Processor SDK Linux Training: Hands on with the Linux SDK

Contents

Introduction

This lab is going to give you a hands on tutorial of the Processor Linux SDK. Each of the following sections below will walk you through a particular part of the SDK describing the actions that are about to be performed, the key points to take away, and the step-by-step instructions to complete the lab. If you have questions or feedback please e-mail the sdk_feedback@list.ti.com (mailto:sdk_feedback@list.ti.com) mailing list.

NOTE: In this guide commands to be executed for each step will be marked in **BOLD**

Lab Configuration

The following are the hardware and software configurations for this lab. The steps in this lab are written against this configuration. The concepts of the lab will apply to other configurations but will need to be adapted accordingly.

Hardware

- AM335x EVM-SK (TMDSSK3358) - Order Now (https://estore.ti.com/TMDSSK3358-AM335x-Starter-Kit-P3110.aspx)

> **NOTE**
>
> This lab has also been validated with the AM437x EVM-SK (TMDXSK437X) (https://store.ti.com/TMDXSK437X-AM437x-Starter-Kit-P46767.aspx) and AM572x EVM (TMDXEVM5728) (https://store.ti.com/tmdxevm5728.aspx). Other target boards can be used, but the steps below related to serial and ethernet connections may differ.

- Router connecting AM335x EVM-SK and Linux Host (For Remote Matrix lab)
- USB cable connection between AM335x EVM-SK and Linux Host using the micro-USB connector (J3/USB0) for serial communication
- 5V power supply (12V power supply required for use with AM57x EVM)

Software

- A Linux host PC. For more information, see the Processor SDK Linux Getting Started Guide.
- Processor Linux SDK installed. This lab assumes the latest Processor Linux SDK is installed in /home/sitara. If you use a different location please modify the below steps accordingly. For more information, see the Processor SDK Linux Installer.
- SD card with Processor Linux SDK installed.
  - For help creating a 2 partition SD card with the SDK content see Processor SDK Linux create SD card script.
- CCSv6 installed. This lab assumes CCS has been installed in /home/sitara. If you use a different location please modify the below steps accordingly.
  - The CCS installer can be downloaded from the SDK for Sitara Processors (http://www.ti.com/tool/linuxezsdk-sitara) page. If extracted into the same directory as the SDK installer then the SDK installer can take care of installing CCS as well.

Out-Of-Box Experience

Description

This section will cover the steps to boot the target board and how to use the Matrix application launcher, which is the default application run when the board is booted.

Key Points

- How to boot the target board
- Navigating Matrix using the touchscreen
- Navigating Matrix using the remote interface

Lab Steps

1. Connect the cables to the EVM. For details on where to connect these cables see the *Quick Start Guide* that came with your EVM.
   1. Connect the USB cable between your Linux host and the mini-USB connector of the EVM.

      > **NOTE**
      >
      > This will provide the serial connection. For other boards you may need to use a different cable such as a straight serial cable.

   2. Connect the network cable.

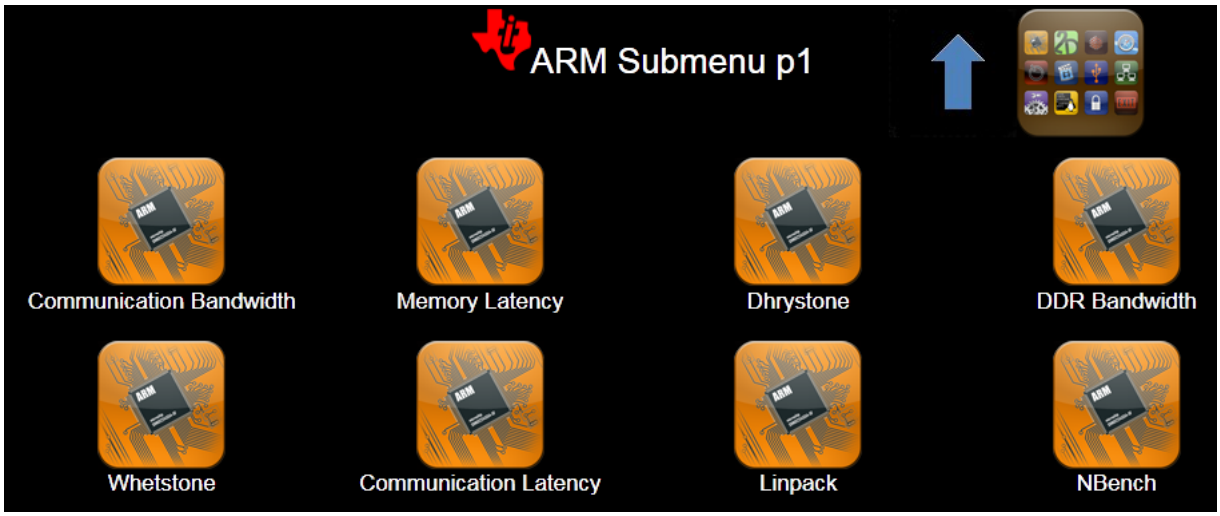
      > **NOTE**
      >
      > To ensure you receive a valid ipaddr when using the AM335x EVM-SK, be sure to connect the ethernet cable to the port closest to the User Push Buttons.

   3. Insert the SD card with Processor Linux SDK installed into the SD connector. For more information, see Processor SDK Linux create SD card script.
   4. Insert the power cable into the 5V power jack.
2. Power on the AM335x EVM-SK. For power button position, consult your *Quick Start Guide*. Allow the boot process to finish. You will know when the boot process has finished when you see the Matrix application launcher on the LCD screen.

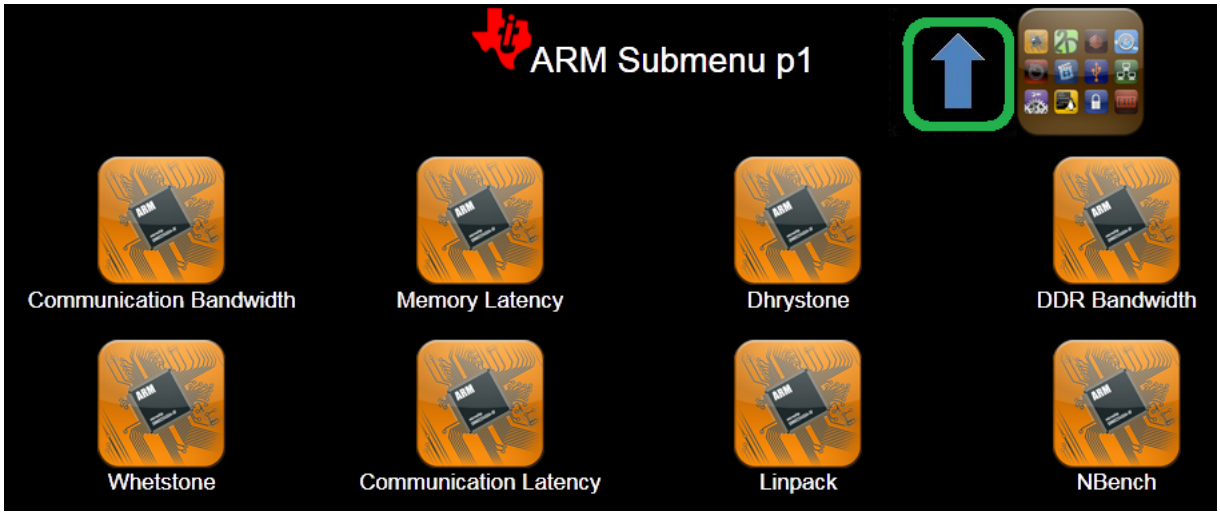
   > **NOTE**
   >
   > You may be required to calibrate the touchscreen. If so follow the on screen instructions to calibrate the touchscreen.

3. Navigate Matrix using the touchscreen.

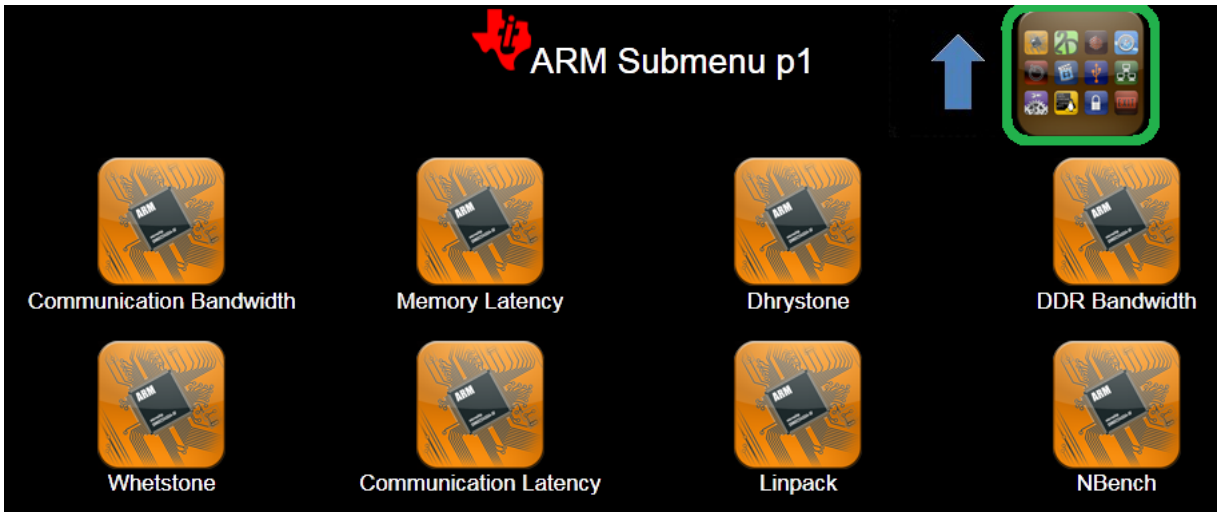
- The icons on the screen will either launch an application or will display a submenu of applications to launch. To select an item just touch the icon on the screen. **Touch the ARM icon to open the ARM submenu and display the applications available within that menu**.



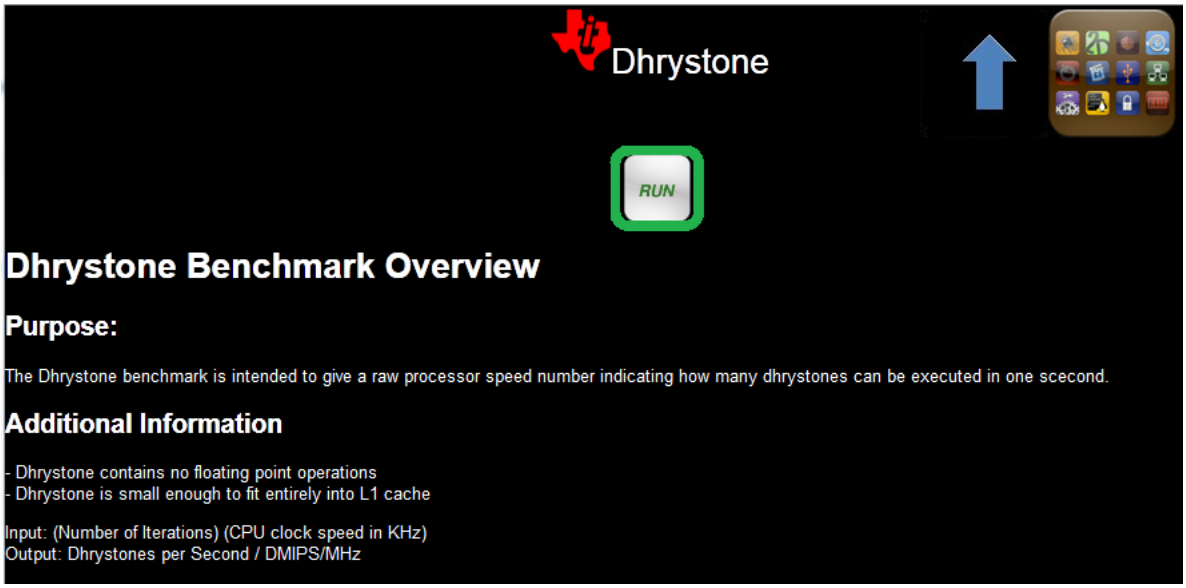
- Whenever you enter a submenu or the description screen of an application (if that application has a description screen) you will see two additional icons appear.
  - The "Up Arrow" Icon (highlighted in green) will return you to the previous screen. This is equivalent to a *Back* functionality. **Touch the Up Arrow to be returned to the main matrix menu and then select ARM again to come back to the ARM submenu**.



- The "Multi" Icon (Highlighted in green) will return you to the Matrix main menu. This is equivalent to a *Home* functionality.



- To run an application touch the application's icon. If the application has a description screen, you will be shown the description and given a **Run** button (highlighted in green). If the application does not have a description touching the application's icon will start execution of that application. **Touch the Dhrystone application to see the Dhrystone application description screen then touch the RUN button to launch the application**.



- When Dhrystone finishes running you can review the output on the LCD. **Touch the Multi Icon to return to the Matrix main menu**.
- Where there are more applications available in Matrix than can be displayed on the screen you will see arrows appear in the upper-right and upper-left corners. Touching these arrows will switch pages to the next page. **Touch the Right arrow to see the next page of applications**.

- To return to the previous page **touch the left arrow**.

4. Navigating Matrix using the remote interface is similar to navigating Matrix with the touchscreen.

- **IMPORTANT**

  Do NOT use your browser's back or home buttons as this will navigate you away from the Matrix page.

  NOTE

  Refreshing the page will restart your connection with Matrix.

- In order to connect to the remote Matrix interface, you will need to know the IP address of your EVM. To find this perform the following actions in Matrix.

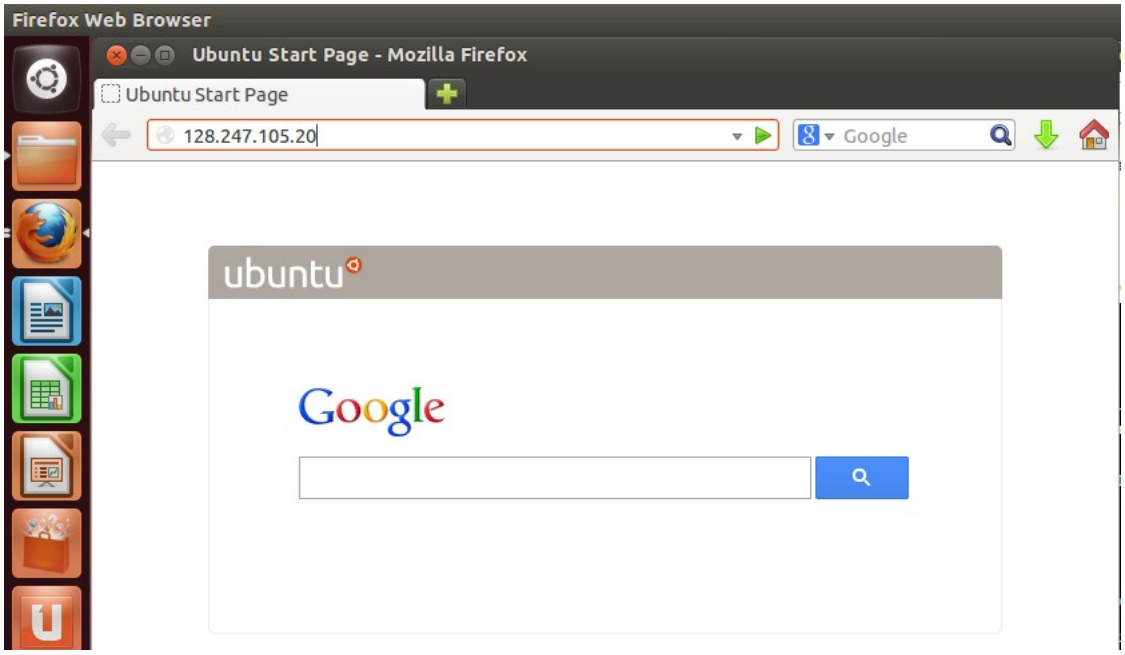
  IMPORTANT

  On the AM335x EVM-SK, please insure the ethernet cable is plugged into the port closest to the user push buttons. Otherwise you will not receive a valid ipaddr for the board.

  - **Touch the Settings icon from the Matrix main menu**.
  - **Touch the Network Settings icon from the Settings Submenu**.
  - You should see output like the following. The EVM IP address has been highlighted in green.

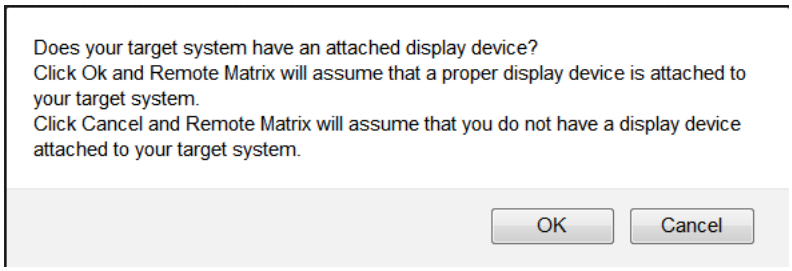
```
------------------------------------------------------------
Network Settings
------------------------------------------------------------

eth0      Link encap:Ethernet  HWaddr 40:5F:C2:76:40:30
          inet addr:128.247.105.65  Bcast:0.0.0.0  Mask:255.255.254.0
          UP BROADCAST RUNNING ALLMULTI MULTICAST  MTU:1500  Metric:1
          RX packets:37670 errors:0 dropped:7197 overruns:0 frame:0
          TX packets:1672 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6611877 (6.3 MiB)  TX bytes:467308 (456.3 KiB)
          Interrupt:40

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:225 errors:0 dropped:0 overruns:0 frame:0
          TX packets:225 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:276607 (270.1 KiB)  TX bytes:276607 (270.1 KiB)
```

- To connect to Matrix you should **enter the EVM IP address obtained in the previous step into your browser's address bar**. You can launch a browser in your Linux host by **clicking the firefox icon** highlighted in the image below.



- You should be prompted to select whether your EVM has a display device attached. Since the LCD is attached to the EVM **select OK**.
- The reason this question is asked is because Matrix does not support shipping graphical data over the remote interface. Applications which are graphical such as the 3D demos will need to be viewed on the LCD attached to the board. An answer of no (pressing Cancel) will prevent graphical applications from being launched from remote Matrix.

> **NOTE**
>
> You can change your selection at any time by pressing the *Refresh* button on your browser to re-connect to Matrix.



5. When you are finished exploring remote Matrix **close the browser**.

Using the Built-in GCC Toolchain

Description

This section will walk you through the location of the built-in GCC toolchain found in the SDK. You will also learn how and when to use the environment-setup script to configure your environment to use this toolchain. Additional information on using this toolchain can be found at Processor Linux SDK GCC Toolchain.

Key Points

- The cross-compiler is now located within the SDK in the linux-devkit directory
- There are many libraries pre-built for the ARM core available in the linux-devkit directory as well
- Sourcing the environment-setup script configures you system to use this cross-compiler and pre-built libraries
- How to identify when the environment-setup script has been sources
- How example applications use the environment-setup script

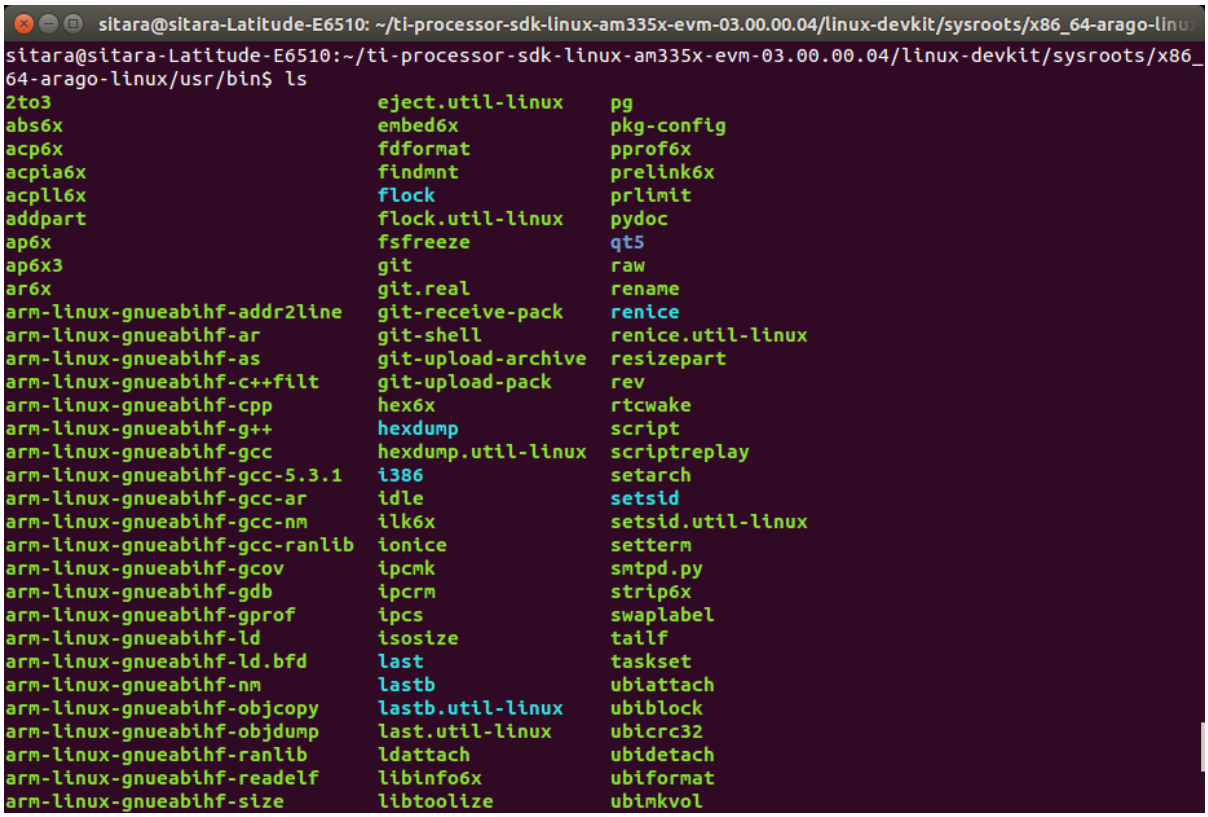
Lab Steps

1. Open a terminal window by **clicking on the Terminal icon on the Launcher toolbar**.
2. The cross-compiler is located in the linux-devkit/sysroots/<Arago Linux>/usr/bin directory of the SDK installation directory. In the terminal window enter the following commands, replacing the <machine>, <version>, and <Arago Linux> fields with the target machine you are using and the SDK version installed. The <Arago Linux> version may be *x86_64-arago-linux*.

> **NOTE**
>
> You can use TAB completion to help with this.

```
cd /home/sitara/ti-processor-sdk-linux-<machine>-<version>/linux-devkit/sysroots/<Arago Linux>/usr/bin
ls
```

3. You should see a listing of the cross-compile tools available like the one below.



4. To locate the pre-built ARM libraries perform the following commands:

```
cd /home/sitara/ti-processor-sdk-linux-<machine>-<version>/linux-devkit/sysroots/<Arago Linux>/usr/lib
ls
```

You should now see a listing of all the libraries (some are contained within their individual sub-directories) available as pre-built packages within the SDK.

5. In order to make it easier to perform cross-compilations and ensure linking with the proper cross-compiled libraries instead of the host system libraries the *environment-setup* script has been created in the linux-devkit directory. This script will configure many standard variables such as CC to use the cross-compile toolchain, as well as adding the toolchain to your PATH and configuring paths for library locations. To utilize the setting provided by the environment-setup script you will need to *source* the script. Perform the following commands to source the environment-setup script and observe the change in the CC variable:

```
echo $CC
cd /home/sitara/ti-processor-sdk-linux-<machine>-<version>/linux-devkit
source environment-setup
echo $CC
```

You should have observed that the CC variable now contains the value of *arm-linux-gnueabihf-gcc* after the sourcing of the environment-setup script. There was also another change that occurred which was that your standard prompt changed from *sitara@ubuntu* to *[linux-devkit]*. The purpose of this change is to make it easy to identify when the environment-setup script has been sourced. This is important because there are times when you DO NOT want to source the environment-setup script. A perfect example is when building the Linux kernel. During the kernel build there are some applications that get compiled which are meant to be run on the host to assist in the kernel build process. If the environment-setup script has been sourced then the standard CC variable will cause these applications to be built for the ARM, which in turn will cause them to fail to execute on the x86 host system.

6. As mentioned above sometimes it is not appropriate to source the environment-setup script, or you only want to source it during a particular build but not affect your default environment. The way this is done in the SDK is to source the environment-setup script inside of the project Makefile so that it is used only during the build process. To see this perform the following actions:

- Close the open terminal since it already has the environment-setup script sourced.
- Open a new terminal

```
cd /home/sitara/ti-processor-sdk-linux-<machine>-<version>/example-applications/matrix-gui-browser-2.0
gedit Makefile.build
```

In the Makefile.build file you can see that we set a variable called *ENV_SETUP* which points to the environment-setup script. In the *qmake* build target we perform the action "@. ${ENV_SETUP}". That line uses the "@" symbol, which in make says to execute a shell command, and the "." notation, which is another way of sourcing a file, to source the environment-setup script prior to calling the *qmake2* command.

7. To see this work use the following commands to cross-compile the matrix-gui-browser project:

- Close the gedit window if you have not already done so.

```
make -f Makefile.build
```

Notice in the output that the arm-linux-gnueabihf-g++ compiler was called during the build. Also notice that your prompt still shows *sitara@ubuntu* which means that while the environment-setup script was sourced for the build, the effects disappeared after the *make* was finished.

> **NOTE**
>
> If you see a "Nothing to be done for..." output, it means you have previously built this file with a **make** command. Run the command **make clean** first to see the noted output.

8. To verify that the matrix_browser executable that was built was in fact cross-compiled we can use the following command:

   file matrix_browser

   You should see output similar to the following which shows that this program was compiled for ARM.
   matrix_browser: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, not stripped

Using the Top-level Makefile

Description

This section will cover the top-level Makefile found within the root of the Processor Linux SDK. We will discuss how it is used, key variables, and then build some targets. You can find more information at Processor Linux SDK Top-Level Makefile.

Key Points

- Location of top-level Makefile
- Additional settings in Rules.make file
- Determining the available build targets
- Building an individual target

Lab Steps

1. If you do not already have a terminal window open, then open one and change directory to the root of the Processor Linux SDK.

   cd /home/sitara/ti-processor-sdk-linux-<machine>-<version>

2. The top-level Makefile is located in the root of the SDK along with a complementary file called Rules.make. The Rules.make file provides variables and settings used by the top-level Makefile as well as the individual project Makefiles. Open the Rules.make file using the following command:

   gedit Rules.make
   Inside of the Rules.make file, you will find variables that describe the target system such as PLATFORM and ARCH. You will also find variables that point to the root of the SDK install, the location of the linux-devkit directory, and the cross-compiler.

