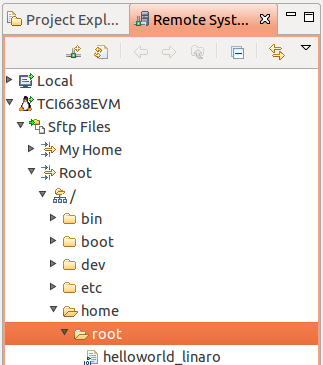
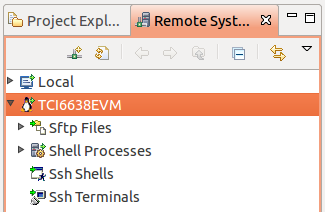
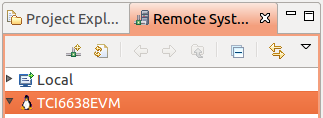
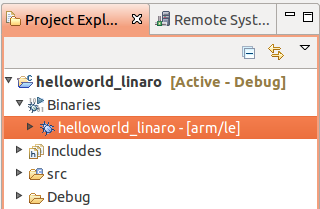




Note – the green arrow indicates that the connection is good. If there is not green arrow, and a proxy server is involved, there may be an additional step required to configure the proxy. Instructions how to configure the proxy are at the end of the Lab. (Task 5)

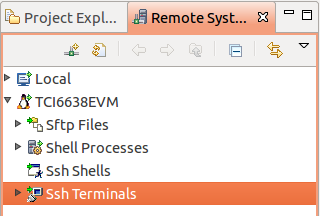
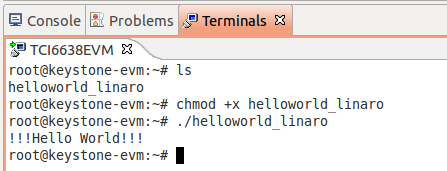
1. Copy and paste binaries onto the target

Select binary file and the source file from project  
Press ***Ctrl-C*** (or right-click, Copy)  
 Click on ***Remote Systems*** tab  
Navigate through the Sftp 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



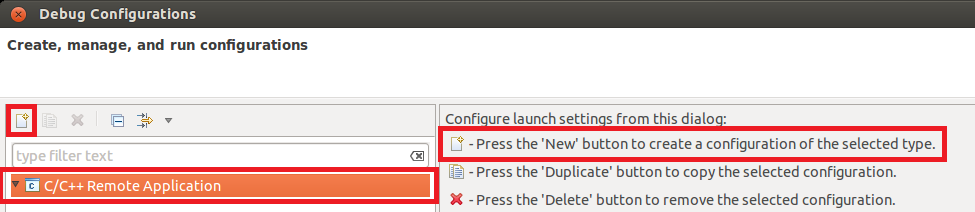
## Task 4: Optional – Use gdb to debug the application

In the edit prospective, select the project myHello

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

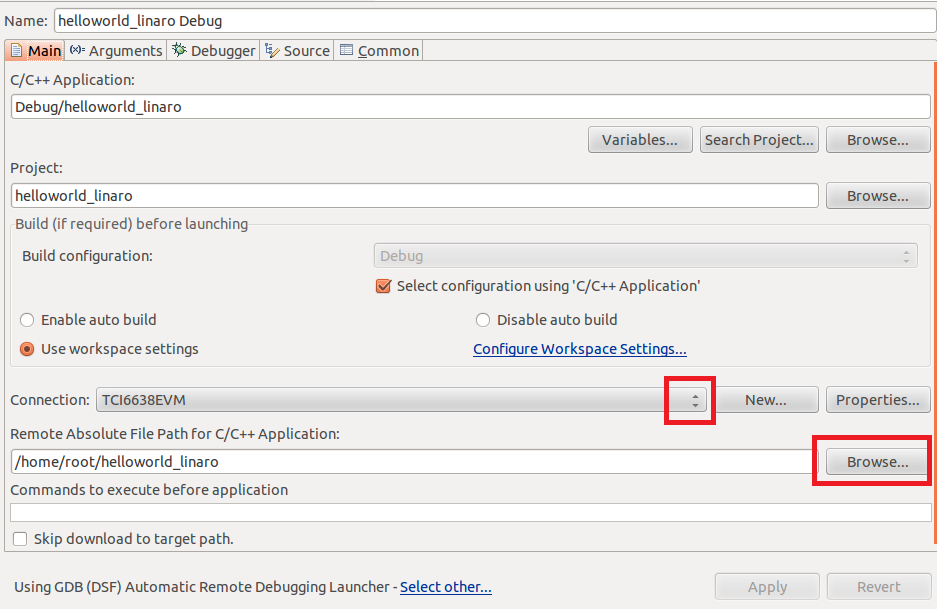
Select ***C/C++ Remote Application***

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:

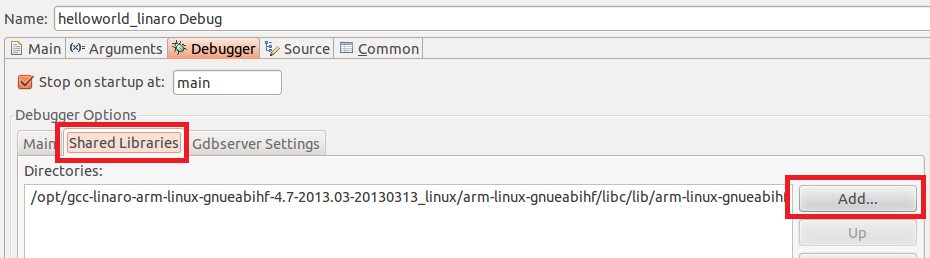
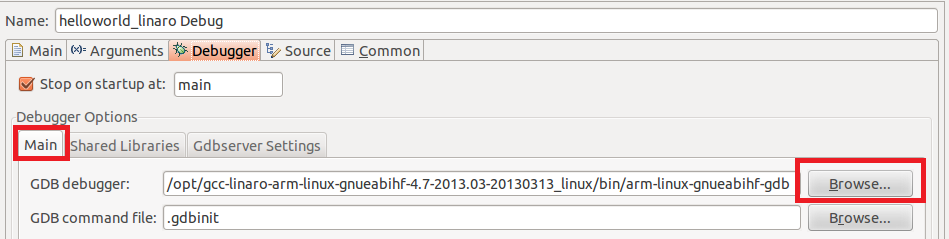
***Main*** sub-tab:

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

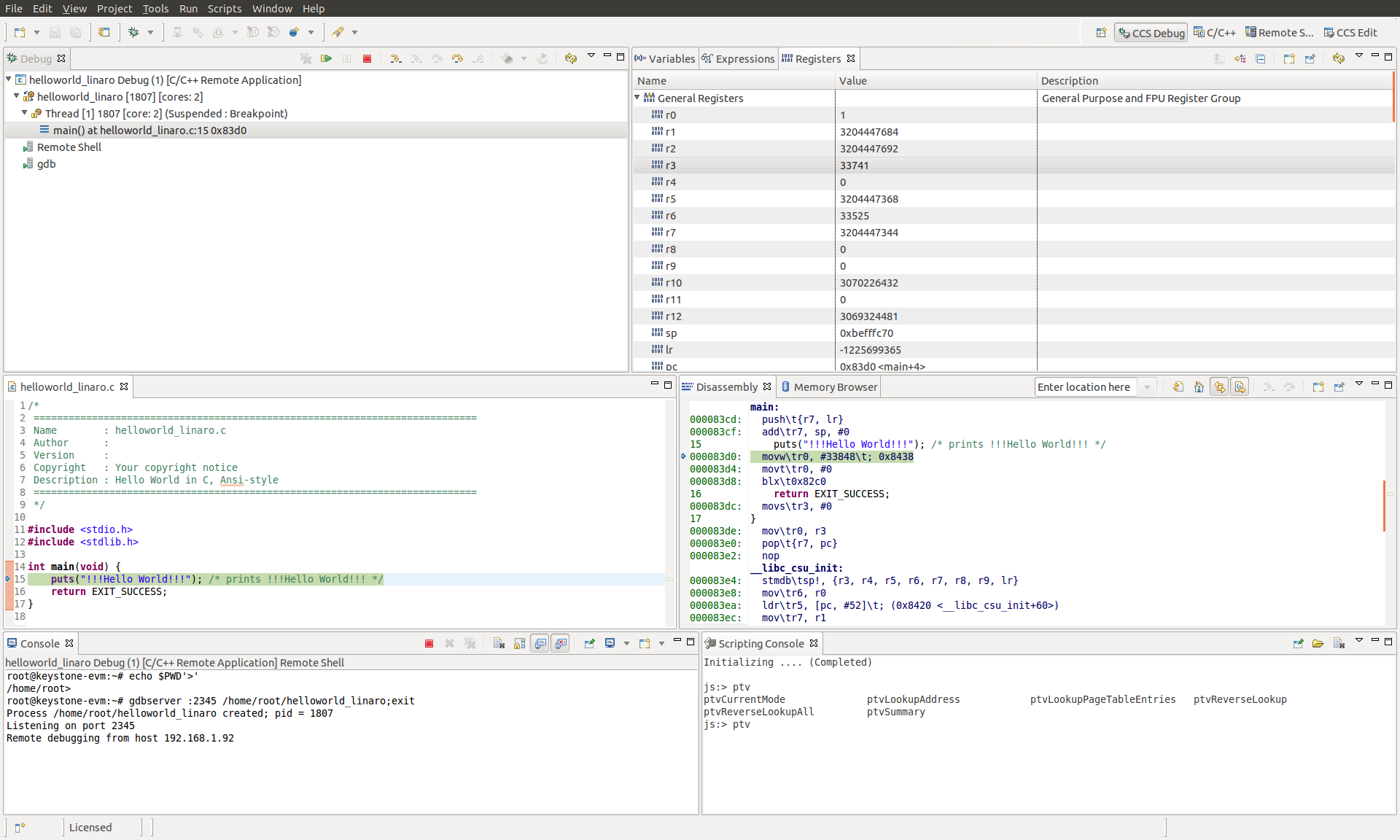
***Shared Libraries*** sub-tab:

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

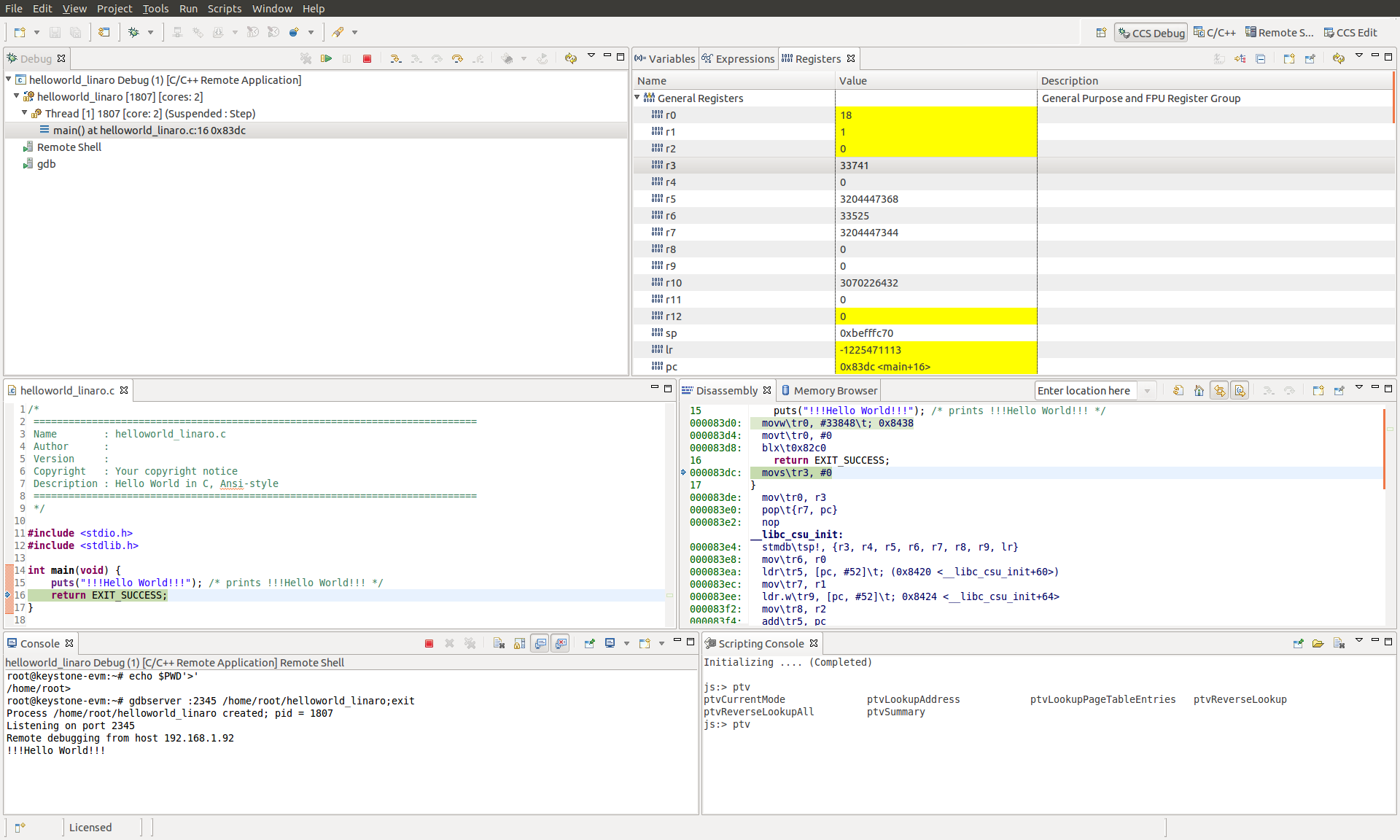
Press ***Apply***, ***Debug***



Using the debug in the debug prospective:

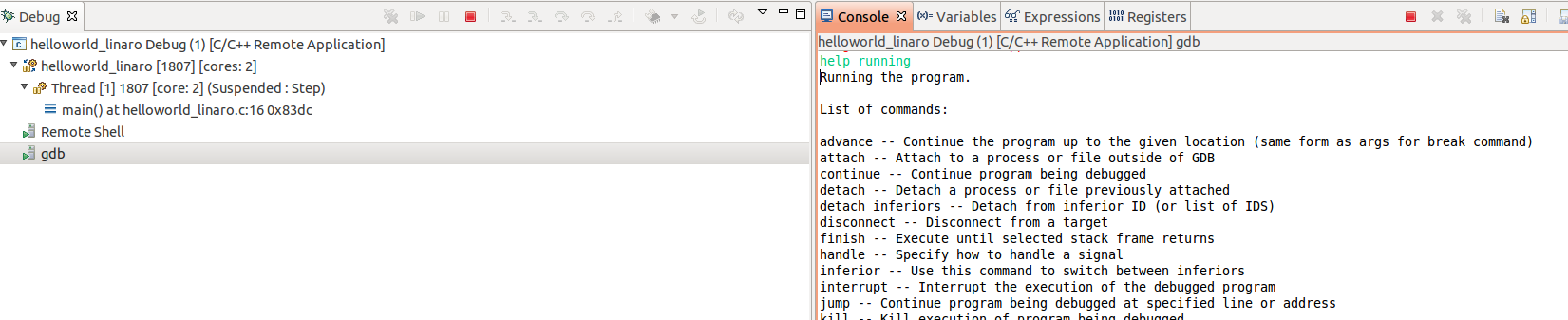


Using the debugger – one step into program:



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 de 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***

# Appendix 1

How to use Fixed IP address in stead of DHCP

================

1. U-boot IP address

To statically configure an IP address for u-boot, use the setenv command:

setenv ipaddr <ip\_address>

If static IP address is configured in u-boot, then it is not applicable to use dhcp to download files.

2. It is necessary to change all downloading commands in printenv from 'dhcp <file\_name>' to 'tftp <file\_name>'

For example:

From

get\_fdt\_net=dhcp ${addr\_fdt} ${tftp\_root}/${name\_fdt}

get\_fdt\_ramfs=dhcp ${addr\_fdt} ${tftp\_root}/${name\_fdt}

get\_fs\_ramfs=dhcp ${addr\_fs} ${tftp\_root}/${name\_fs}

get\_kern\_net=dhcp ${addr\_kern} ${tftp\_root}/${name\_kern}

get\_kern\_ramfs=dhcp ${addr\_kern} ${tftp\_root}/${name\_kern}

get\_mon\_net=dhcp ${addr\_mon} ${tftp\_root}/${name\_mon}

get\_mon\_ramfs=dhcp ${addr\_mon} ${tftp\_root}/${name\_mon}

get\_uboot\_net=dhcp ${addr\_uboot} ${tftp\_root}/${name\_uboot}

get\_uboot\_ramfs=dhcp ${addr\_uboot} ${tftp\_root}/${name\_uboot}

to

get\_fdt\_net=tftp ${addr\_fdt} ${tftp\_root}/${name\_fdt}

get\_fdt\_ramfs=tftp ${addr\_fdt} ${tftp\_root}/${name\_fdt}

get\_fs\_ramfs=tftp ${addr\_fs} ${tftp\_root}/${name\_fs}

get\_kern\_net=tftp ${addr\_kern} ${tftp\_root}/${name\_kern}

get\_kern\_ramfs=tftp ${addr\_kern} ${tftp\_root}/${name\_kern}

get\_mon\_net=tftp ${addr\_mon} ${tftp\_root}/${name\_mon}

get\_mon\_ramfs=tftp ${addr\_mon} ${tftp\_root}/${name\_mon}

get\_uboot\_net=tftp ${addr\_uboot} ${tftp\_root}/${name\_uboot}

get\_uboot\_ramfs=tftp ${addr\_uboot} ${tftp\_root}/${name\_uboot}

3. Linux Kernel IP address

To statically configure IP address for Linux Kernel, The bootargs need to be modified.

Replace "ip=dhcp" in bootargs with "ip=<ip\_address>:::::eth0:off".