3. Close the gedit session and open a new gedit session for the Makefile.

   gedit Makefile

4. The important items to notice in this window are:
   - This Makefile includes the Rules.make file with the "-include Rules.make" line.
   - The *all* target shows the individual make targets that you can call to build individual components instead of always building everything.
5. How to build an individual component perform the following commands to only build the u-boot and SPL bootloaders.
   - Close the gedit window if you have not already done so.

   make u-boot-spl

   > **NOTE**
   >
   > We do not build all of the targets in this lab due to time constraints, but you could also just type *make* to build all of the components.

Understanding the board-support Directory

Description

This section will teach you about the key components of the board-support directory within the Processor Linux SDK.

Key Points

- Pre-built images are available which have been validated with the SDK
- Linux kernel ships with a configuration file matching the configuration used for the SDK
- There is no longer an x-loader program, instead MLO comes from u-boot
- Additional drivers are located in the extra-drivers directory

Lab Steps

1. If you do not already have a terminal open then open a Linux terminal.
2. Change directory to the board-support directory.

   cd /home/sitara/ti-processor-sdk-linux-<machine>-<version>/board-support

3. List the contents of the directory.

   ls

```
sitara@sitara-Latitude-E6510: ~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support
sitara@sitara-Latitude-E6510:~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support$ ls
extra-drivers                          prebuilt-images
linux-4.4.12+gitAUTOINC+3639bea54a-g3639bea54a  u-boot-2016.05+gitAUTOINC+b4e185a8c3-gb4e185a8c3
sitara@sitara-Latitude-E6510:~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support$
```

4. List the contents of the prebuilt-images directory.

   ls prebuilt-images

```
sitara@sitara-Latitude-E6510: ~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support
sitara@sitara-Latitude-E6510:~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support$ ls prebuilt-image
s/
am335x-boneblack.dtb  am335x-evm.dtb    MLO-am335x-evm             zImage-am335x-evm.bin
am335x-bone.dtb       am335x-evmsk.dtb  u-boot-am335x-evm.img
am335x-bonegreen.dtb  am335x-icev2.dtb  u-boot-spl.bin-am335x-evm
sitara@sitara-Latitude-E6510:~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support$
```

   These images have been pre-compiled and tested to work on the target device. They represent a known good starting point.

5. When building a Linux system it is often helpful to start with a kernel configuration that is known to be working, and then add and remove pieces to optimize the system. In the Processor Linux SDK, we provide the kernel configuration that was used to build the pre-built kernel image inside of the prebuilt-images directory. This configuration can be seen using the command below (replacing the version with the actual kernel version).

   ls linux-<kernel version>/arch/arm/configs/tisdk_*
   You should see a file with the format *tisdk_<machine>_defconfig*. You can use this configuration as the base configuration for your own kernel builds.

6. In the latest SDKs there is no longer a separate code base to generate the MLO bootloader. Instead this bootloader is now part of the u-boot code base and is generated using the second program loader (SPL) functionality of u-boot. In the previous Lab you compiled u-boot using the *make u-boot* command. If you look at the contents of the u-boot directory using the command below you will notice that there exists both the MLO file as well as the u-boot.img file. These were both built during the u-boot build operation.

   ls u-boot-<version>

```
sitara@sitara-Latitude-E6510: ~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support
cmd     doc        include   MAINTAINERS  post      System.map       u-boot.bin    u-boot.img      u-boot.sym
common  drivers    Kbuild    MAKEALL      README    test             u-boot.cfg    u-boot.lds
sitara@sitara-Latitude-E6510:~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support$ ls u-boot-2016.05+gi
tAUTOINC+b4e185a8c3-gb4e185a8c3/
api         configs   fs         MAINTAINERS  README           TISDK-README  u-boot-dtb.bin    u-boot.srec
arch        disk      include    MAKEALL      scripts          tools         u-boot-dtb.img    u-boot.sym
board       doc       Kbuild     Makefile     snapshot.commit  u-boot        u-boot.img
cmd         drivers   Kconfig    MLO          spl              u-boot.bin    u-boot.lds
common      dts       lib        net          System.map       u-boot.cfg    u-boot.map
config.mk   examples  Licenses   post         test             u-boot.dtb    u-boot-nodtb.bin
sitara@sitara-Latitude-E6510:~/ti-processor-sdk-linux-am335x-evm-03.00.00.04/board-support$
```

   In the above image the MLO and u-boot.img files were highlighted. Both of these files were created during the single u-boot build operation. The benefit is that you are now able to share the same u-boot code to build a initial boot loader as well as to build the full featured u-boot boot loader.

7. For devices with out of tree modules those device drivers are located in the *extra-drivers* directory. This is done to group items that require rebuilding when the kernel configuration changes into a single directory, the *board-support* directory. You can see the out of tree drivers using the command below:

   ls extra-drivers

Preparing to Develop using the setup.sh Script

Description

This section will cover how to use the setup.sh script inside of the Processor Linux SDK to configure your board for development with TFTP and NFS. Due to time limitations some operations will be skipped since they will have already been done for you. These operations that have been done already will be called out in the section below. For more information see Processor SDK Linux Setup Script.

Key Points

- Installation of required software to setup development environment
- Creation of NFS file shares
- Setup of TFTP for kernel transfers
- Configuring the EVM using minicom scripts

Lab Steps

The following steps will help you validate that you have a valid serial connection with your target board.

> **IMPORTANT**
>
> These steps are written for boards like the EVM-SK and Beaglebone which use an FTDI USB-to-Serial adapter. For boards with a straight UART connection you can ignore the steps for /dev/ttyUSBx and instead just use the serial port on your Linux host that is connected to the board.

1. Connect the USB cable between your Linux host and your target board

> **IMPORTANT**
>
> On the AM335x EVM-SK, the USB cable enables a USB-to-serial interface to the Linux host PC. If using a virtual machine, please ensure the AM335x EVM-SK is a selected USB device.
>
> If you are running your VMWare image with Windows 7, some USB-to-Serial adapters do not properly work with Windows 7. Make sure that your adapter specifically says that it supports Windows 7. You may also need to install an additional driver for your adapter.

2. Open a terminal and run the following command to find the USB serial adapters available on the system. The FTDI chip used on the board presents two serial interfaces to the Linux host. The first interface is the JTAG interface and the second one is the serial console interface.

   ls /dev/ttyUSB*
   You should see output like:

   ```
   /dev/ttyUSB0
   /dev/ttyUSB1
   ```

   > **NOTE**
   >
   > If using VMWare, you may need to pass the "Future Technology Devices" device into the VMWare image

3. Since the board's serial interface is the second interface, you will want to open a serial console to the second device node. In this case that is **/dev/ttyUSB1**. This can be done by doing:

   minicom -w -s
   Select Serial port setup and press ENTER
   Press A to modify the Serial Device and change the device to /dev/ttyUSB1. Press ENTER
   Press ENTER to exit the serial setup
   Select Exit and press ENTER

4. You should now see a minicom window.
5. Power on the board. Depending on your board, you may see a boot sequence or simply a "ccccc" being printed in the minicom window. This means the serial connection is working as expected. You may leave this console RUNNING in the background.

   > **NOTE**
   >
   > For the Beaglebone the board will power on as soon as the USB cable is connected. You can reset the board by pressing the reset button which should provide you with the expected output.

6. If you have not already done so, close any open minicom windows using **CTRL+A** then **Z**, then press **X** and select **Yes** to leave the minicom.
7. In a terminal window execute the following commands:

   cd /home/sitara/ti-processor-sdk-linux-<machine>-<version>
   ./setup.sh

- You will be warned that this script requires administrator priveledges (sudo access) to run. This is required to allow installation of packages and system configuration for NFS and TFTP
  - Press **ENTER** to continue
- If you are prompted for the *[sudo] password for sitara:* then use password **sitara**. If you have a different user account, use that user account password.

  > **NOTE**
  >
  > You will see the system attempt to update the host software. This step has already been done for you so that access to an internet connection is not required. You can review the list to see which host packages which were previously installed using setup.sh.

- You will now be prompted for the directory to install the target file system. The default should look like */home/sitara/ti-processor-sdk-linux-<machine>-<version>/targetNFS*.
  - Press **ENTER** to take this default.
  - Press **ENTER** when warned that sudo permissions are required.
- To save extraction time for the root file system the file system may have already been extracted for you. In this case you will be asked if you was to rename the existing file system, overwrite the file system, or skip the file system extraction.
  - Select **s** and press **ENTER** to skip the extraction since the file system has been extracted already.
- Next the setup script will inform you that a default path for binary installation has been configured in the Rules.make file.
  - Press **ENTER** to continue.
- You will now be warned that sudo access is required to add the NFS file system that was just created to the /etc/exports file. This is so it can be used as a root file system for the target device to boot over NFS.
  - Press **ENTER**.
- You will see the system stop and then start the NFS server to that the file system is exported.
- You will now be prompted about which directory to make your TFTP root directory.
  - Press **ENTER** to take the default of */tftpboot*
- You will be warned that sudo access is required to configure the TFTP server.
  - Press **ENTER**.
- You will be told that the uImage file already exists. This was pre-copied to save time.
  - Press **s** to skip copying this image again.
- You will see the system setup stop and start the xinetd (TFTP server).
- You will now be prompted to enter the serial port for the board so that minicom can connect to the board. Since you are using a board with a built in USB-to-Serial adapter you do not need to enter the serial port as it will be detected later.
  - Press **ENTER** to continue.
- You will now be prompted for your host IP. This is required to configure u-boot to boot images from your host system. You should take the default IP address.
  - Press **ENTER**.
- Now you will select where to obtain the Linux kernel from. In this lab we will be configuring to boot from TFTP.
  - Type **1** and press **ENTER**.

- The next prompt will ask you where the root file system should come from. In this lab we will use NFS.
  - Type **1** and press **ENTER**.
- You will now be presented with the list of kernel images available in the /tftpboot directory. You should use the default image for your TFTP image of *uImage-am335x-evm.bin*
  - Press **ENTER**.
- You will notified that a Beaglebone/EVM-SK has been detected. Select yes to configure for the EVM-SK .
  - Type **y** and press **ENTER**.
- You will be asked if you would like to reboot the board with these new settings.
  - Press **ENTER** to reboot the board.

    > **IMPORTANT**
    >
    > This script expects the board to NOT be logged in. If you have logged in the board, the reboot will fail. In this case, you can reboot the board manually.

8. Your target device will now be booting the Linux kernel from the TFTP server on your host machine with an NFS root file system from the targetNFS directory inside of your SDK installation.
   - You can verify that your device is booted from NFS using the command below:
     1. Press **ENTER** to see the Arago Project prompt and login using **root**
     2. **cat /proc/cmdline** you should see the output list the root as /dev/nfs and the nfsroot should be configured to the NFS share you just created.

        > **NOTE**
        >
        > You can also use the **mount** command to verify that your root partition was mounted over NFS.

9. You system is now configured. You can always re-run the setup.sh script to change your board configuration.

   > **IMPORTANT**
   >
   > If you want to reset your u-boot environment back to the default you can do the following in minicom
   >
   > 1. Login to the board using **root**
   > 2. Reboot the board using **init 6**
   > 3. When asked to **Hit any key to stop autoboot** stop the boot process to enter the u-boot prompt.
   > 4. Reset the default boot environment using **env default -f -a**
   > 5. You can now boot the board with the default boot arguments using the **boot** command.
   >
   > These changes will be lost on the next boot. To permanently make changes you should either delete the uEnv.txt file or re-run the setup.sh script.

Working with the Example Applications

Description

This lab will show you how to work with the example applications using the Linux command line. We will also explore some of the techniques used in the example applications to make cross compiling easier.

Key Points

- Example applications have CCS/Eclipse project files
- Building example applications from the command line
- Verifying the executable was built for ARM

Lab Steps

1. It is possible to rebuild the individual example applications using Makefiles. For example to rebuild the arm-benchmarks example application, open a Linux terminal and navigate to:

   cd /home/sitara/ti-processor-sdk-linux-<machine>-<version>/example-applications/arm-benchmarks-<version>

2. Then you can use the following commands:

   make clean
   make

3. After the build has finished you can verify that the executables were build for the ARM device using the *file* command.

   file dhrystone/Release/dhrystone
   As we saw earlier the output lists that the dhrystone executable was for an ARM device:

   dhrystone/Release/dhrystone: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, not stripped

4. The individual benchmarks also have CCS/Eclipse project files in them when can be used to build these projects using CCS. To see these project files do:

   cd dhrystone
   ls -la

5. You should see files like *.cproject* and *.project* listed. These files allow building the dhrystone project using CCS.

Importing Example Applications into CCS

Description

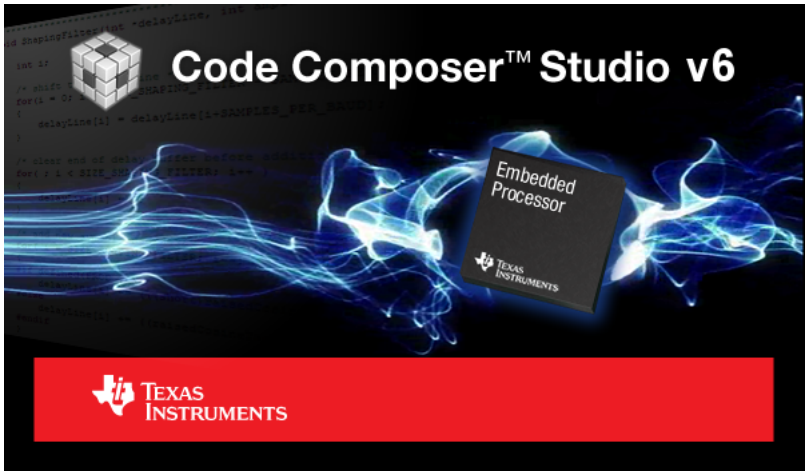
This section will cover how you can use the CCS/Eclipse project files seen in the previous section to import the example application projects into CCS and build them. For more information see the Code Composer Studio (http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v6) page.

Key Points

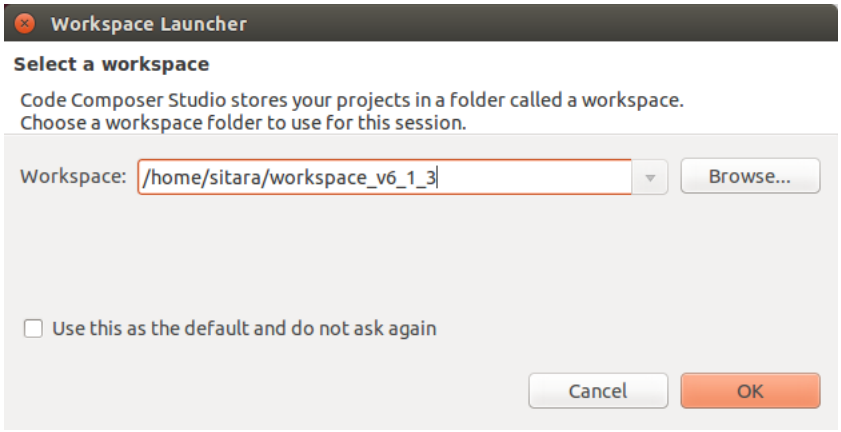
- Example applications have existing CCS/Eclipse project files that can be imported.
- CCSv6 can be used to compile the example applications

Lab Steps

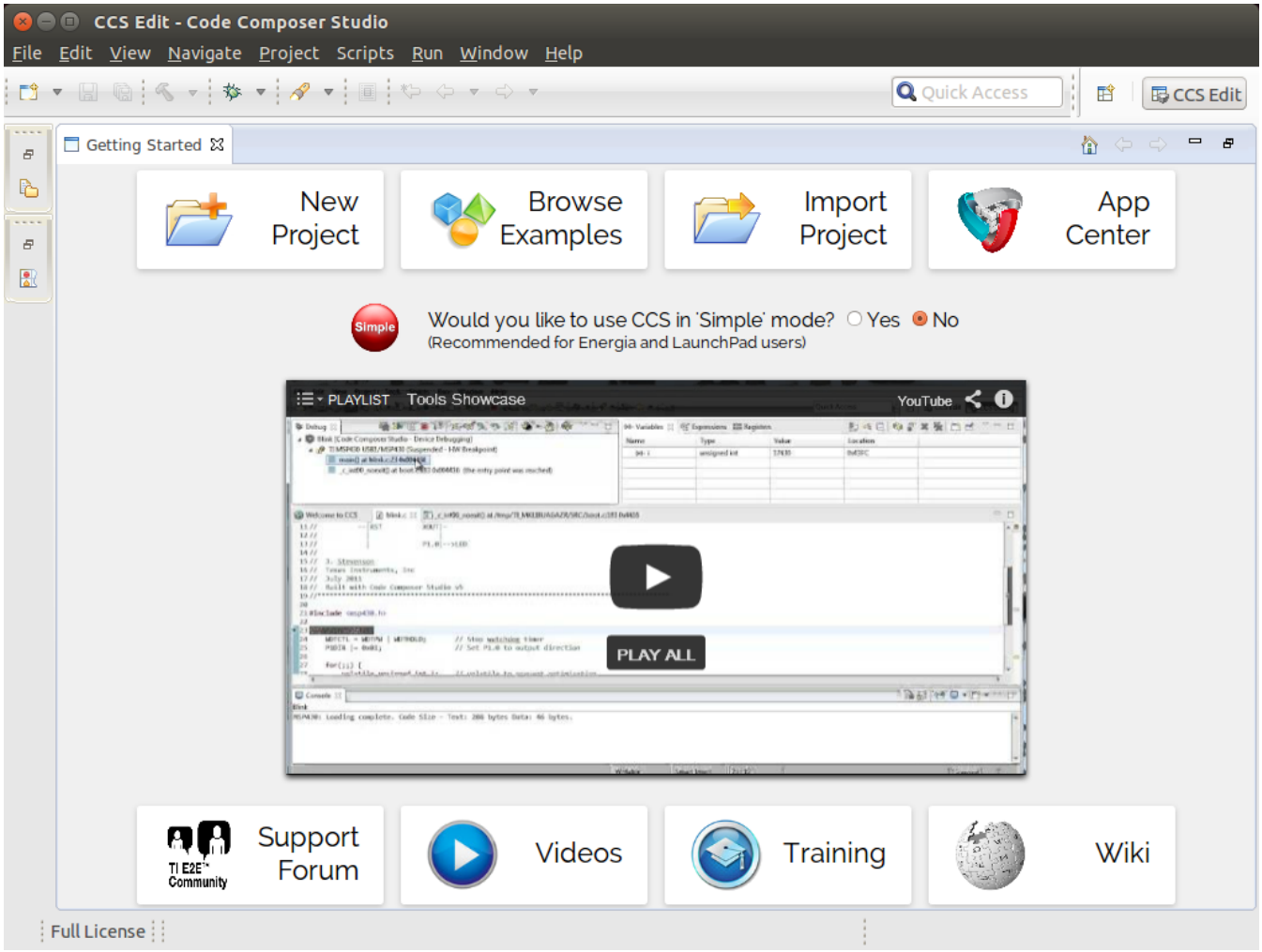
1. Launch CCSv6 using the following steps:
   - **Double-Click the Code Composer Studio v6 icon on the desktop**. You will see a splash screen appear while CCS loads.



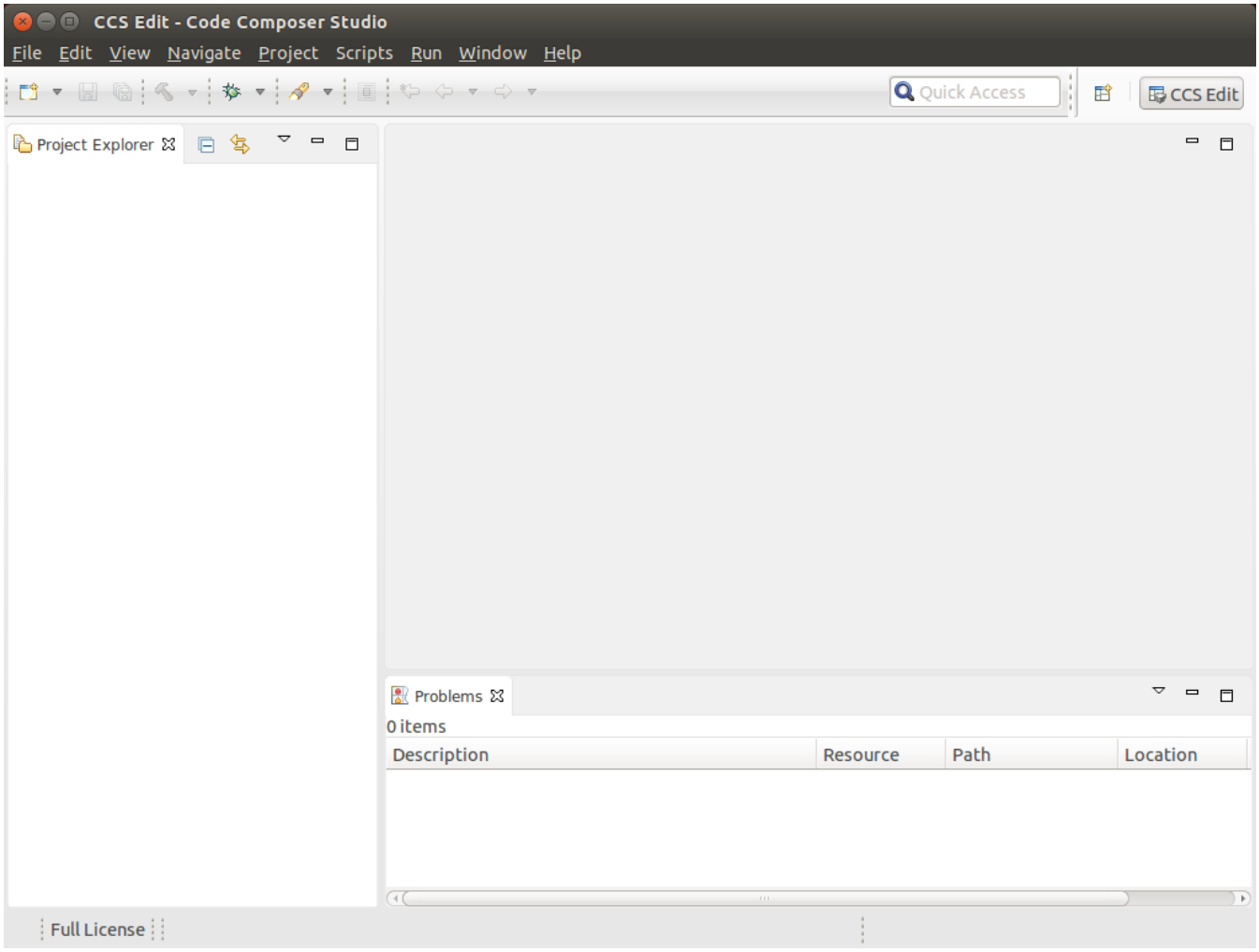
   - The next window will be the *Workspace Launcher* window which will ask you where you want to locate your CCSv6 workspace. Use the **default** value.



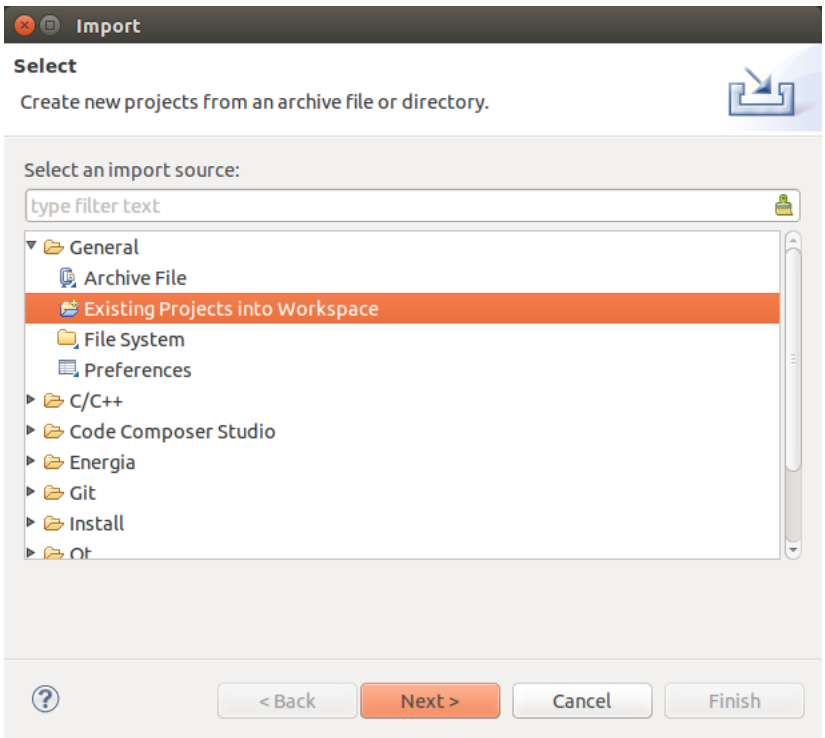
   - CCS will load the workspace and then launch to the default *TI Resource Explorer* screen.



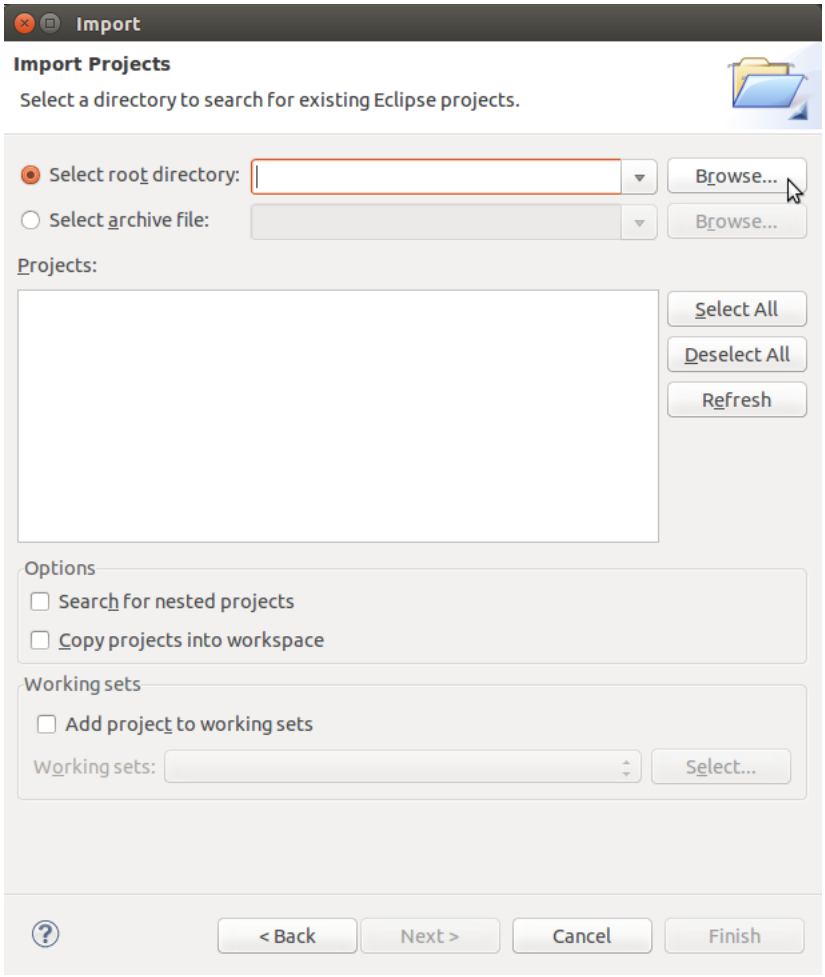
- **Close the TI Resource Explorer screen**. This screen is useful when making TI CCS projects which use TI tools. The Processor Linux SDK uses open source tools with the standard Eclipse features and therefore does not use the TI Resource Explorer. You will be left in the Project Explorer default view.



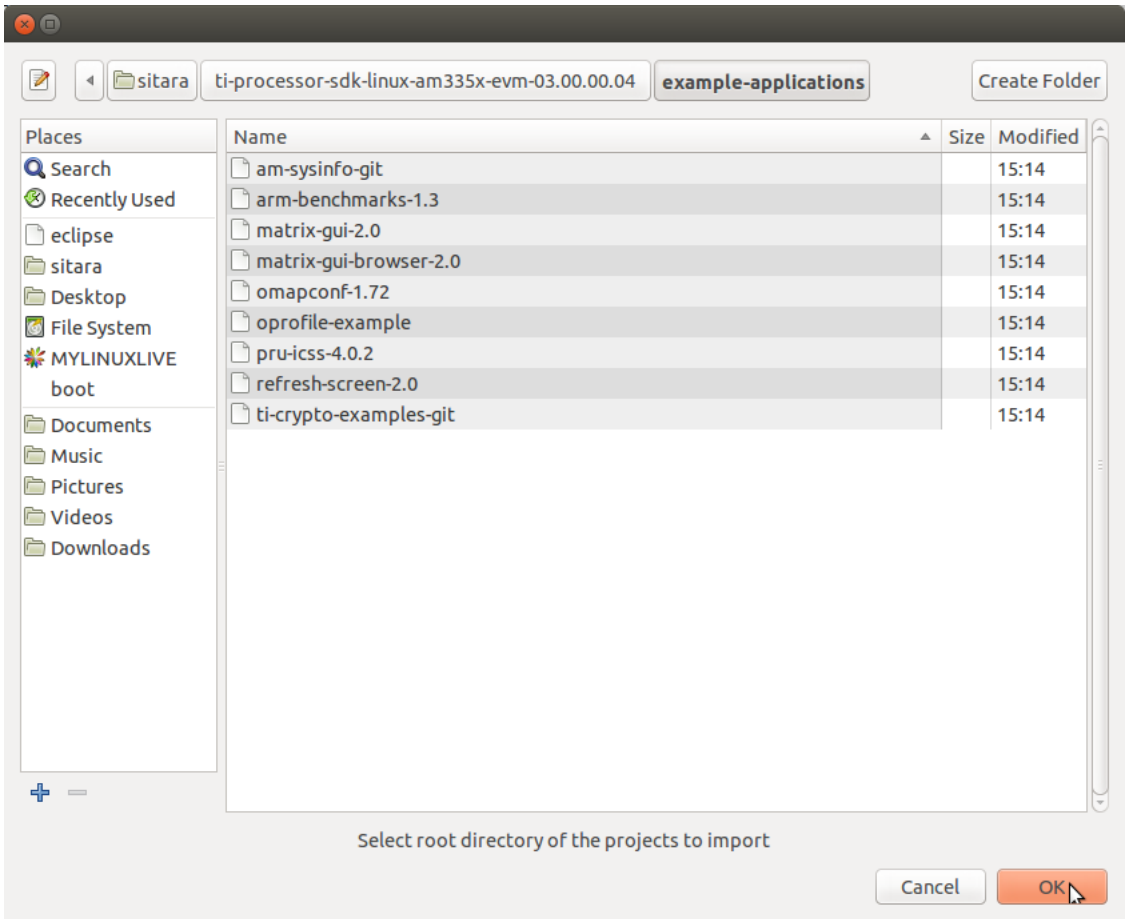
2. From the main CCSv6 window, select **File -> Import...** menu item to open the import dialog.
3. Select the **General -> Existing Projects into Workspace** option.



4. Click **Next**.
5. On the *Import Projects* page click **Browse**.



6. In the file browser window that is opened navigate to the **<SDK INSTALL DIR>/example-applications** directory and click **OK**.

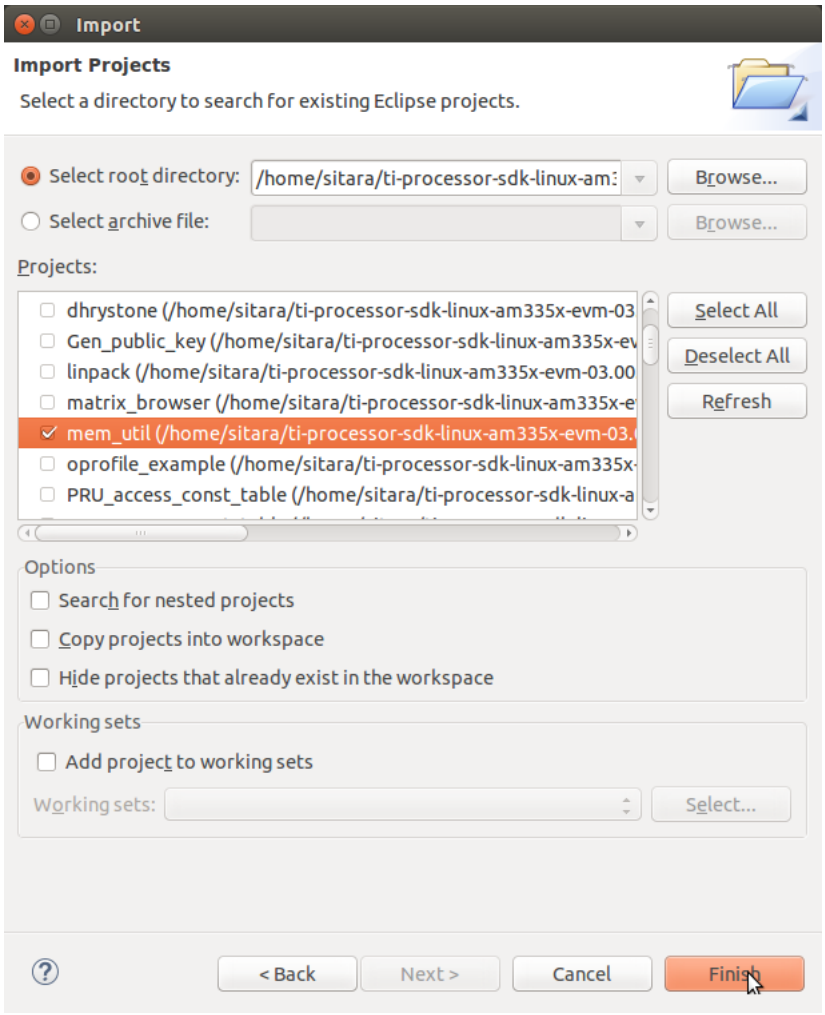


7. The *Projects:* list will now be populated with the projects found.

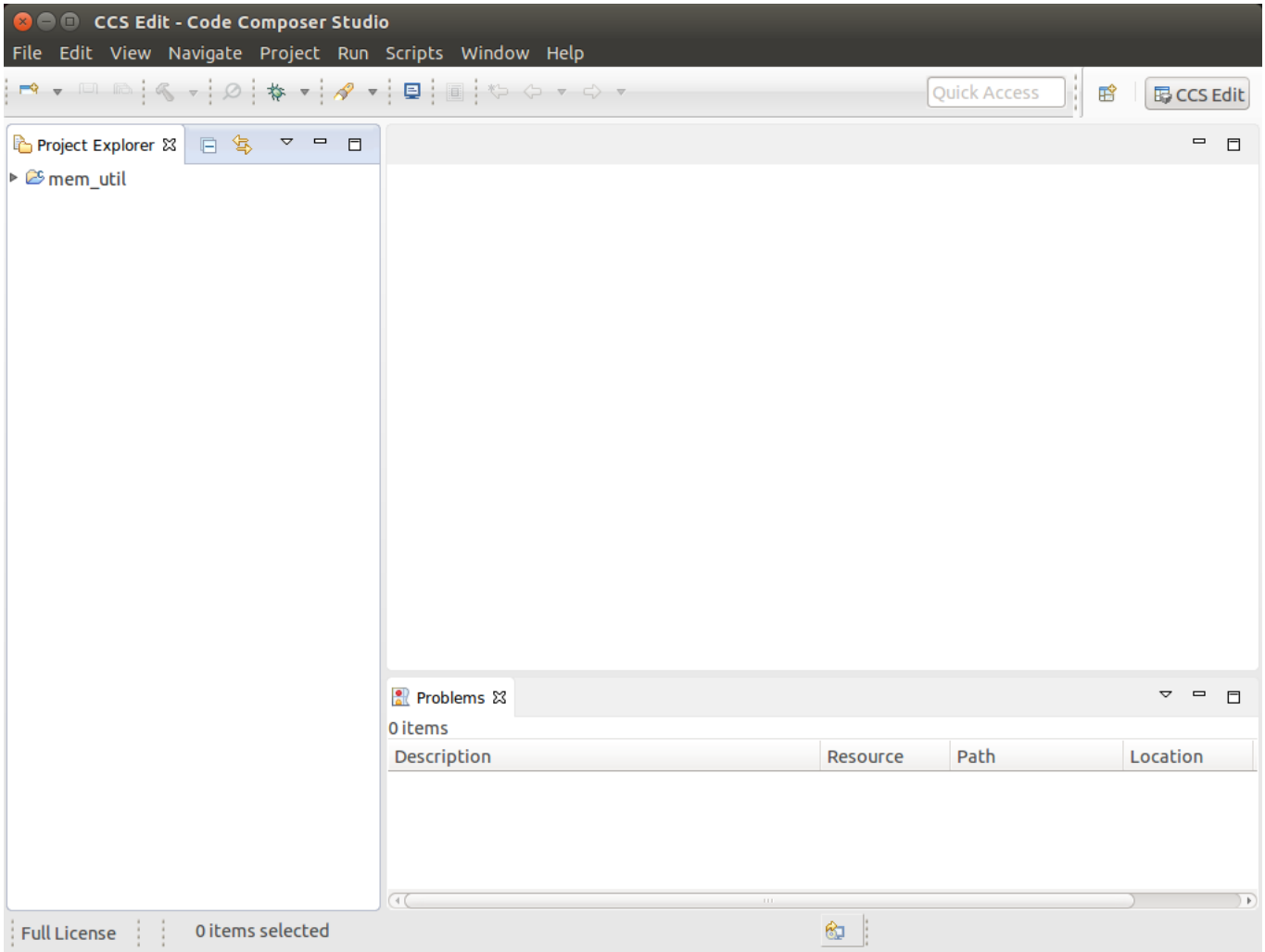
> **NOTE**
>
> Uncheck the following projects. They are Qt projects and are imported using a different method. For more information, see the Hands on with QT training.

- matrix_browser
- refresh_screen

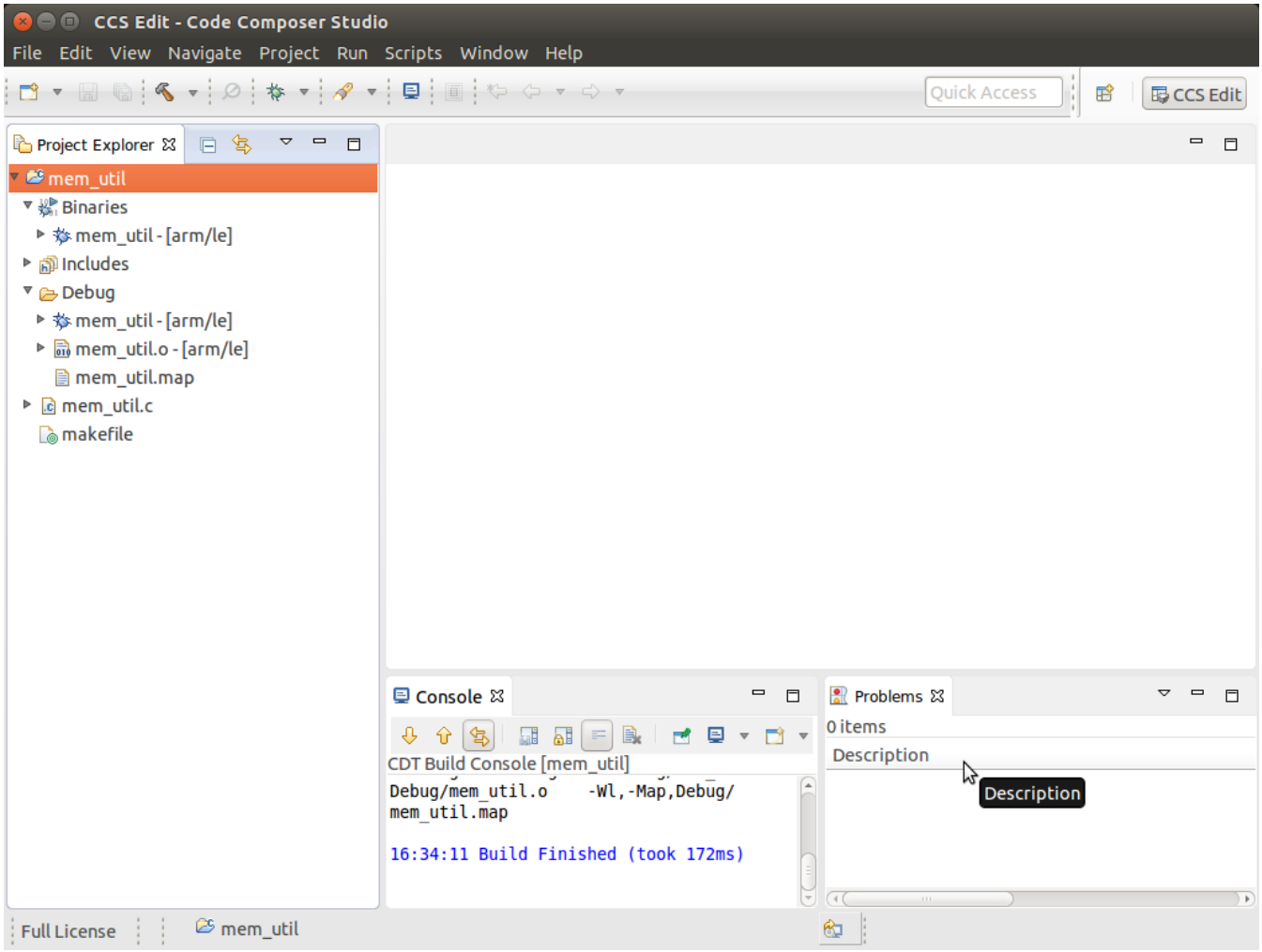
8. Select the projects you want to import. The following screen capture shows importing all of the example projects for an ARM-Cortex device, excluding the matrix_browser project.



9. Click **Finish** to import all of the selected projects.
10. You can now see all of the projects listed in the *Project Explorer* tab.



11. In order to build one of the projects use the following steps. For this example we will use the *mem-util* project.
    - Right-Click on the **mem-util project** in the *Project Explorer*.
    - Select the build configuration you want to use.
        - For Release builds: **Build Configurations -> Set Active -> Release**
        - For Debug builds: **Build Configurations -> Set Active -> Debug**
    - Select **Project -> Build Project** to build the highlighted project.
    - Expand the mem-util project and look at the mem_util.elf file in the Debug or Release directory (depending on which build configuration you used). You should see the file marked as an [arm/le] file which means it was compiled for the ARM.



> **NOTE**
>
> You can use **Project -> Build All** to build all of the projects in the *Project Explorer*.

Creating a New CCS Project

Description

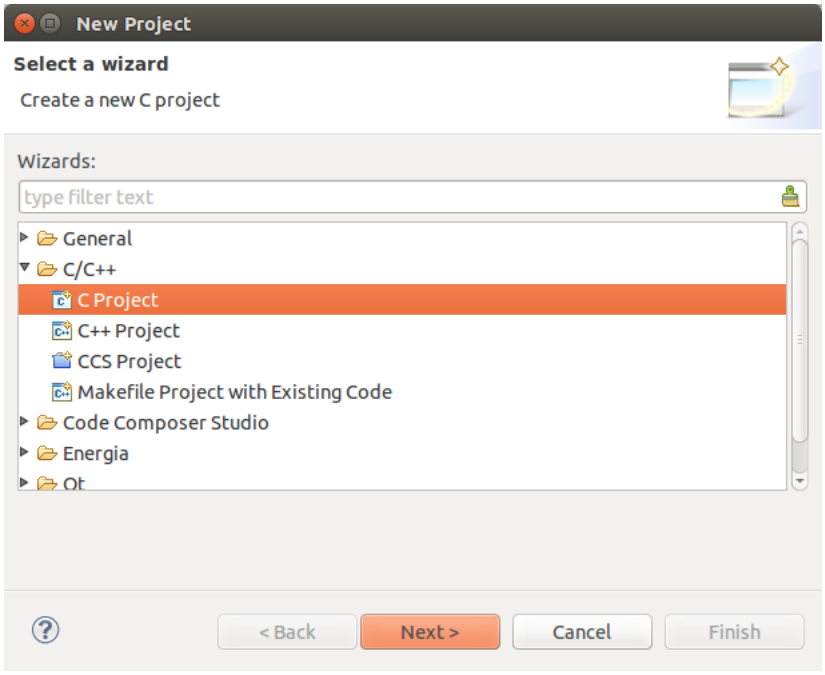
This section will cover how to make a new project in CCS and cross-compile it for the ARM device. For more information see Code Composer Studio v5 Users Guide (http://processors.wiki.ti.com/index.php/Code_Composer_Studio_v5_Users_Guide).

Key Points

- How to create a new cross-compile project
- How to add sources to the project and build them

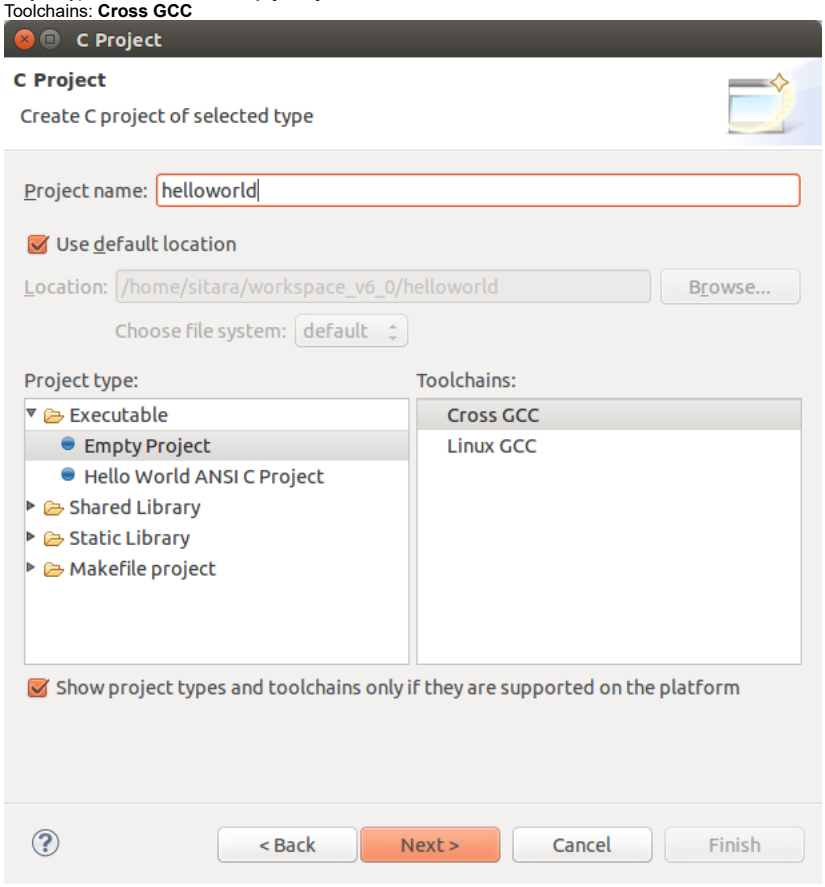
Lab Steps

1. From the main CCSv6 window, select **File -> New -> Project...** menu item.
2. In the *Select a wizard* window, select the **C/C++ -> C Project** wizard.

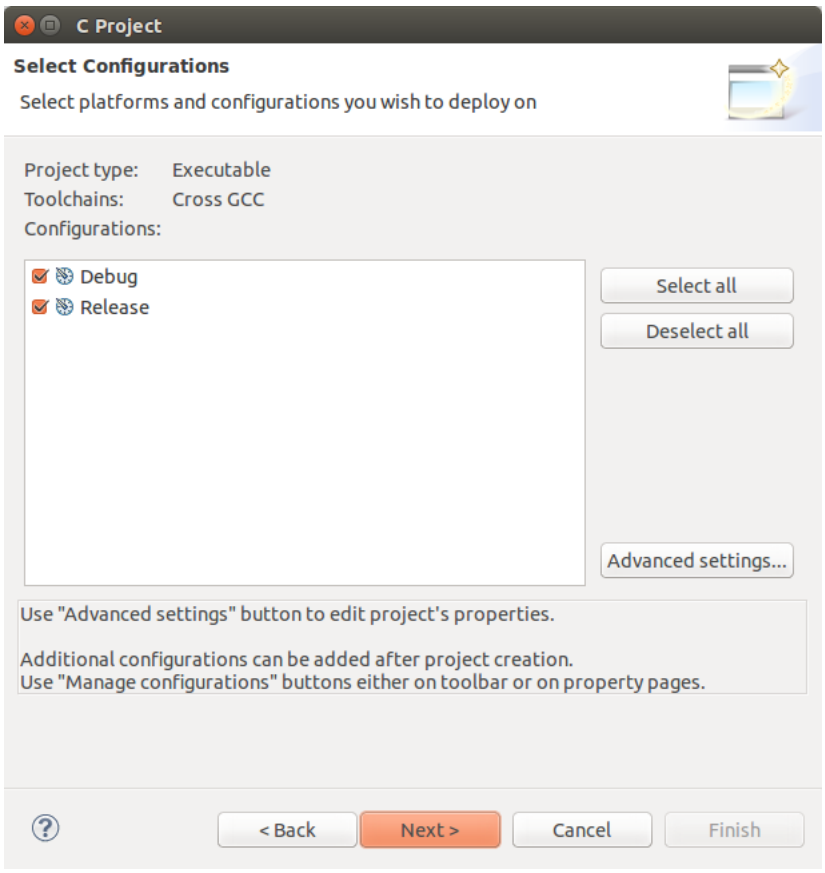


3. Click **Next**.
4. In the *C Project* dialog set the following values:

Project Name: **helloworld**
Project type: **Executable -> Empty Project**
Toolchains: **Cross GCC**



5. Click **Next**.
6. In the *Select Configurations* dialog, you can take the default *Debug* and *Release* configurations or add/remove more if you want.

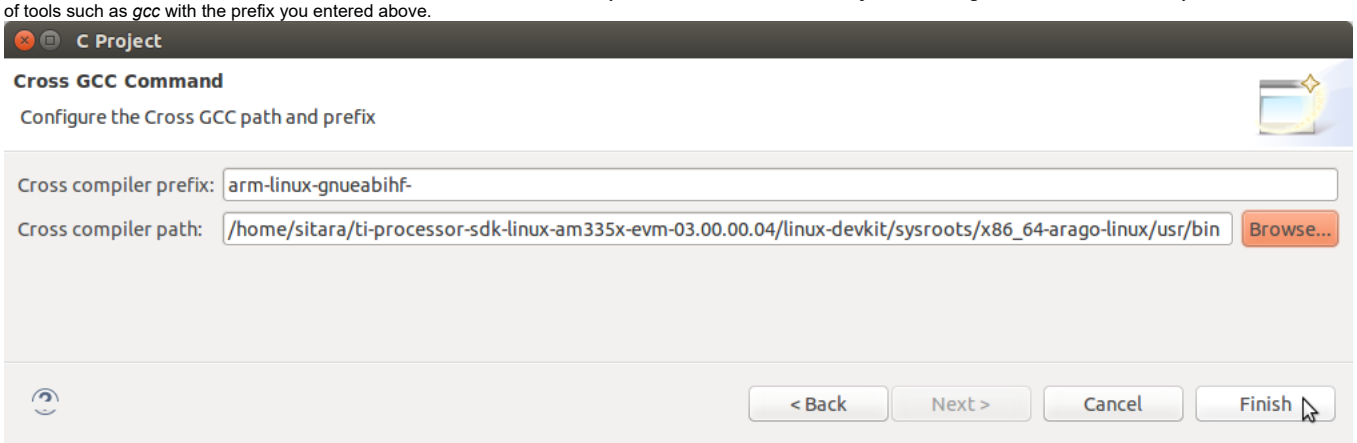


7. Click **Next**.
8. In the *Command* dialog set the following values:

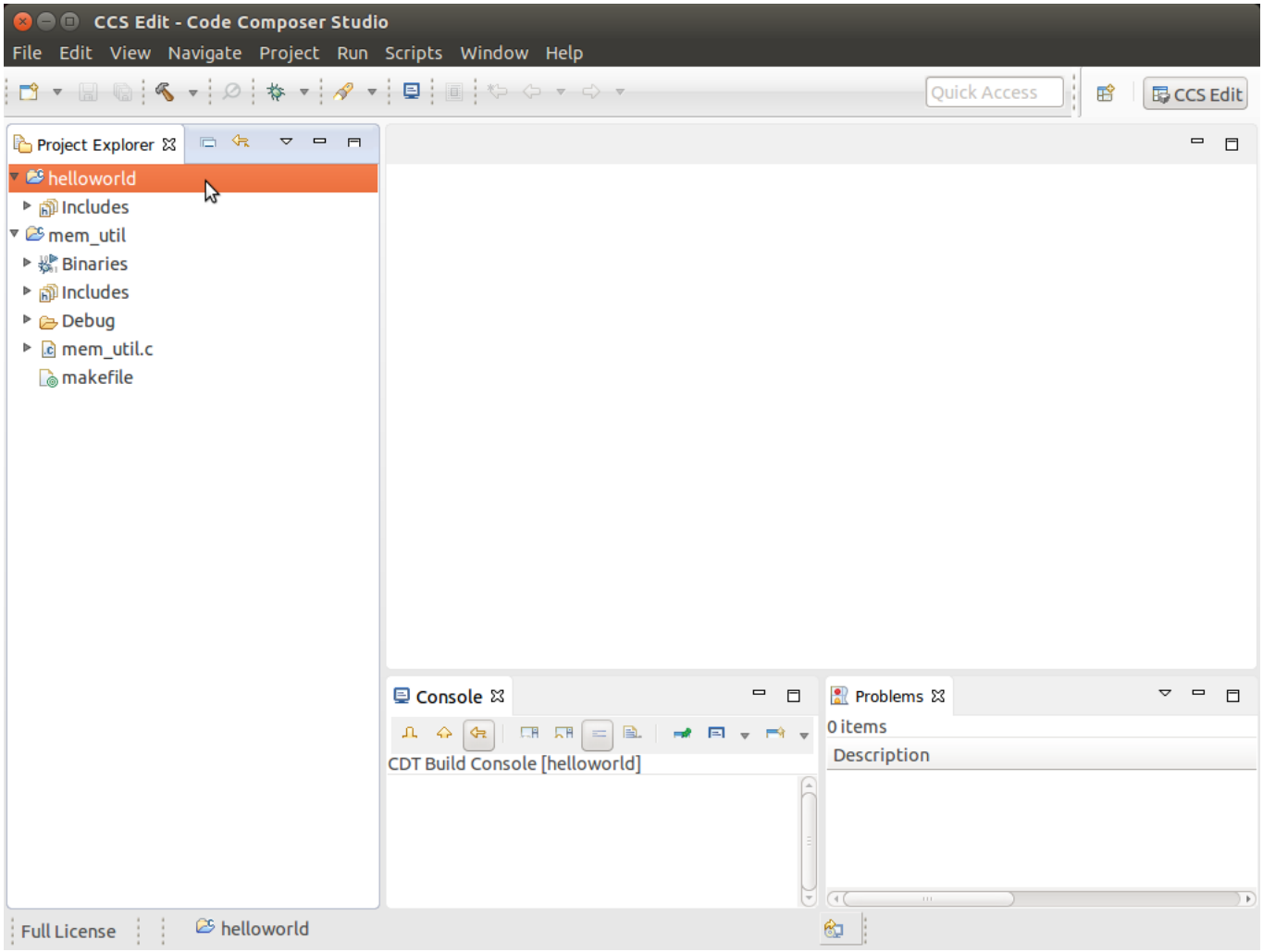
   Tool command prefix: **arm-linux-gnueabihf-**.

   > **NOTE**
   >
   > Note the the prefix ends with a "-". This is the prefix of the cross-compiler tools as will be seen when setting the *Tool command path*.

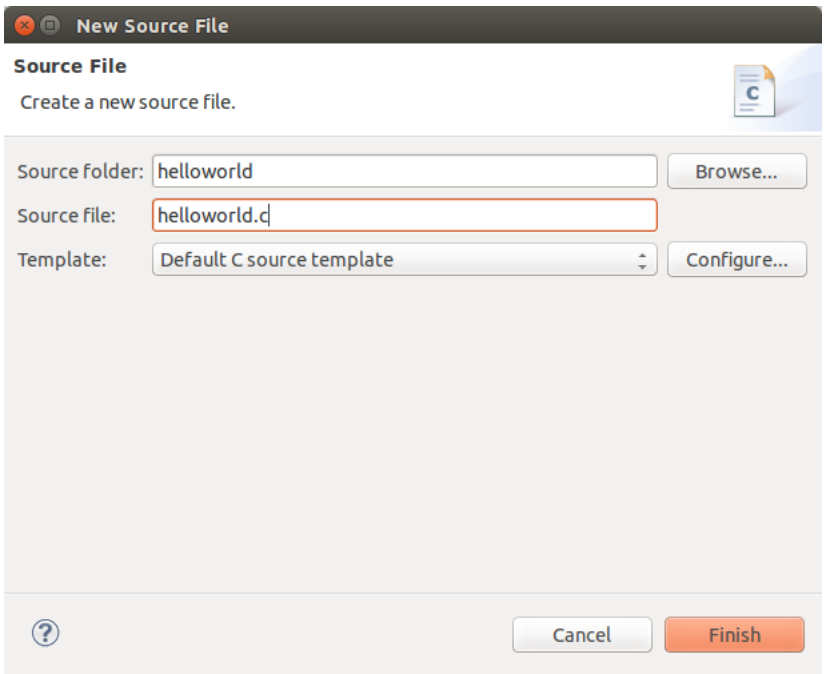
   Tool command path: **/home/sitara/ti-processor-sdk-linux-<machine>-<version>/linux-devkit/sysroots/<Arago Linux>/usr/bin**
   Use the *Browse.*. button to browse to the Sitra Linux SDK installation directory and then to the **linux-devkit/sysroots/<Arago Linux>/usr/bin** directory. You should see a list of tools such as *gcc* with the prefix you entered above.



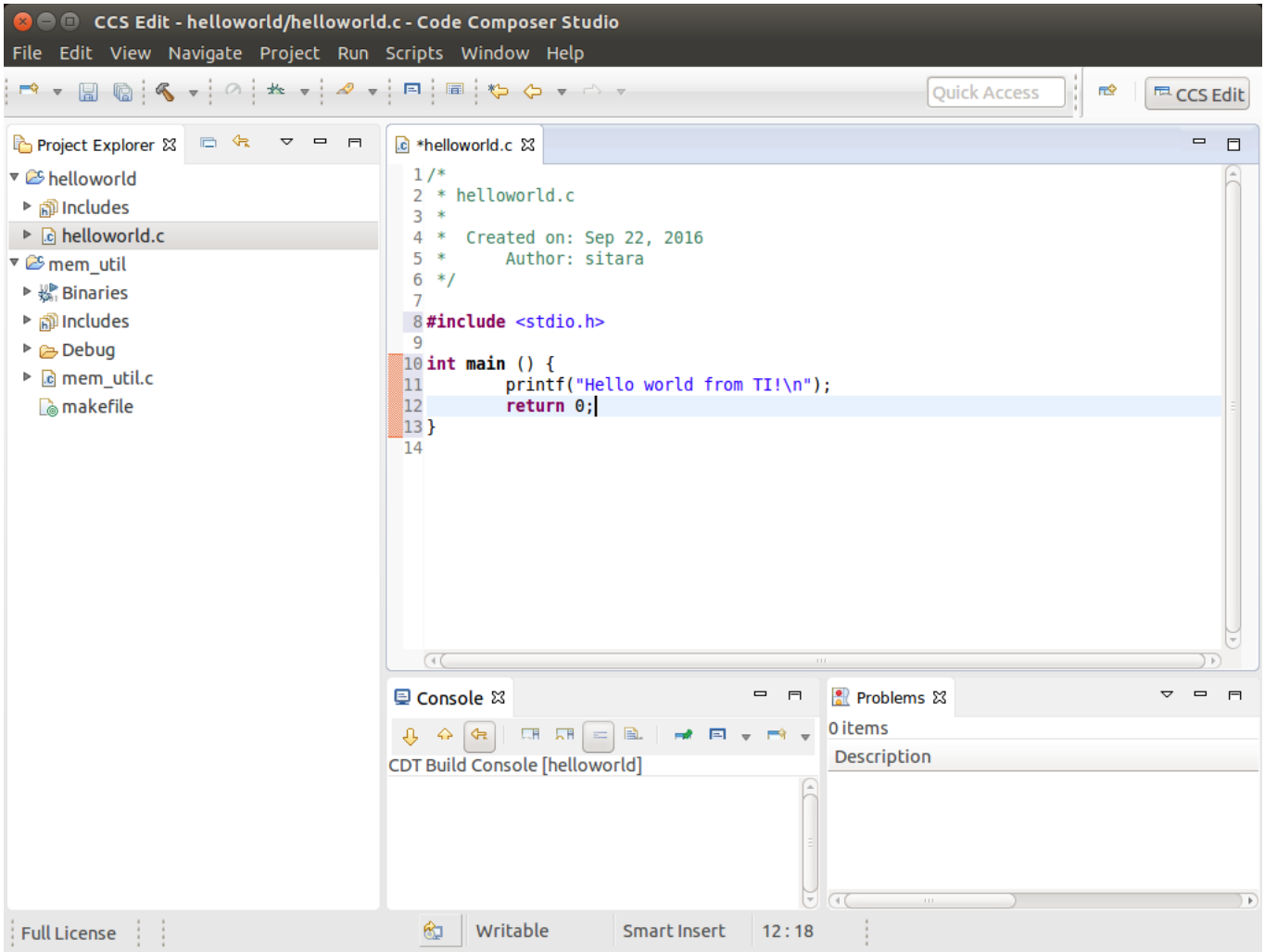
9. Click **Finish**.
10. After completing the steps above you should now have a *helloworld* project in your CCS *Project Explorer* window, but the project has no sources.



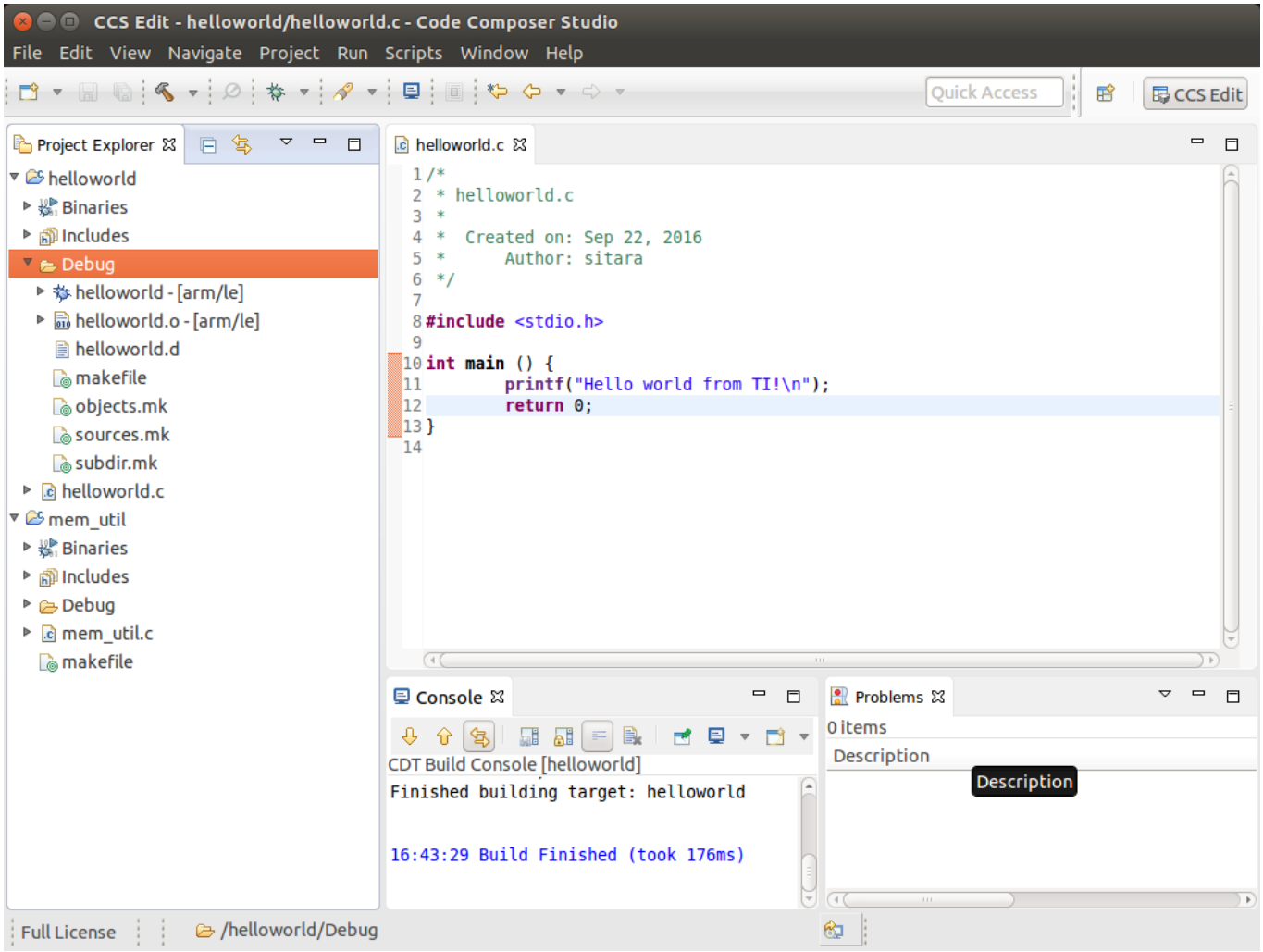
11. From the main CCSv6 window select **File -> New> Source File** menu item.
12. In the *Source File* dialog set the *Source file:* setting to **helloworld.c**



13. Click **Finish**.
14. After completing the steps above you will have a template *helloworld.c* file. Add your code to this file like the image below:



15. Compile the *helloworld* project by selecting **Project -> Build Project**
16. The resulting executable can be found in the *Debug* directory.



Configuring Remote System Explorer (RSE)

Description

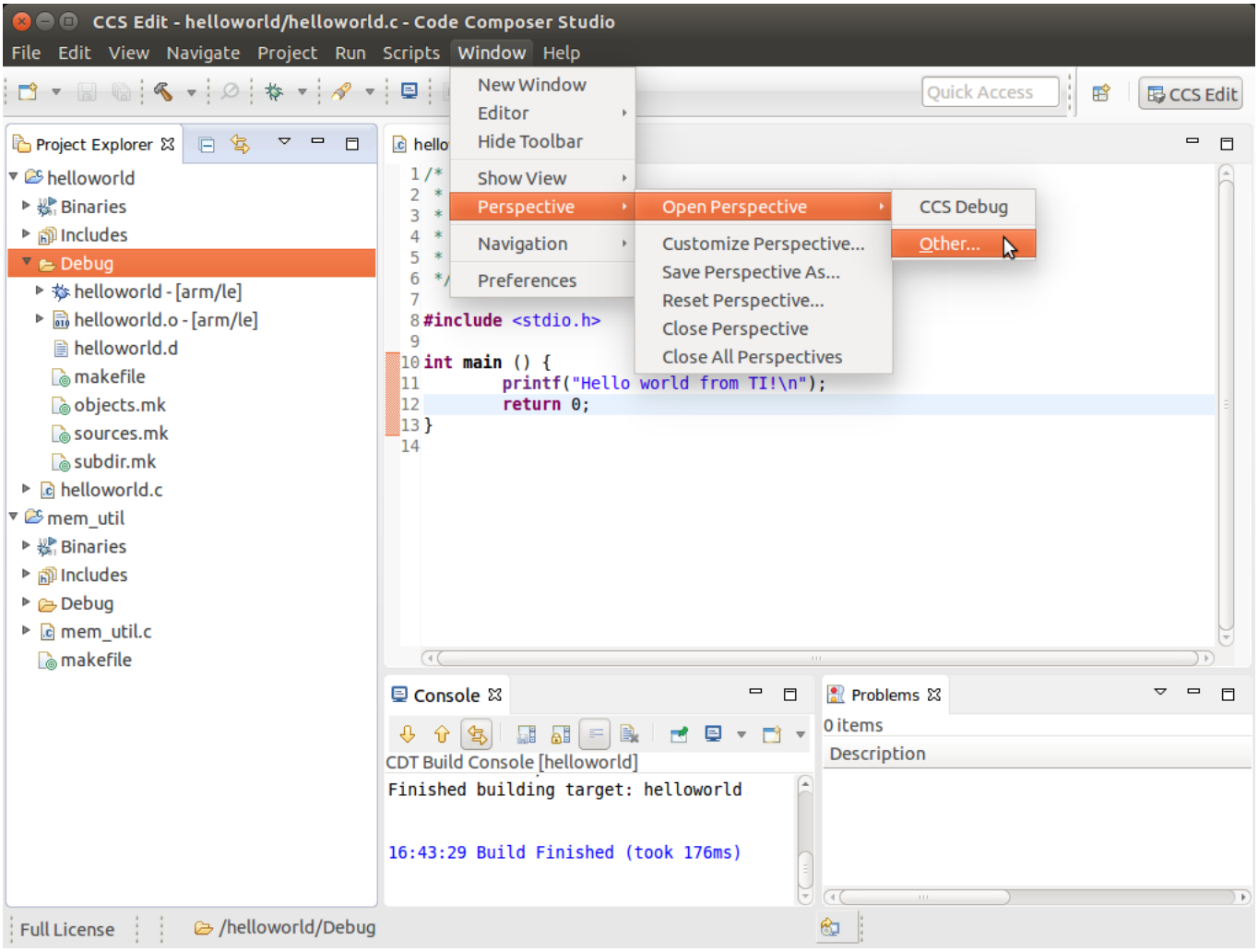
Now that we have compiled the helloworld project it would be useful to execute the built results on the EVM. Since we are still booted from NFS we could just copy the helloworld executable to the NFS file system, but instead you will learn how to use remote system explorer (RSE) to transfer the file. The actual transfer steps are covered in the next section, but before the transfer can be done RSE must be configured first. For more information, see http://processors.wiki.ti.com/index.php/How_to_setup_Remote_System_Explorer_plug-in (http://processors.wiki.ti.com/index.php/How_to_setup_Remote_System_Explorer_plug-in)

Key Points

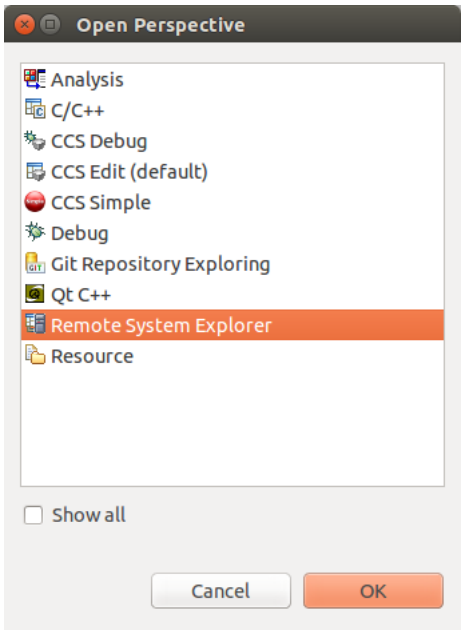
- Creating a new RSE connection with the target device

Lab Steps

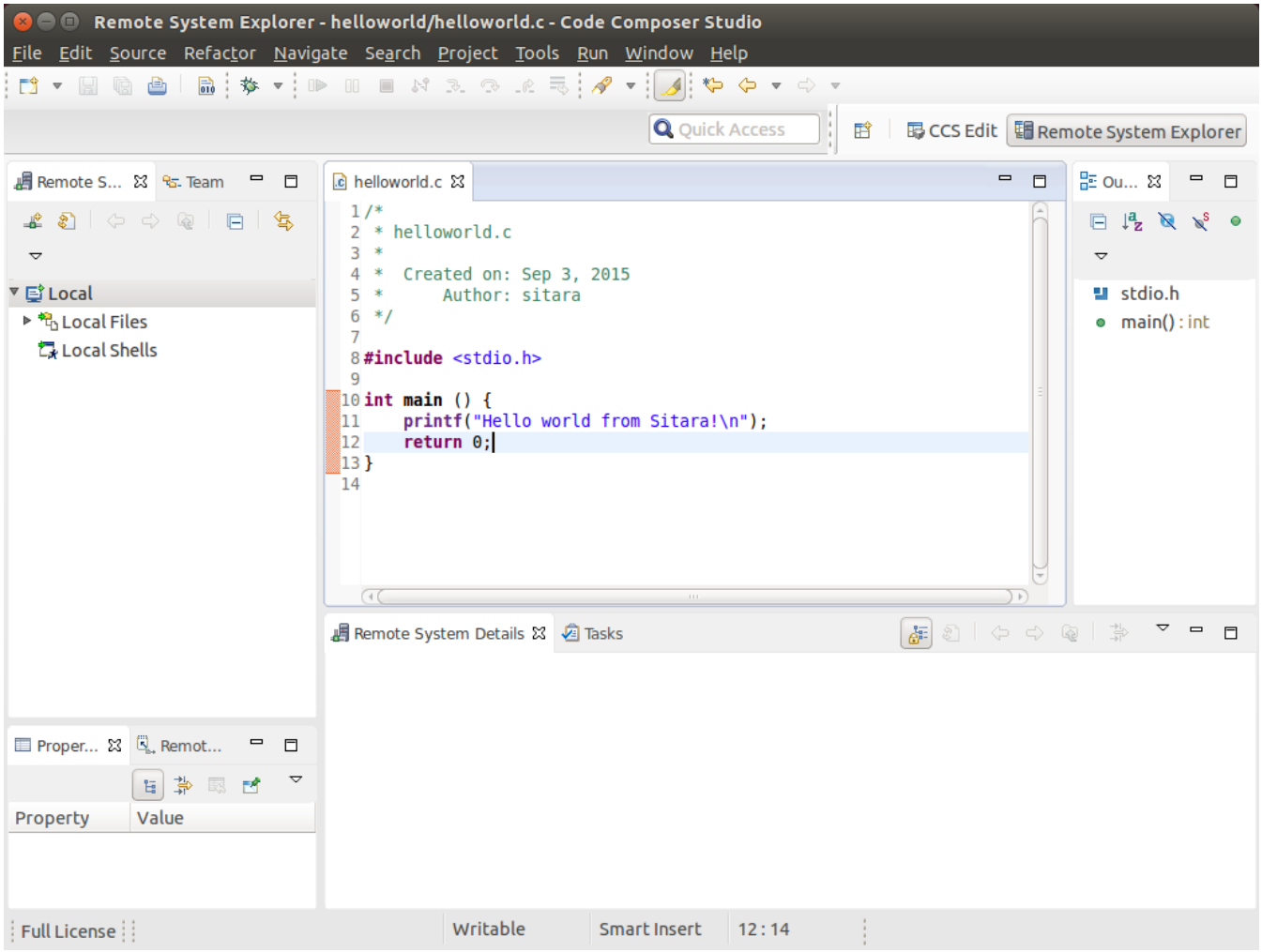
1. Open the RSE perspective.
   - Go to **Window -> Perspective -> Open Perspective -> Other...**



- In the menu window select **Remote System Explorer** to open this perspective.

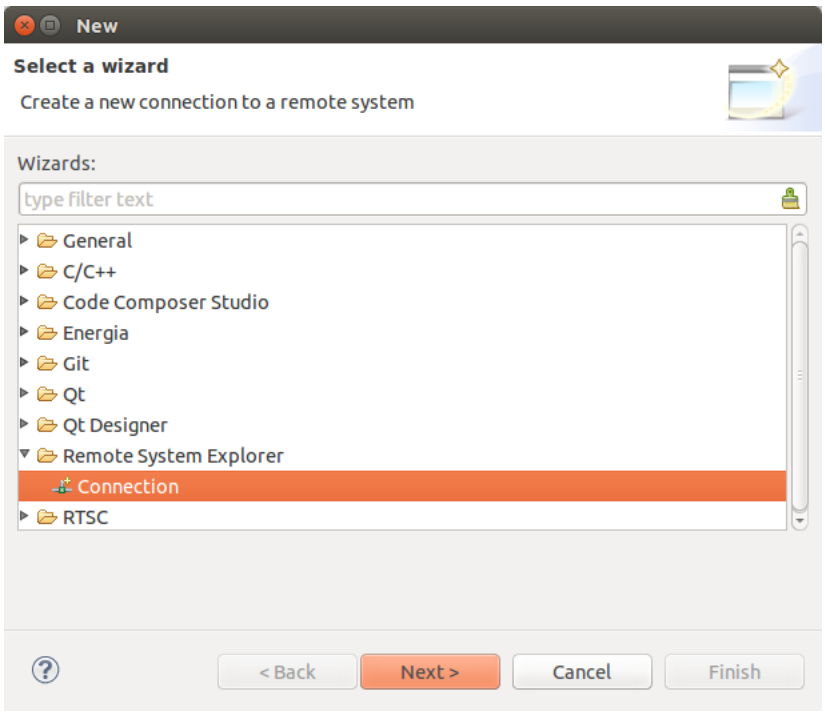


- Click **OK**.
- You will now have the RSE view opened.

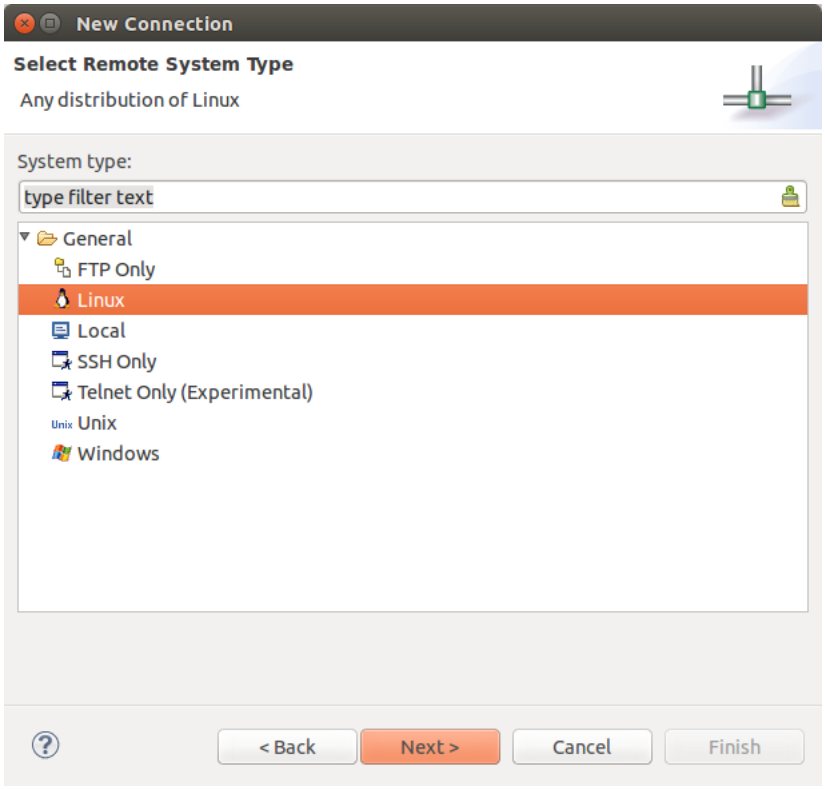


2. The following steps will create a new RSE connection to your target device.
3. Click **File -> New -> Other...**

- In the **Select a wizard** window select **Remote System Explorer -> Connection**.

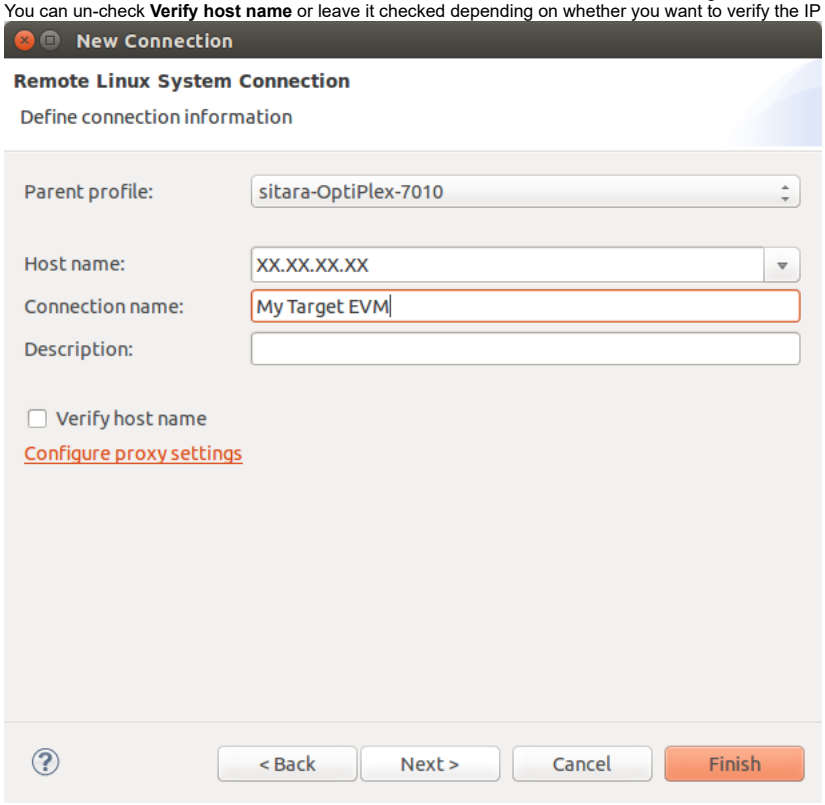


- Click **Next**.
- In the **Select Remote System Type** window select the **Linux** system type.

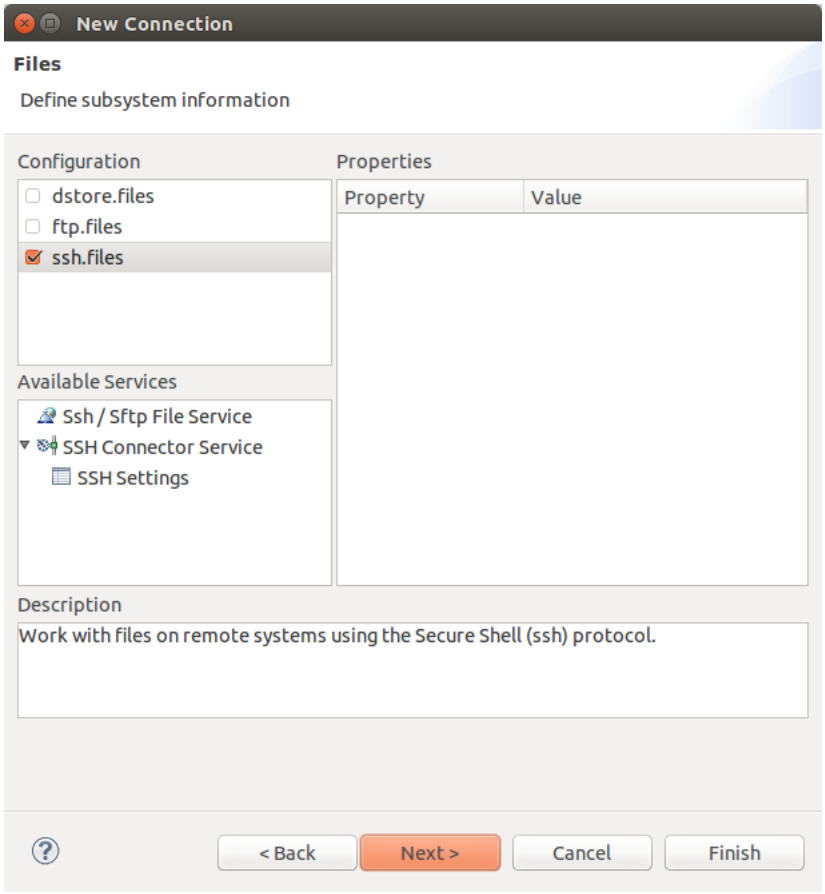


- Click **Next**.
- In the **Remote Linux System Connection** window enter.

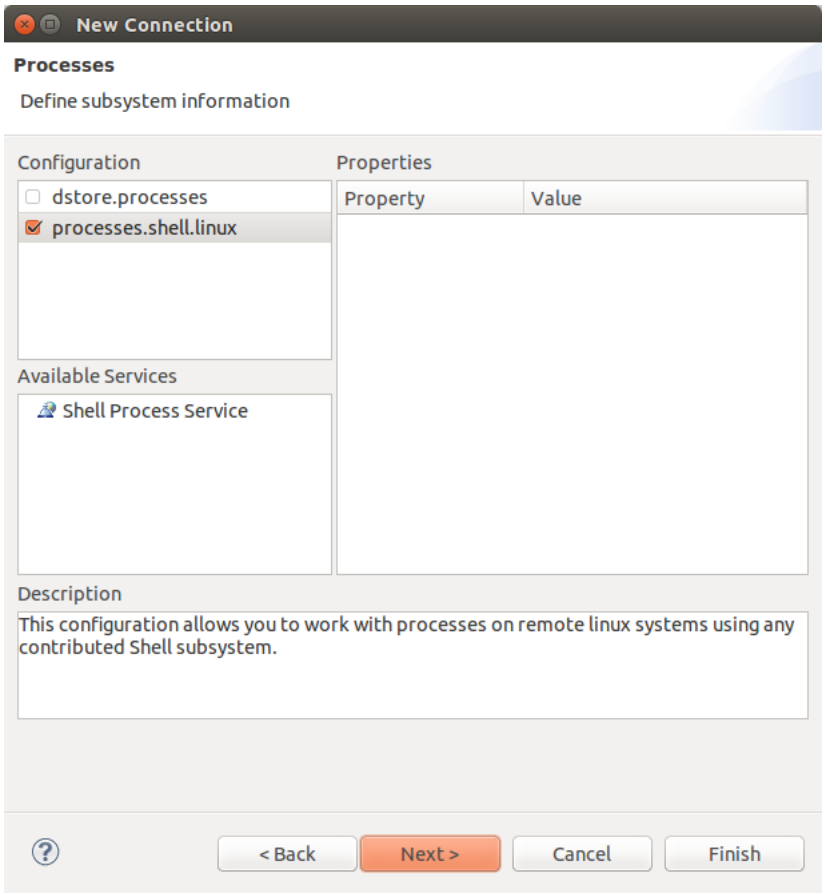
Host name: Enter the IP address of your target EVM.
Connection name: The default value is the same as the host name, but this can be changed to a more human readable value like *My Target EVM*
You can un-check **Verify host name** or leave it checked depending on whether you want to verify the IP address you entered for the *Host name* field.



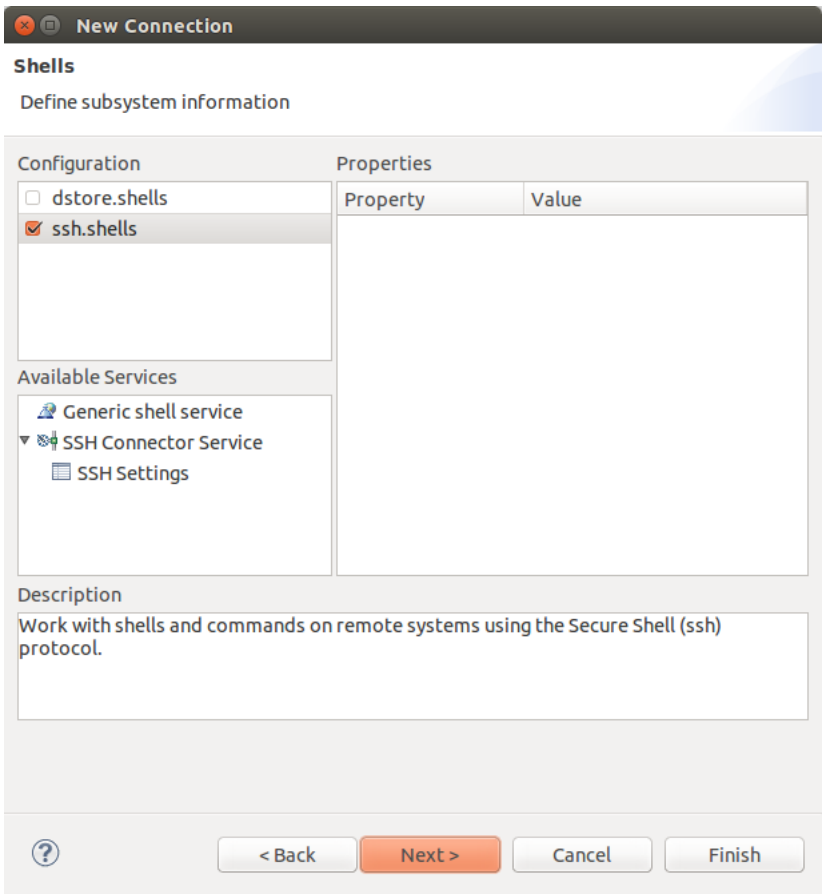
- Click **Next**.
- Check **ssh.files** to use the *Secure Shell* protocol for communication



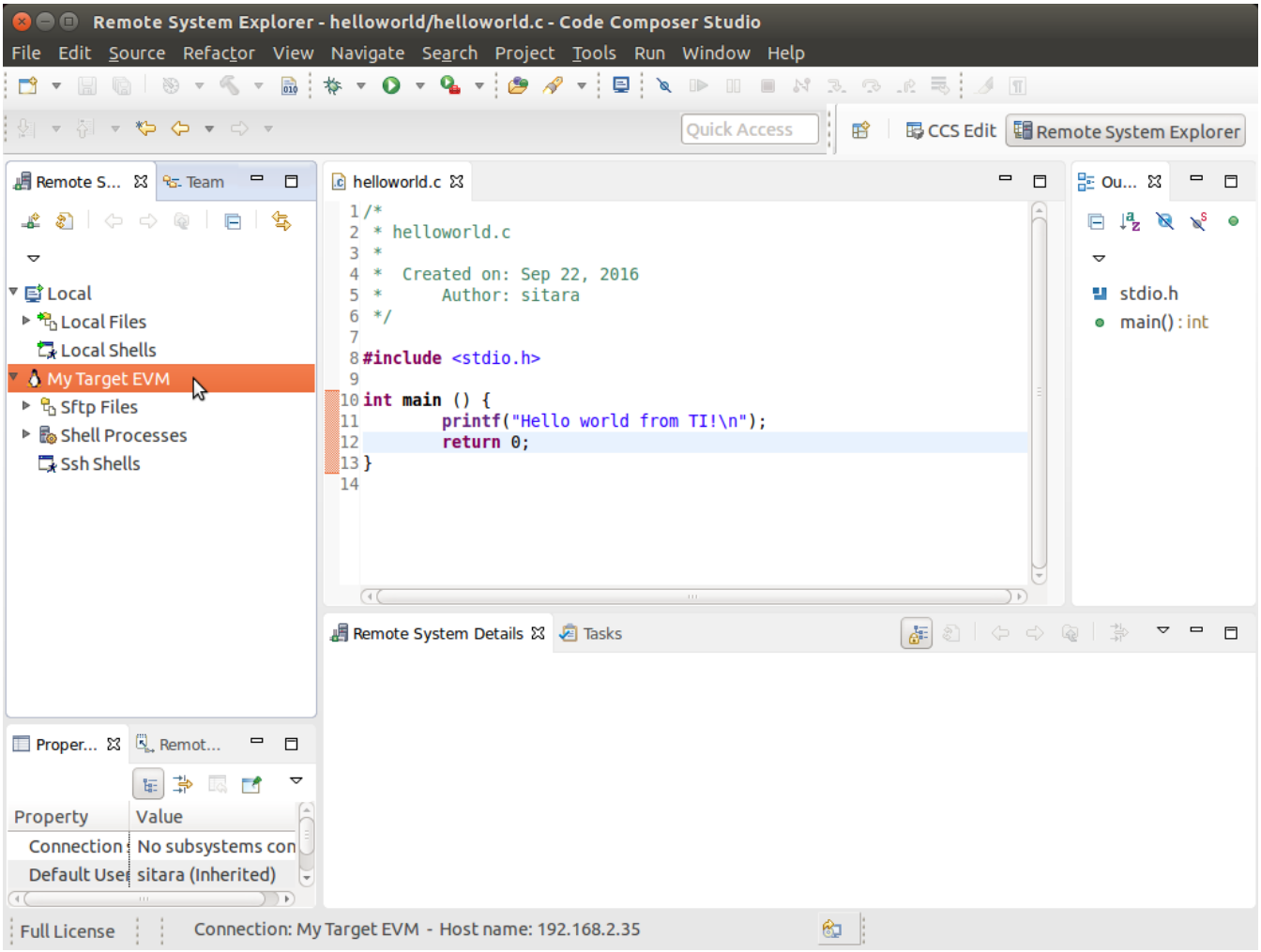
- Click **Next**.
- Check **processes.shell.linux** to use a shell to work with processes on the remote system.



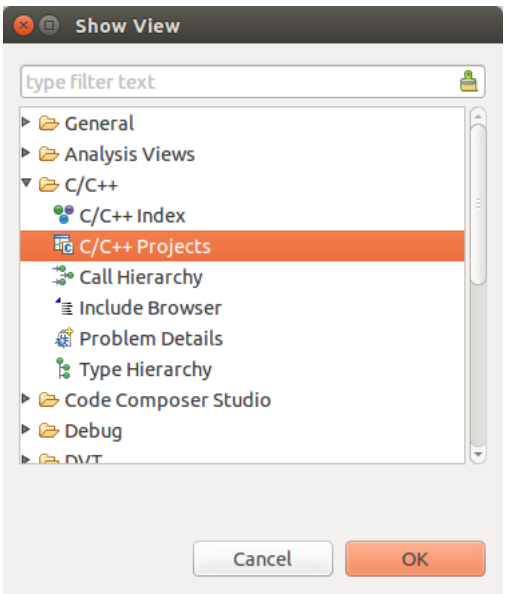
- Click **Next**.
- Check **ssh.shells** to use *Secure Shell* to work will shell commands.



- Click **Finish**.
- You will now see your EVM configuration in the RSE view.



4. When the RSE view was enabled you likely had your C/C++ view closed. In order to get that view back so that you can explore your project perform the following steps
   - Select **Window -> Show View -> Other...**
   - In the **Show View** dialog select **C/C++ -> C/C++ Projects**

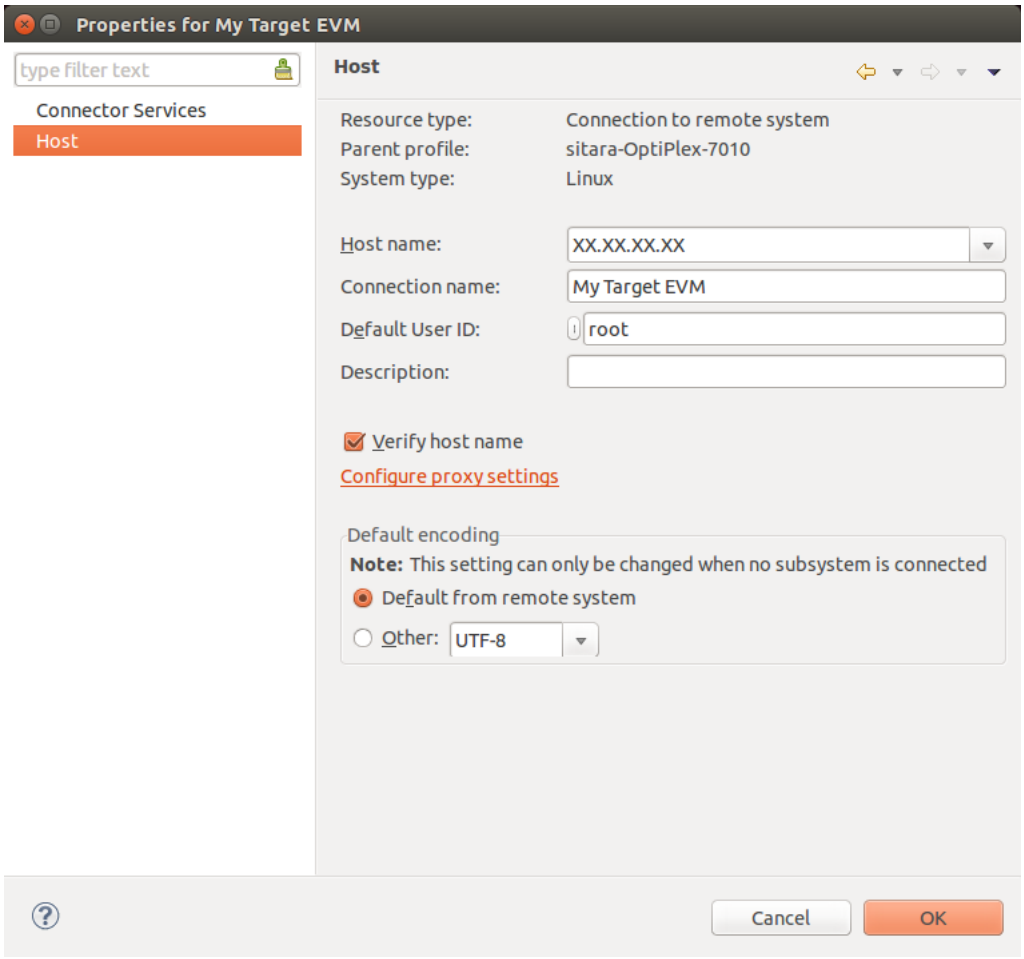


   - Click **OK**.

> **NOTE**
>
> If you do not like the location of the C/C++ Projects view you can drag it to another location in CCS my dragging and dropping the Tab.

5. You should now be able to switch between the RSE and C/C++ Projects view using the tabs in CCS.
6. After the New Connection Wizard has been completed and the Remote System Explorer view has been opened, the new connection must be configured to communicate with the target EVM.

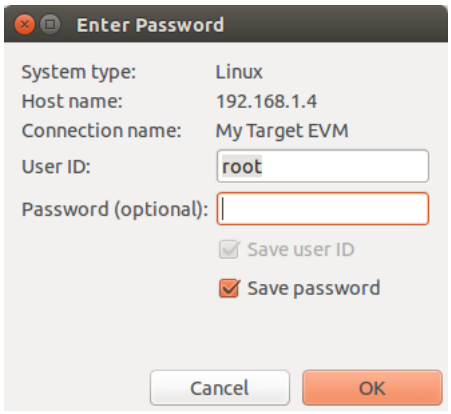
- Right-Click on the *My Target EVM* node and select **Properties** from the context menu.
- In the **Properties** window click on **Host**.
- Change the **Default User ID:** to **root**



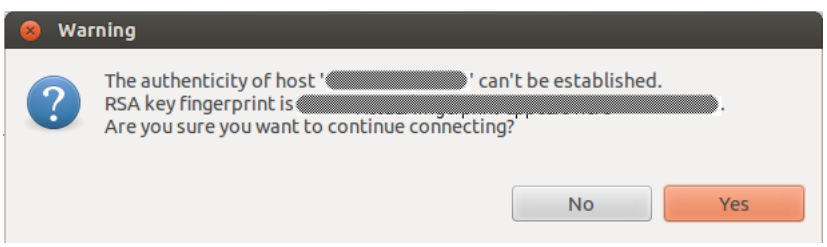
- Click **OK**.

7. The Remote System Explorer is now ready for use. The first time the target EVM file system is booted a private key and a public key is created in the target file system. Before connecting to the target EVM the first time, the public key must be exported from the target EVM to the Linux host system. To configure the key do

- Right-Click the **My Target EVM** node and select **Connect**.
- If you are prompted for a password, leave it blank.



- Click **OK**.
- A dialog like the one shown below may appear. Click **Yes** to accept the key.



- If prompted that */home/sitara/.ssh/known_hosts* does not exist select **Yes** to the dialog boxes.

Transferring Files with RSE

Description

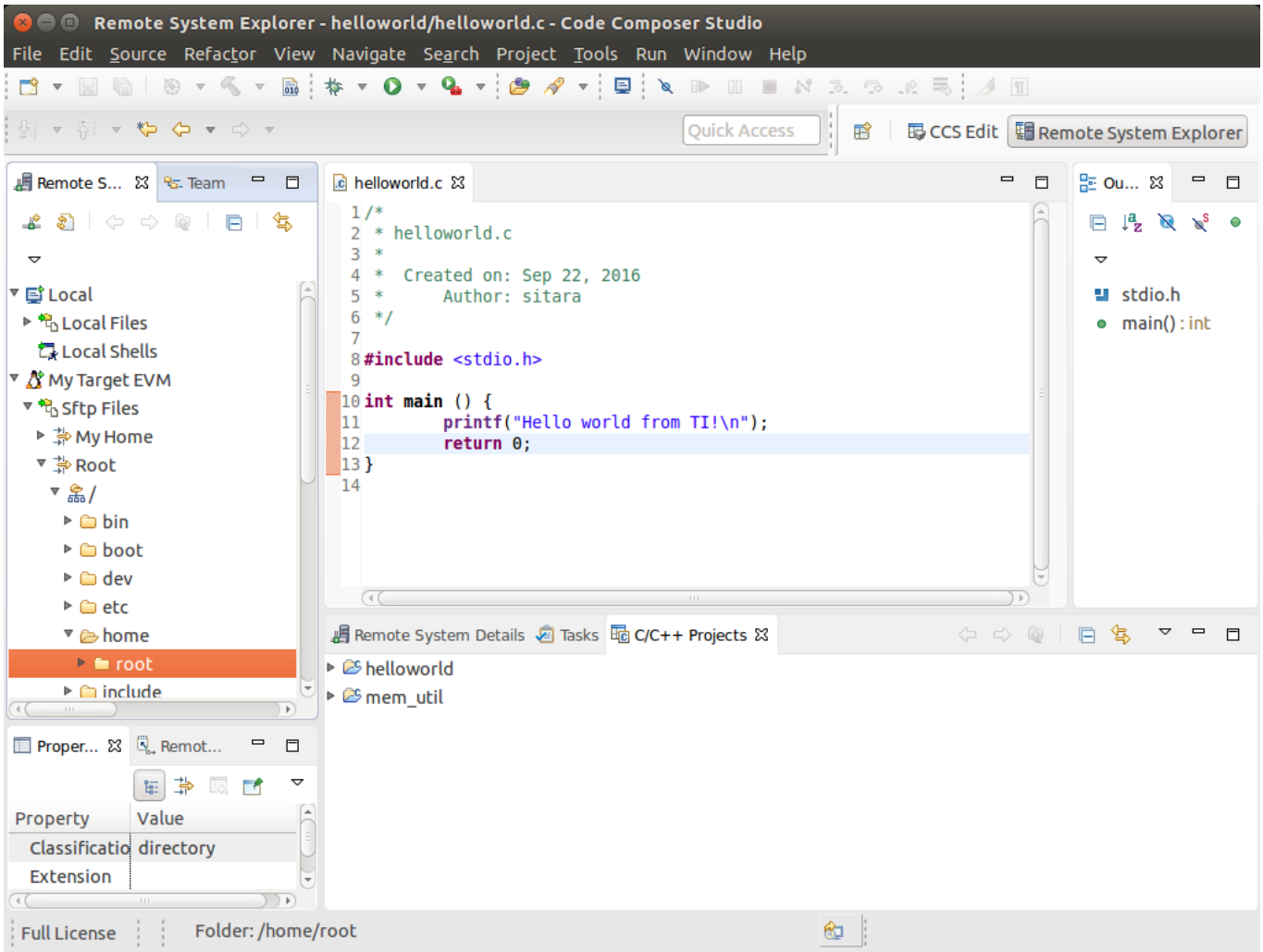
Now that the RSE connection to your target device has been configured you can use RSE to transfer files to the target system. In this section we will cover how to transfer the helloworld executable with RSE. In later sections we will learn how to use RSE with the CCSv6 debugging capability to automatically transfer the executables.

Key Points

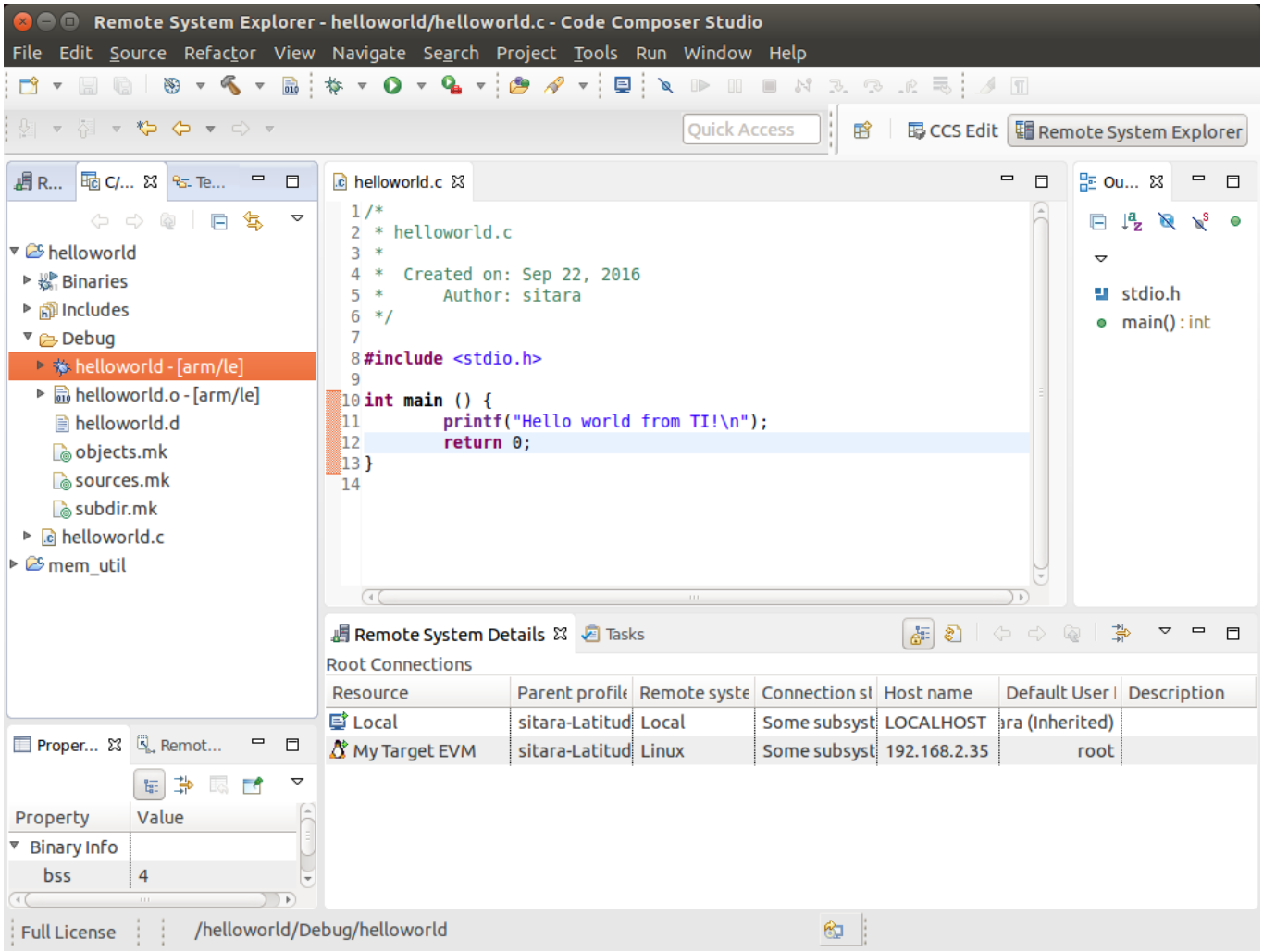
- RSE can be used to transfer files between the host and the target device
- RSE can open terminals on the target device to allow running the executables

Lab Steps

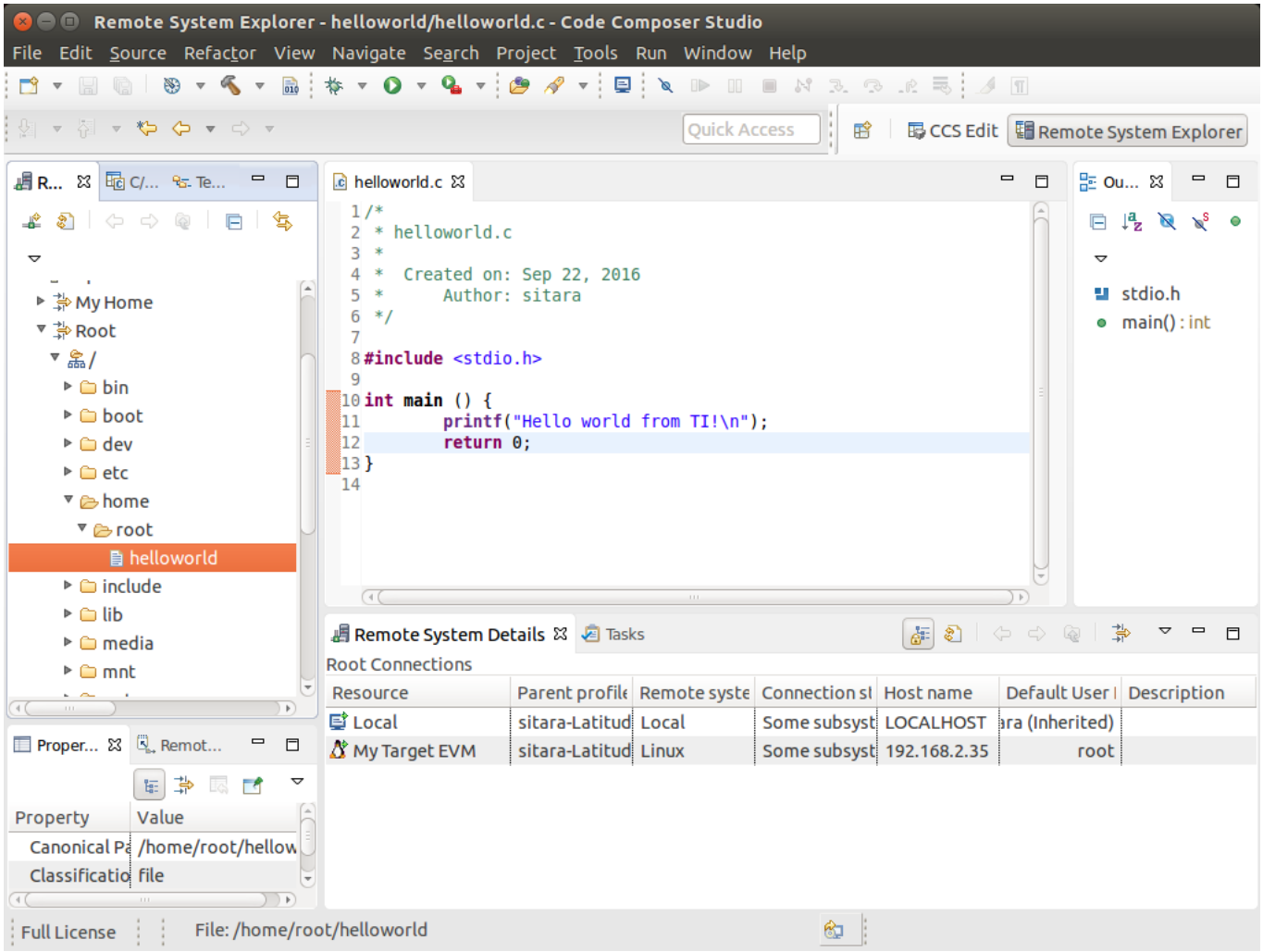
1. Switch back to the RSE view.
2. Open **Sftp Files -> Root -> /home -> root**.



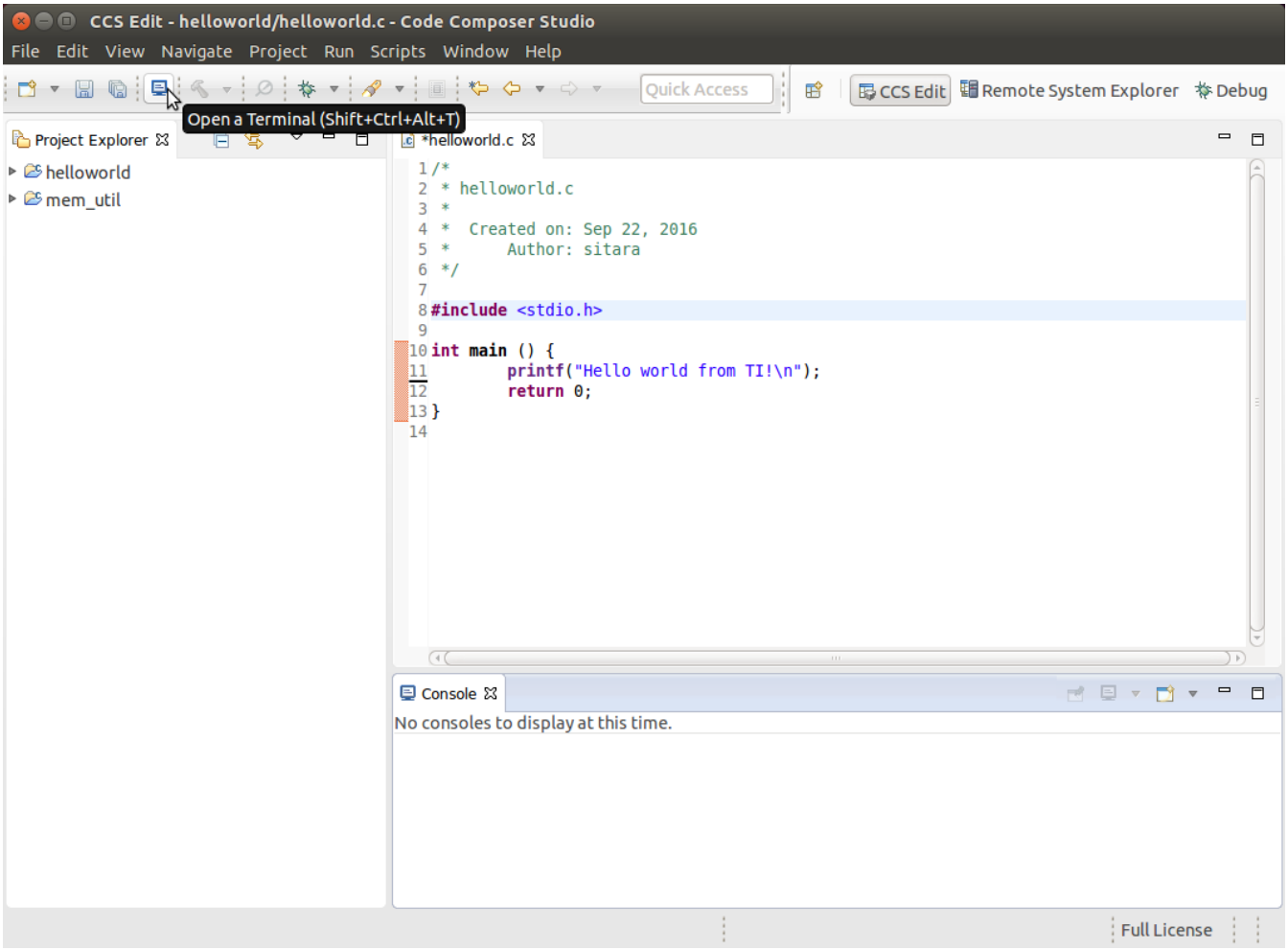
3. Switch to the C/C++ view using the tab. Expand *helloworld* to find the helloworld executable in the Debug directory.



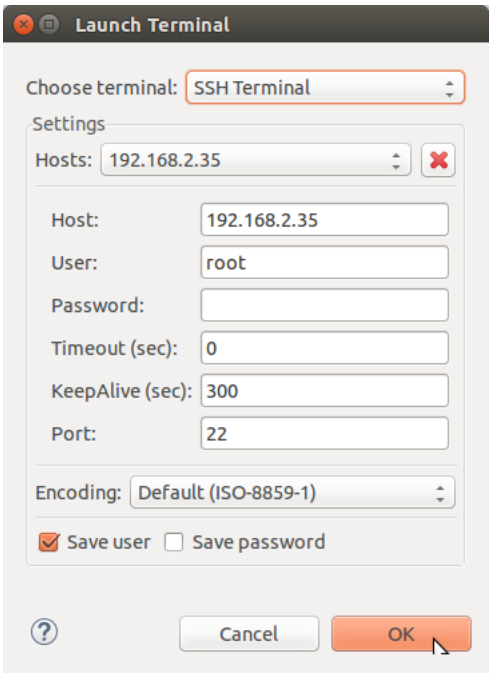
4. Right-Click the helloworld executable and select **Copy** .
5. Switch back to the RSE view and Right-Click on the root directory and select **Paste** .
6. You should now have the helloworld executable transferred to your target device.



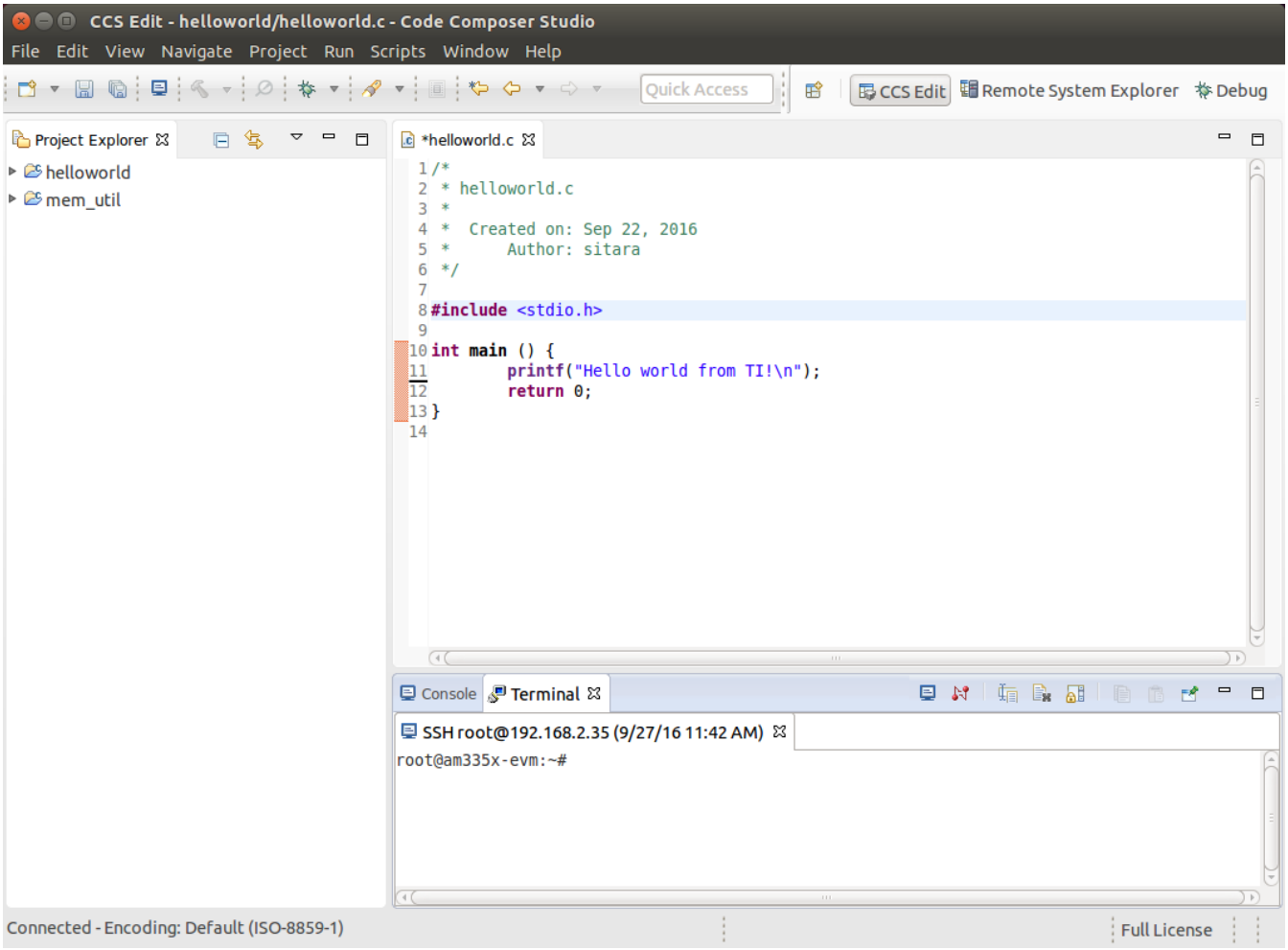
7. In order to execute the helloworld binary that was just transferred you need to open a terminal to the target device. Or, you could use some other terminal/console (i.e. Teraterm, minicom, etc.). Since CCS provides one built right in, let's use that one. The following steps will cover opening a new terminal to the target device.
   - Click on the "Open Terminal" icon on the toolbar:



- Use the below settings to configure the terminal (make sure to use your target's IP Address):

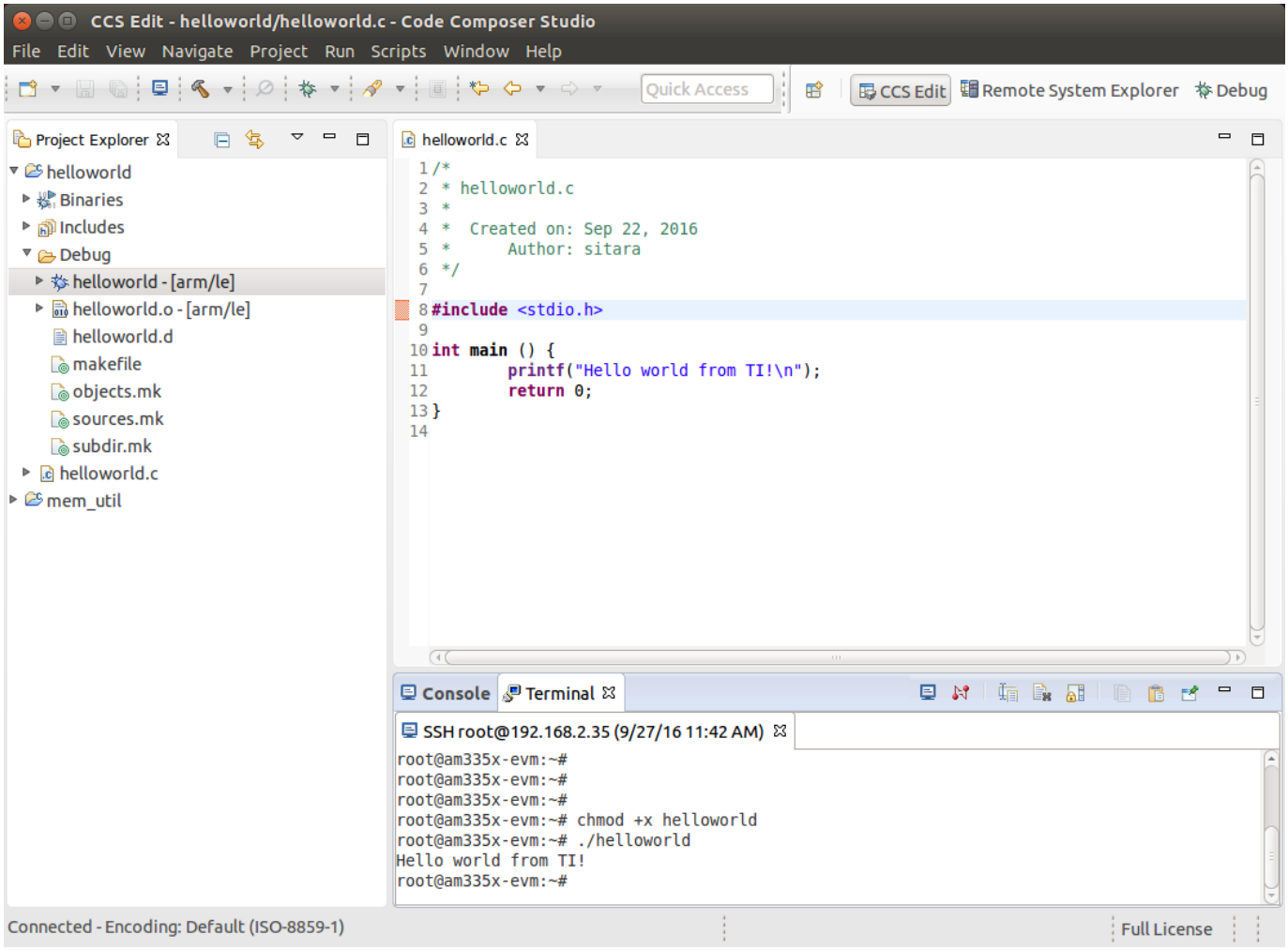


- A terminal window will now be open:

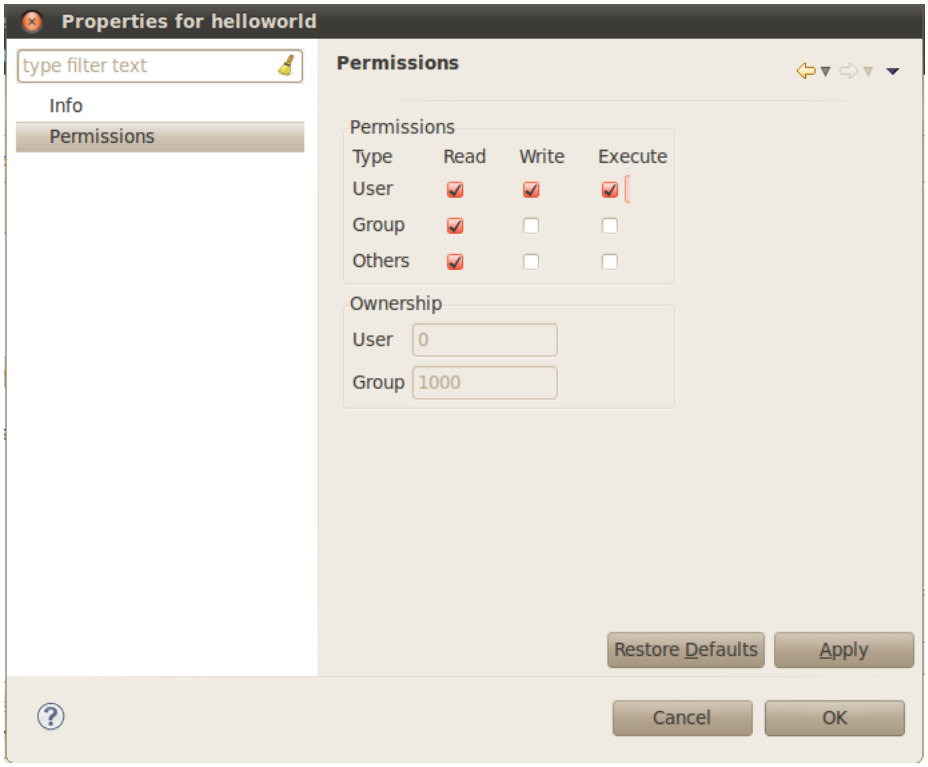


8. To run your helloworld application use the following commands in the terminal window:

chmod +x helloworld
./helloworld



- It is also possible to make a file executable using RSE by right-clicking the file in the RSE view and selecting **Properties -> Permissions** and enabling the Execute permission.



- Click **Apply**.

Debugging with CCS and GDB

Description

Now that you have learned to use CCS to create a project, build your sources for the ARM, transfer the executable to the target, and run that executable it would be useful to know what to do if there is something wrong with the executable. In this section you will learn how to configure CCS to debug the helloworld application.

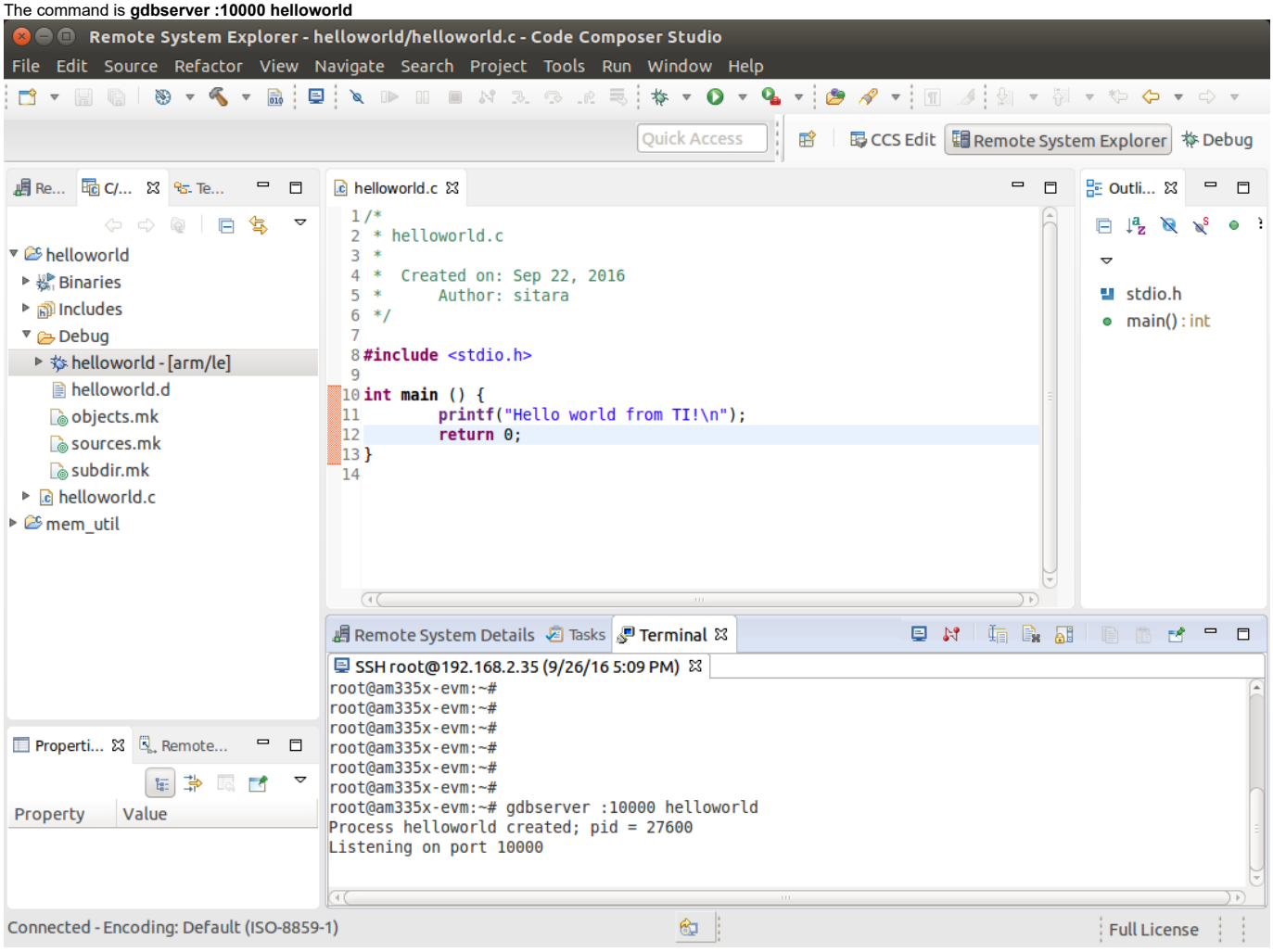
Key Points

- CCS can be configured to automatically build your project files.
- RSE is useful for transferring files between the debug host and the target.
- CCS has a terminal that can be used to run commands on the target, including starting gdbserver.
- You can use break points to stop execution at a particular point.

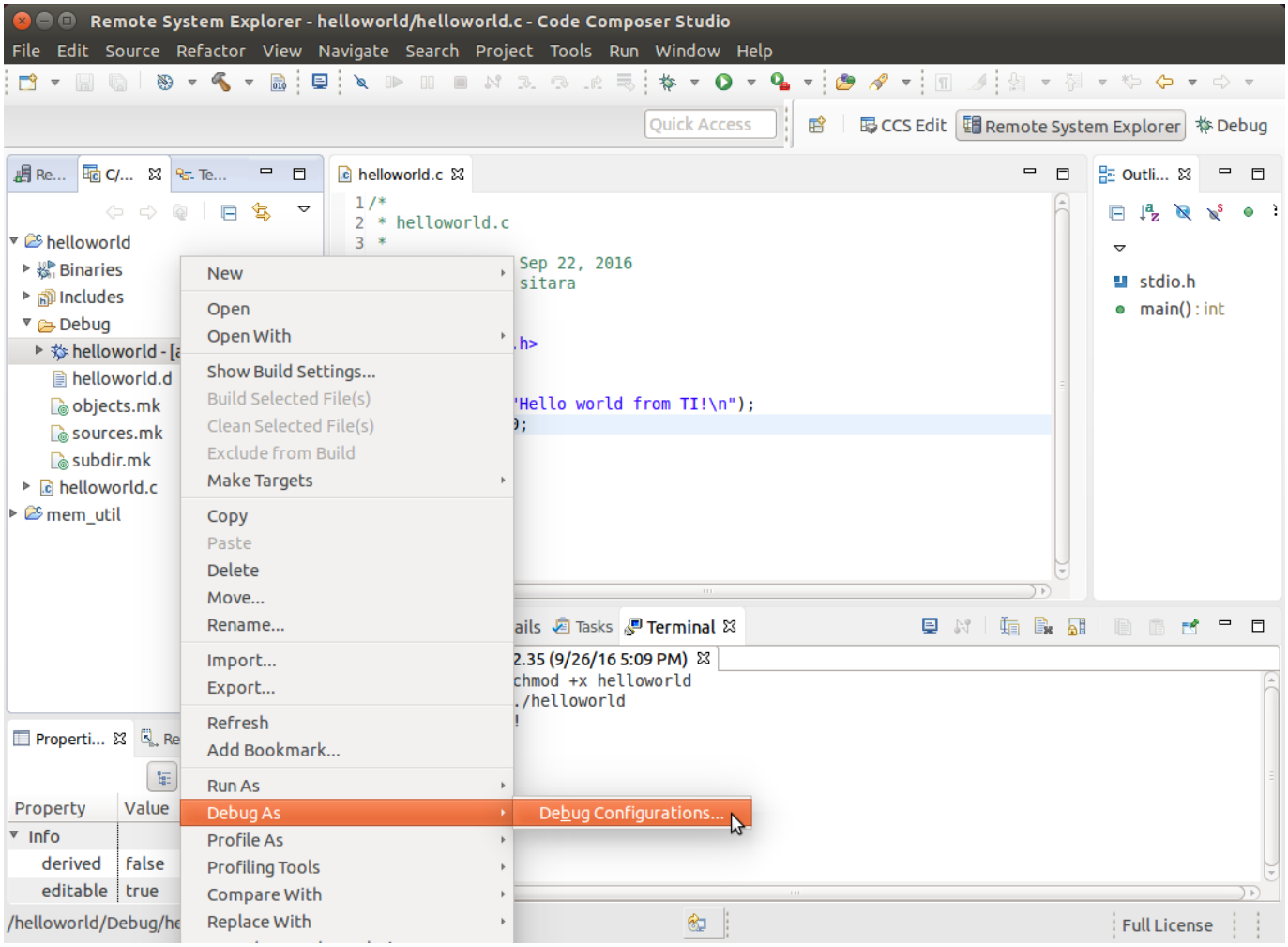
Lab Steps

1. First, start gdbserver on the target using the terminal in CCS:

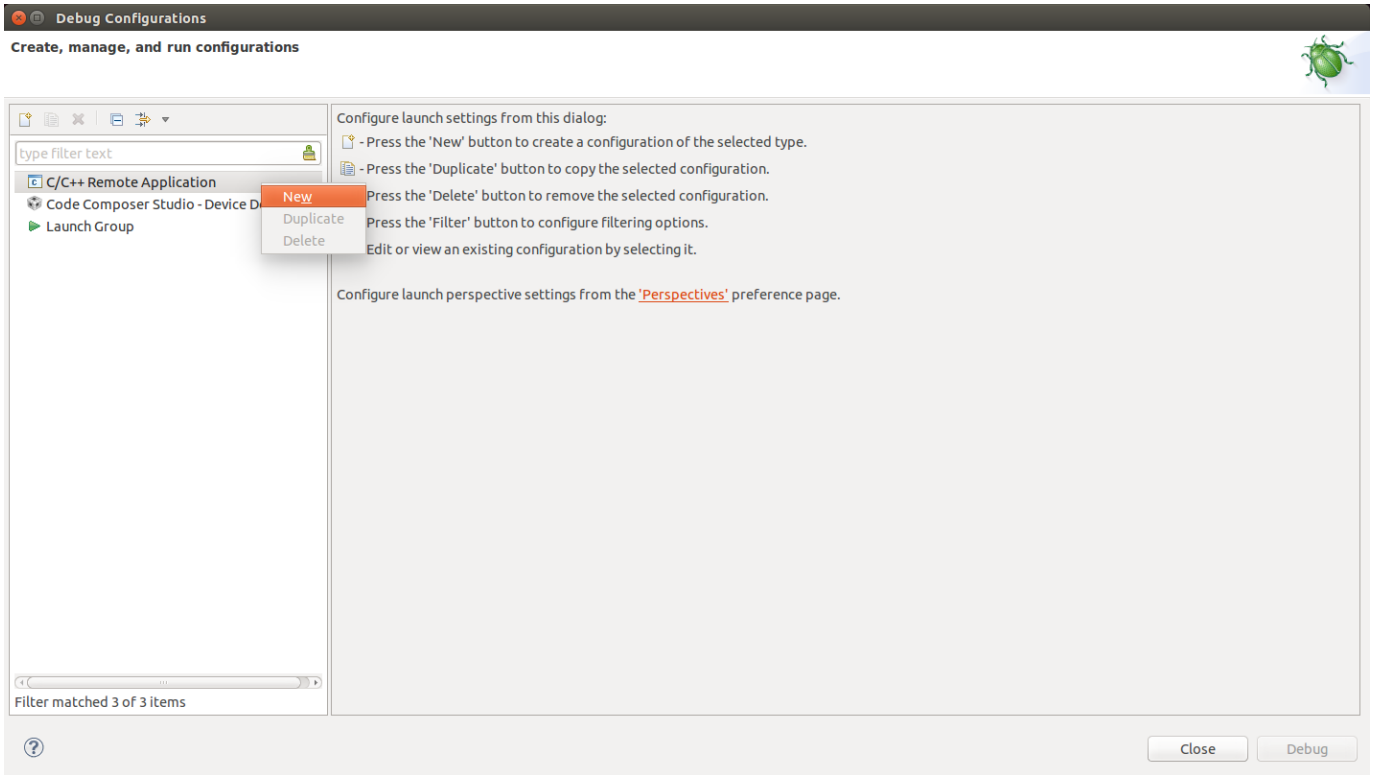
   The command is **gdbserver :10000 helloworld**



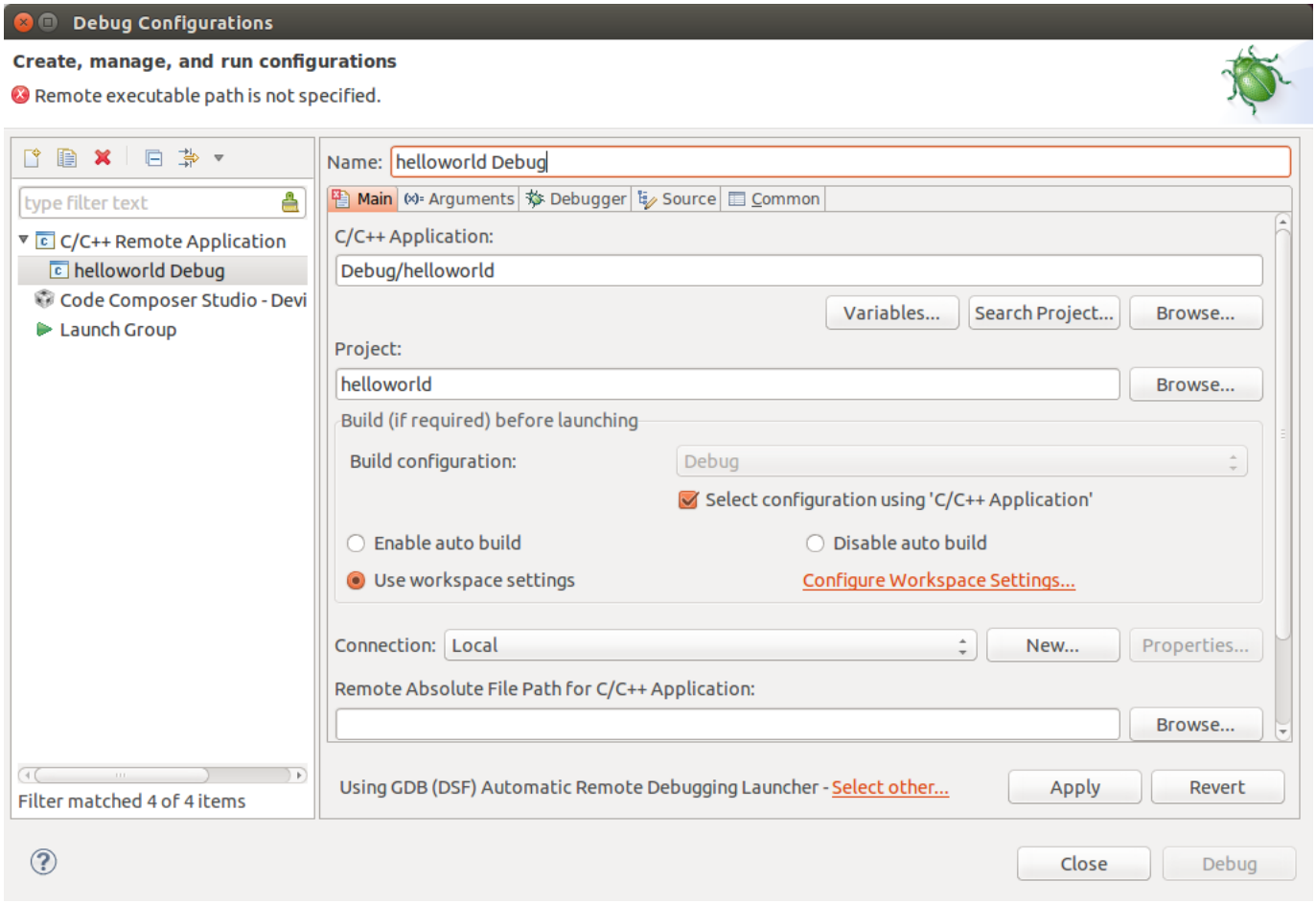
2. Switch to the C/C++ Project view
3. Right-Click the Debug/helloworld executable and select **Debug As -> Debug Configurations...**



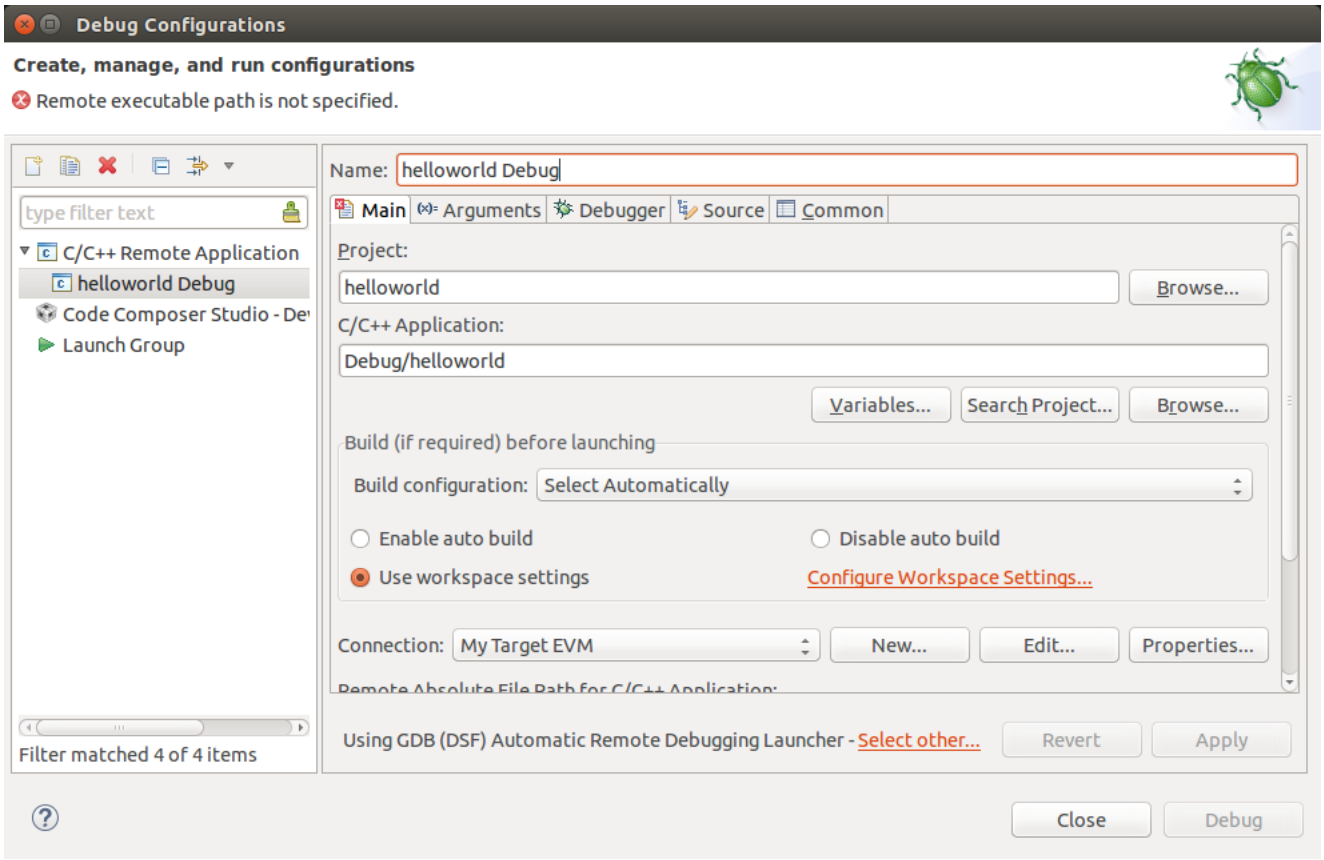
4. Right-Click C/C++ Remote Application and select **New**.



5. By default you will have a *helloworld* configuration created. Change the Name to **helloworld Debug**.



6. On the *Main* tab, add the Remote Absolute File Path to point to **/home/root/helloworld**. If the *Connection* is selected as Local, change it to **My Target EVM**

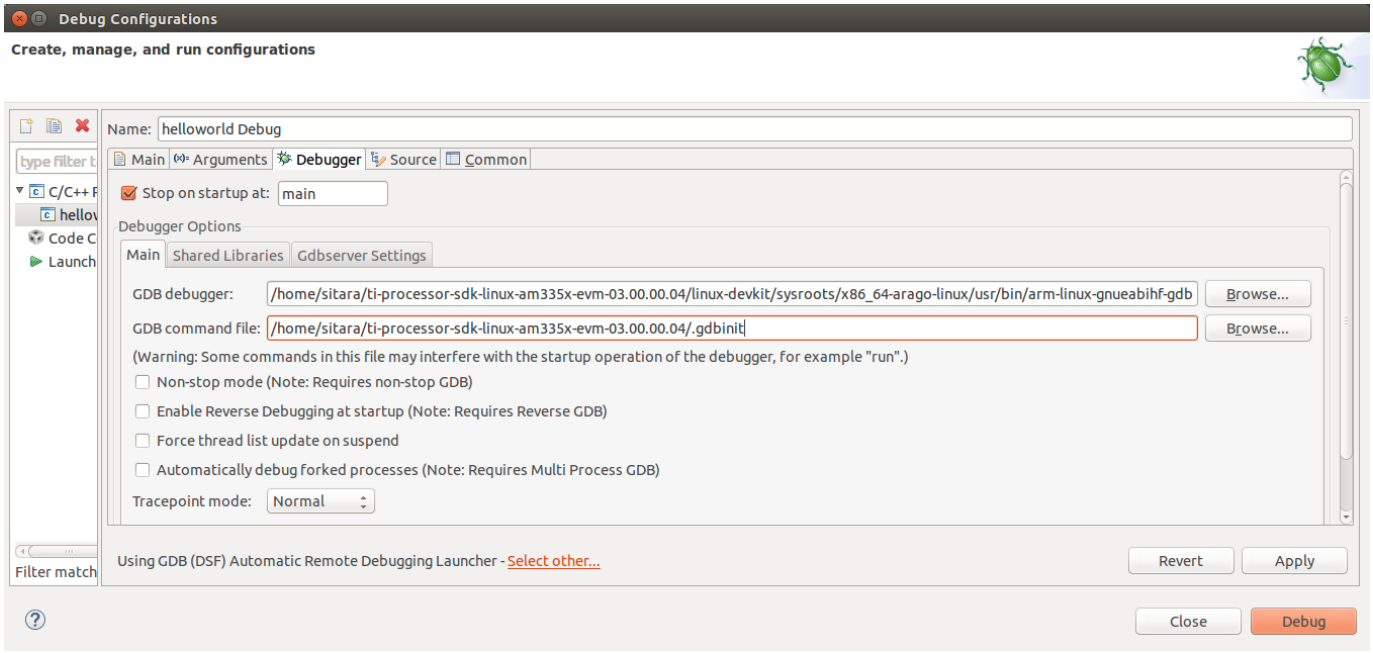


7. Switch to the *Debugger* tab and modify the following settings:

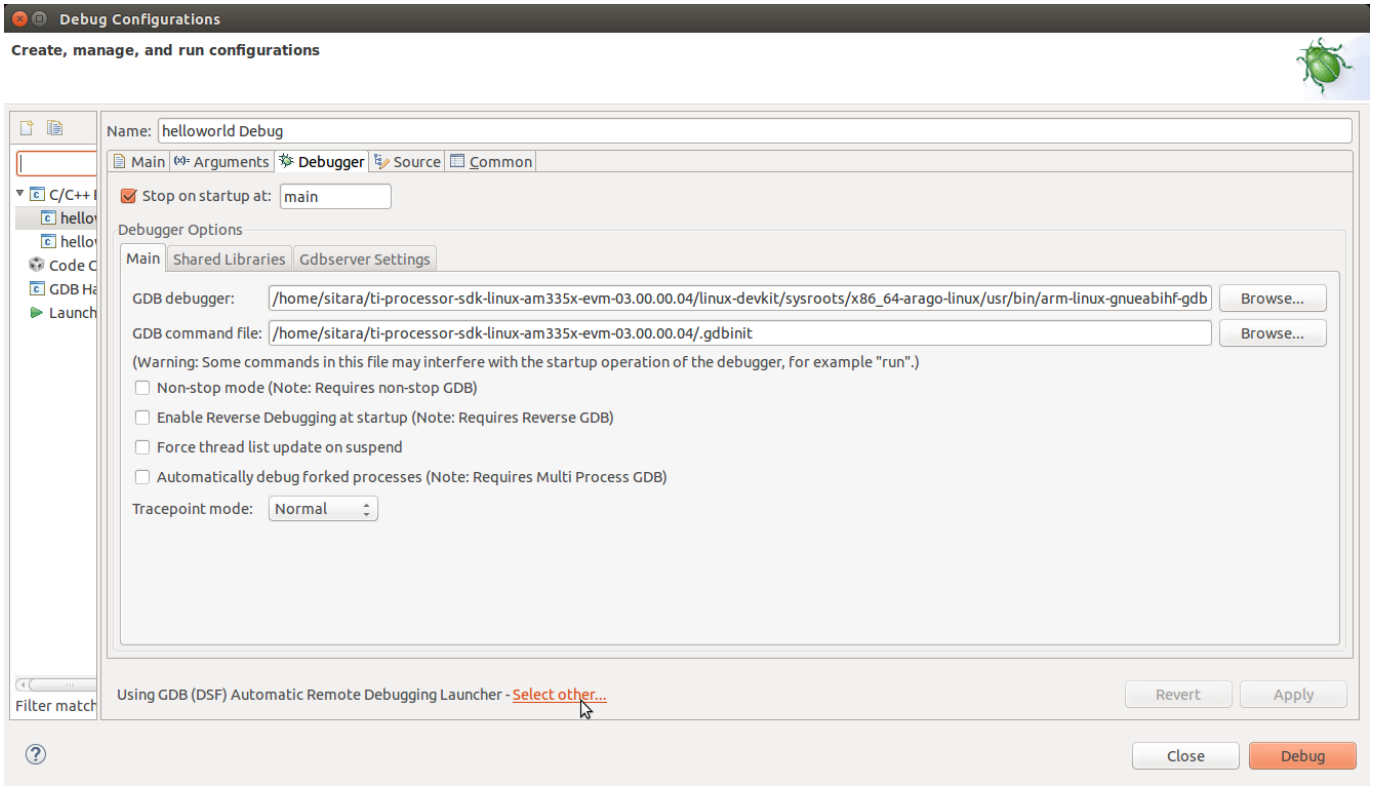
- *GDB debugger:* **/home/sitara/ti-processor-sdk-linux-<machine>-<version>/linux-devkit/sysroots/<Arago Linux>/usr/bin/arm-linux-gnueabihf-gdb**

GDB command file: **ti-processor-sdk-linux-<machine>-<version>/.gdbinit**

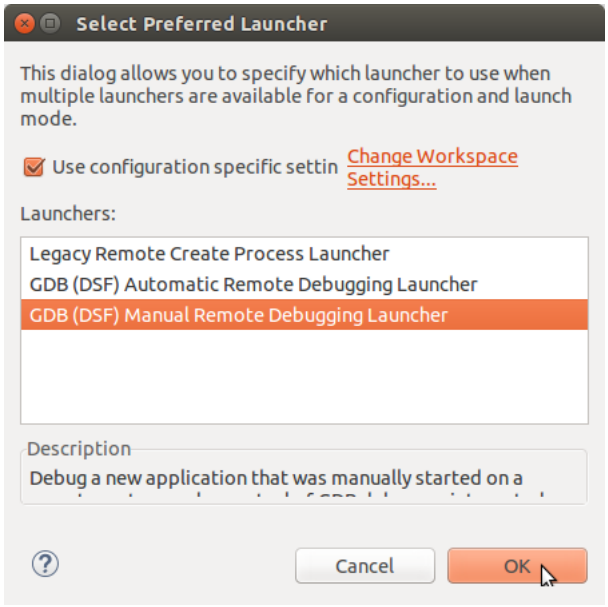
> **NOTE**
>
> If you use the file browser, you will need to Right-Click and select **Show hidden files** to find the .gdbinit file.



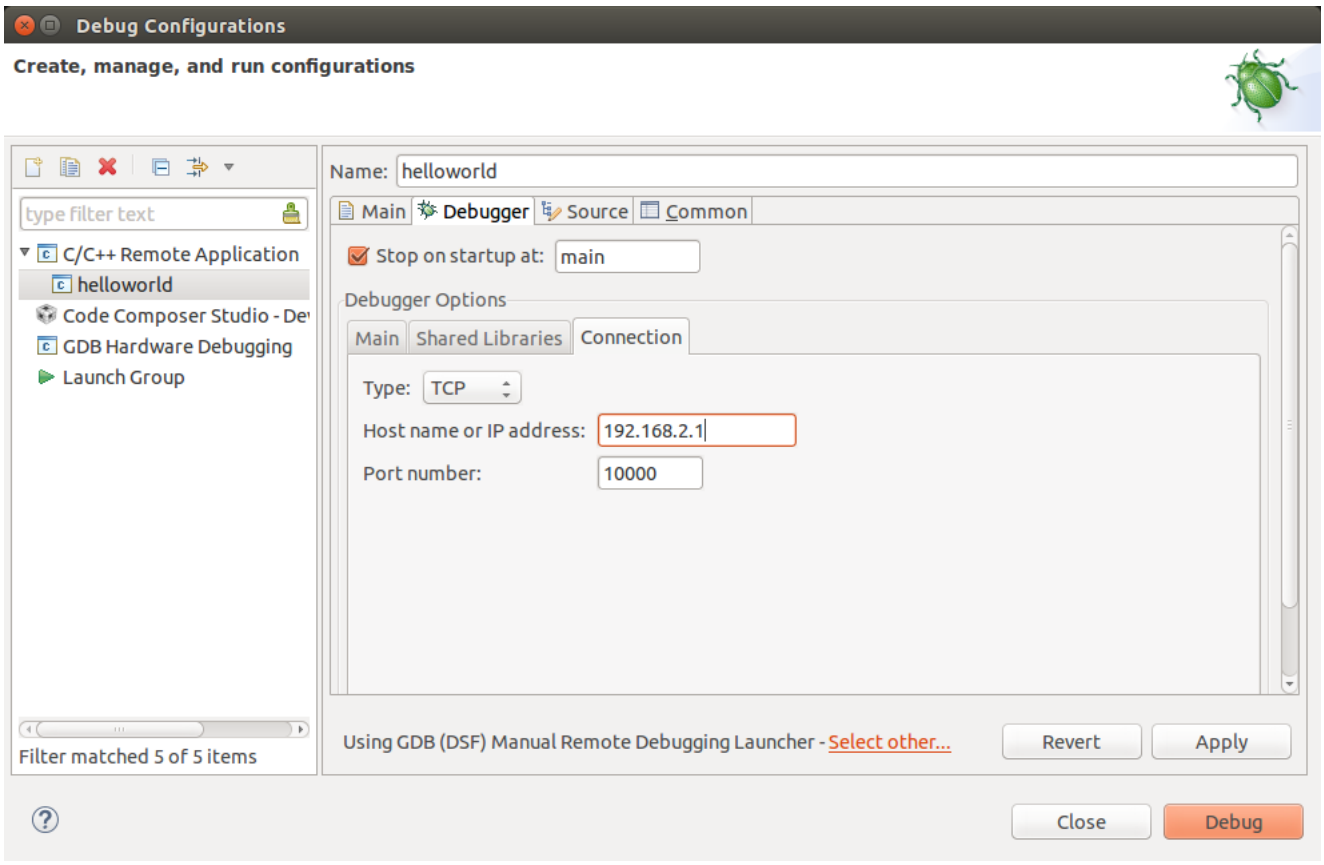
8. Now click on "Select Other..." next to Using GDB (DSF).



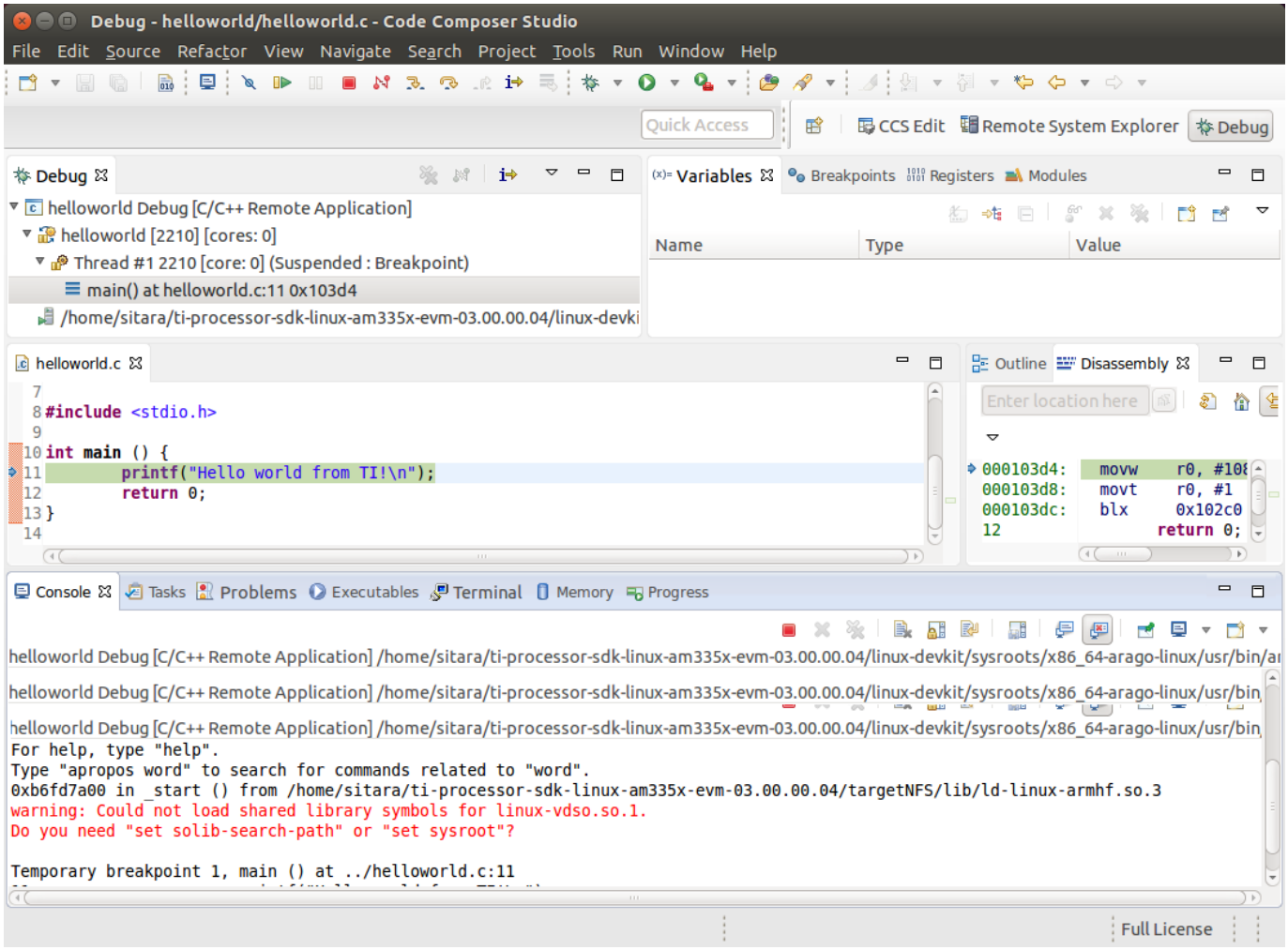
9. Select the "manual configuration"



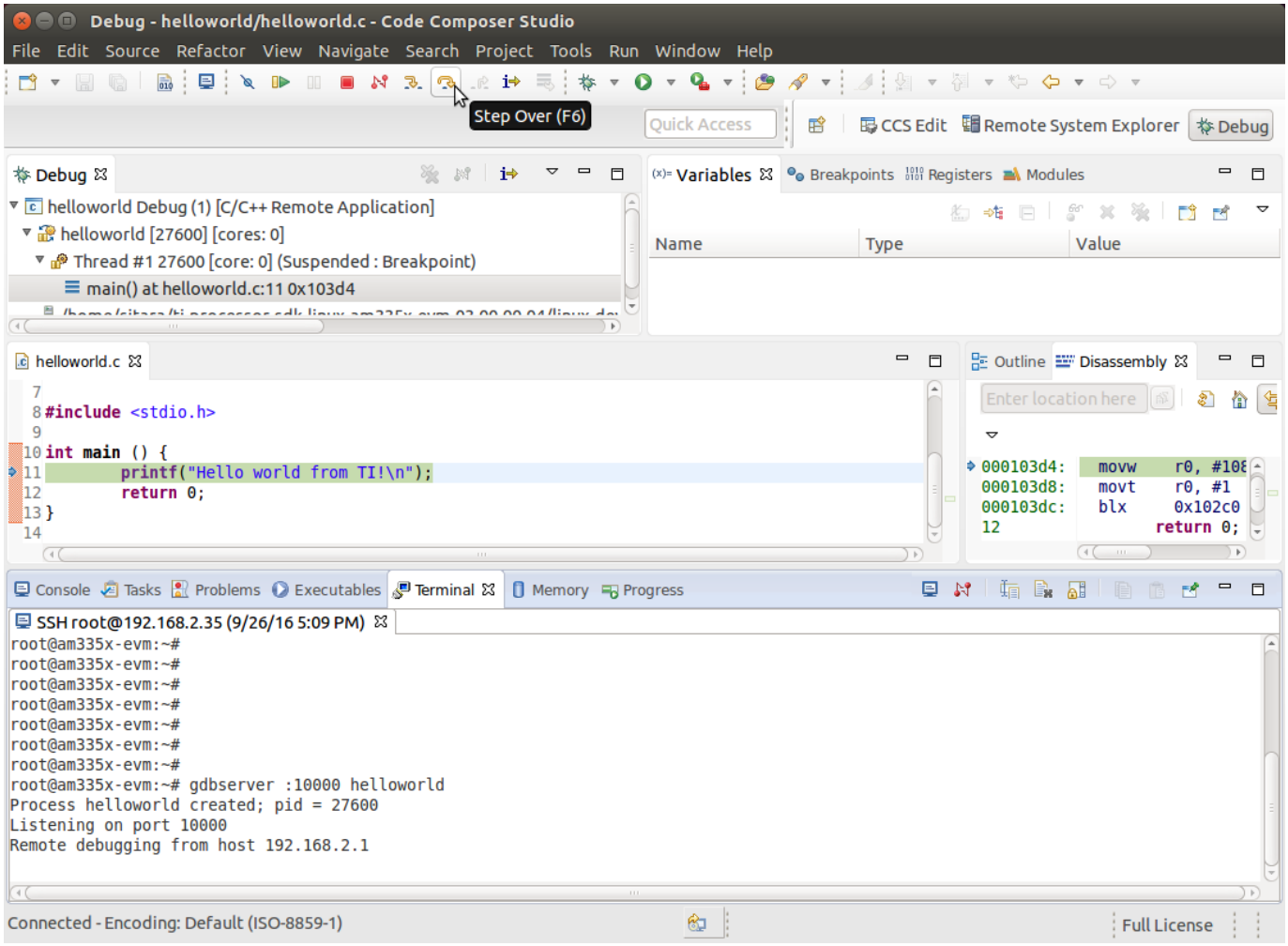
10. On the "connection" tab, set the server IP address:



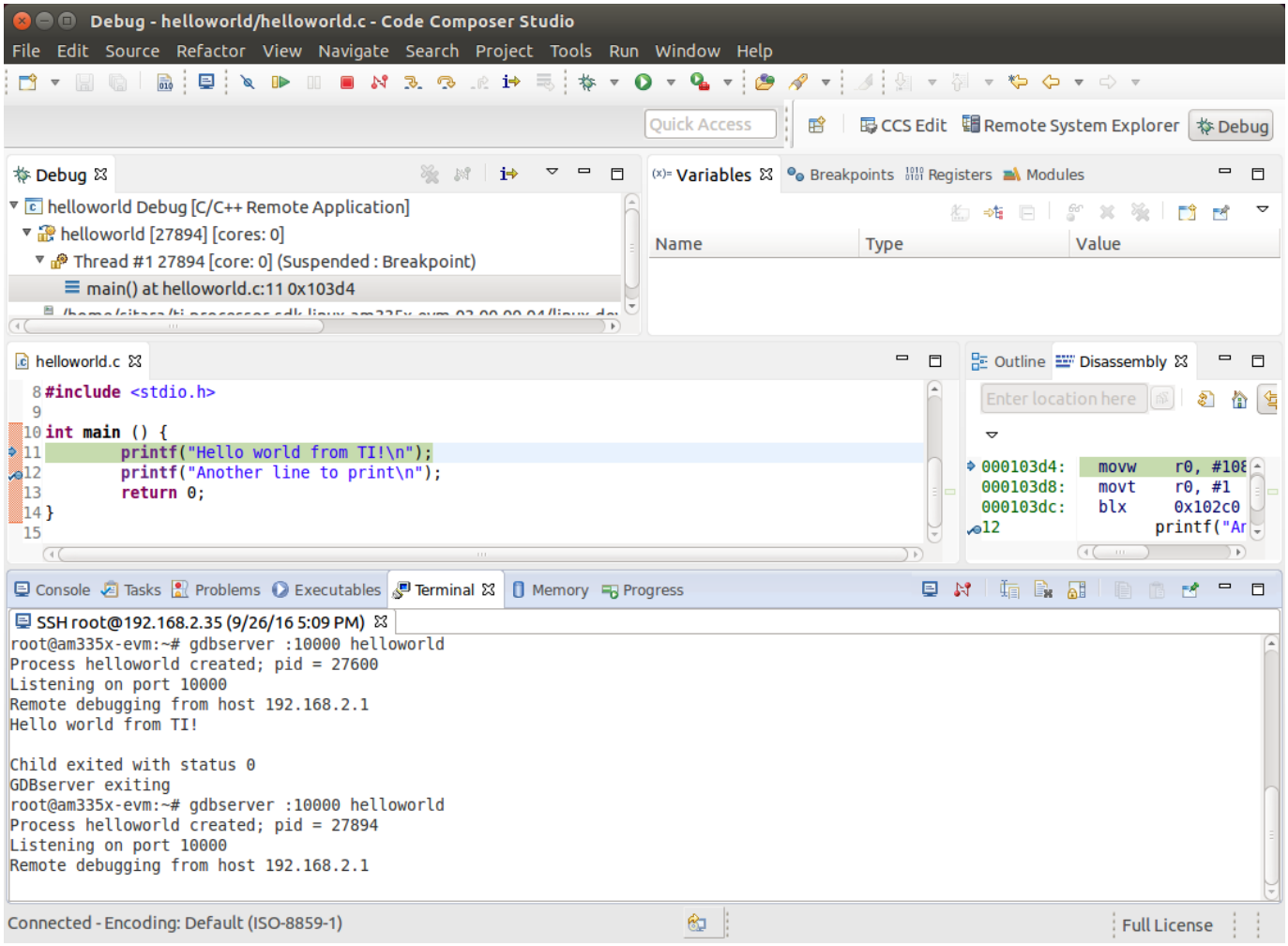
11. Click **Apply**.
12. Click **Debug** to compile the project and start the debug. You system should be halted at the first line of the *main* function. You can now use the controls on the top of the window to control the debug session.



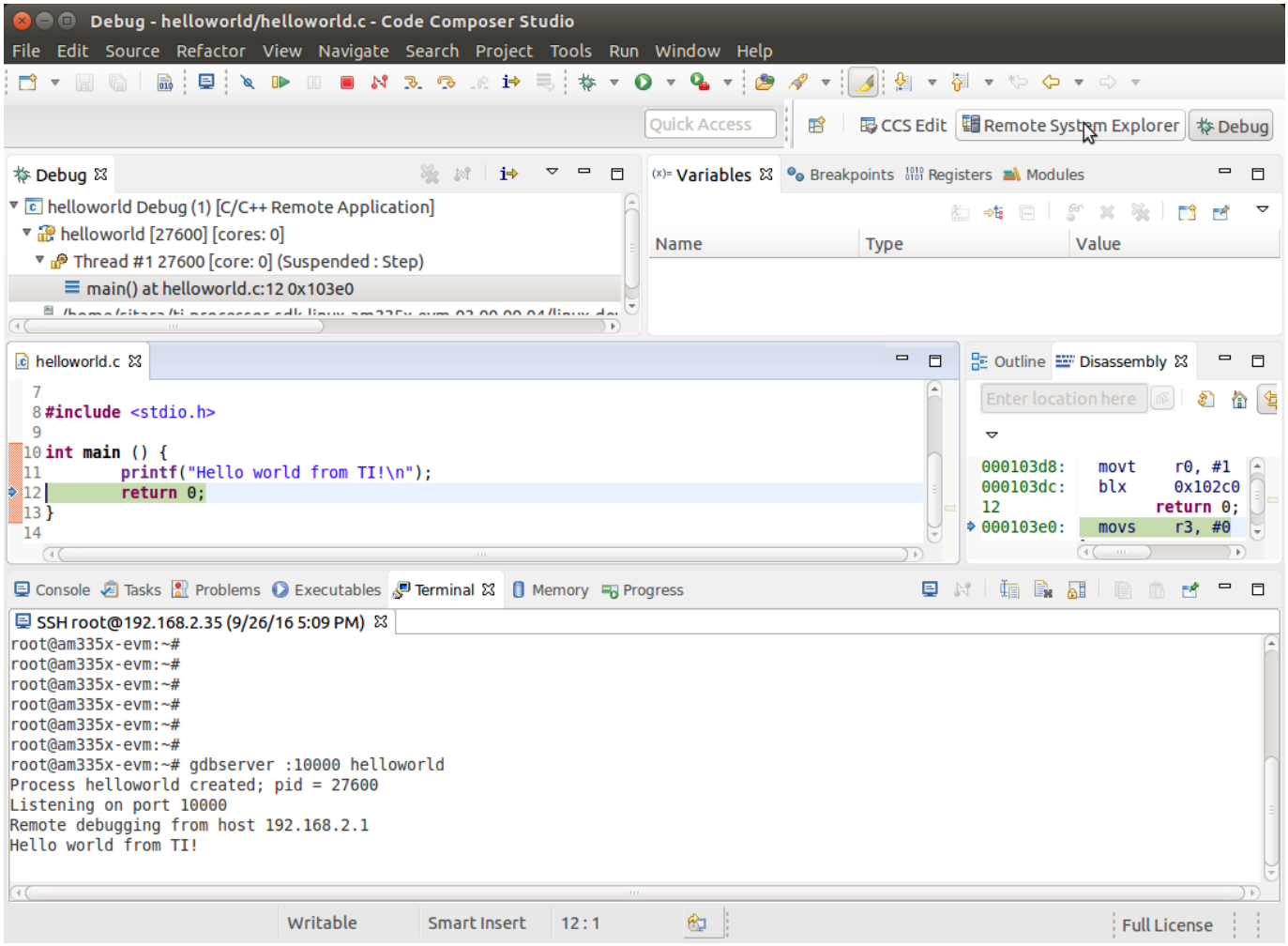
13. Click on the **Step Over** button. You should see the active line move to the *return 0* line and the Hello World message should be printed in the console output.



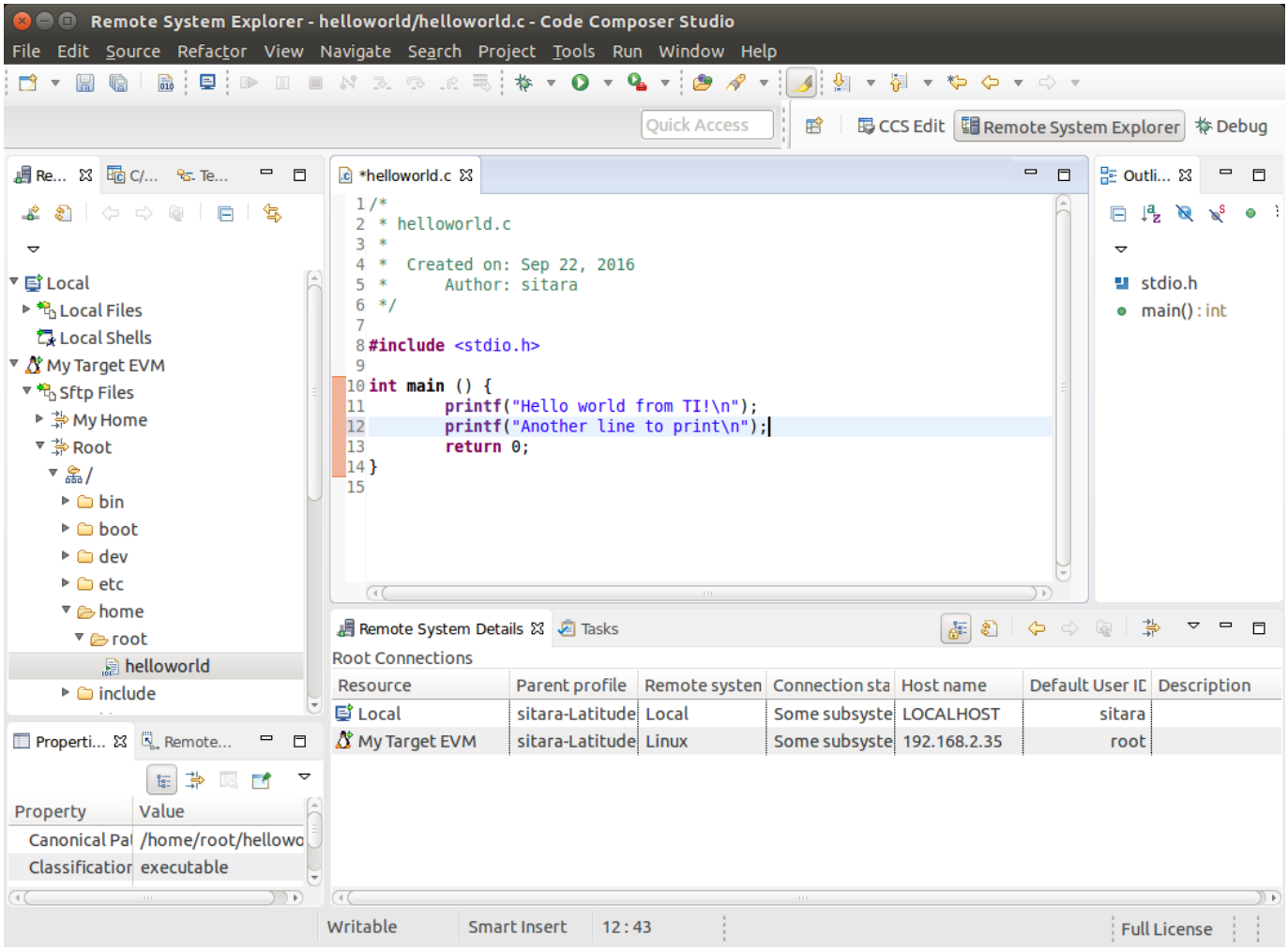
14. Click the **Resume** button to finish execution and exit the program.
15. Use the terminal to restart gdbserver on the target (it exited when the program finished running):



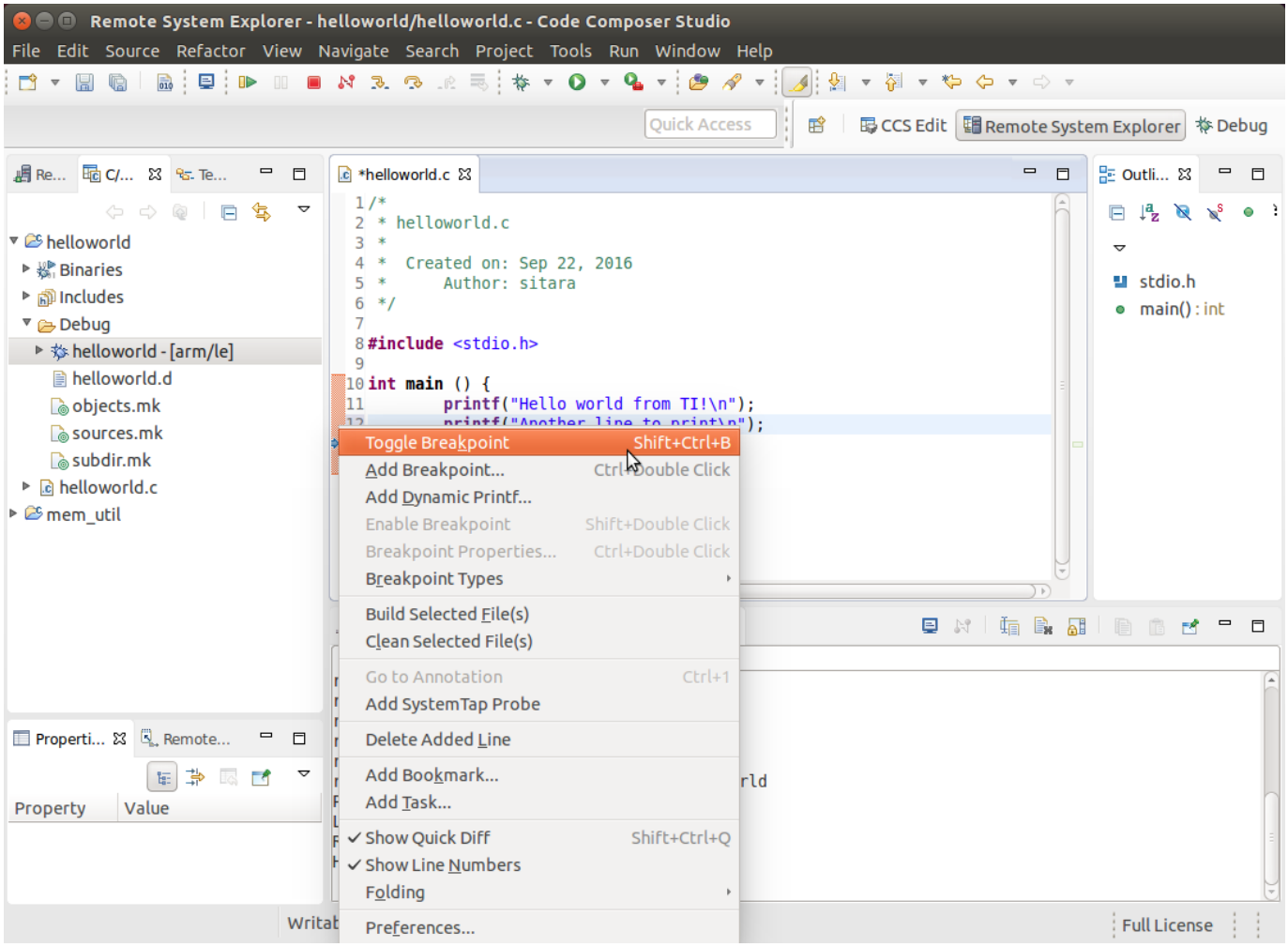
16. Now select **Remote System Explorer** in the upper-right corner of the window to leave the debugger view.



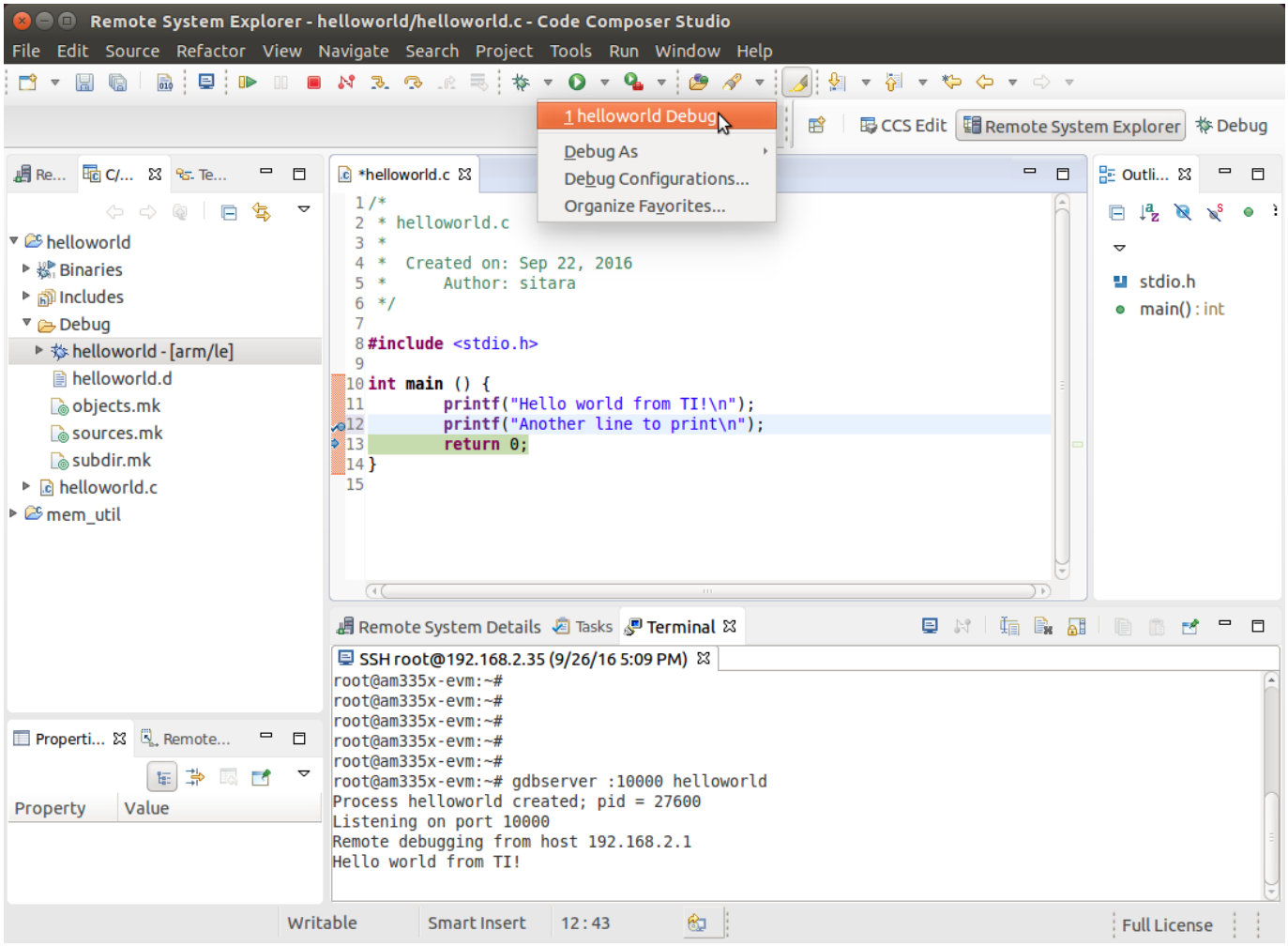
17. Edit the helloworld.c file like the image shown below to add another *printf* line.



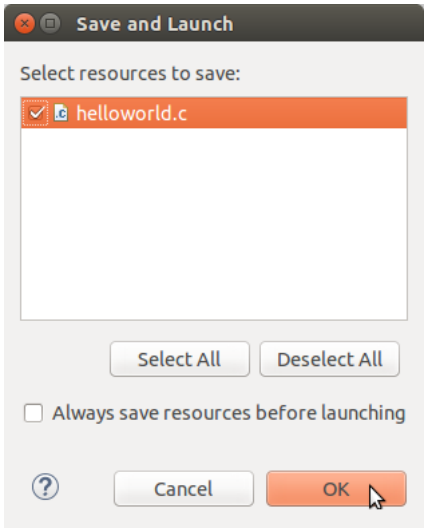
18. Right click on the grey column to the left of the new line and select **Toggle Breakpoint**.



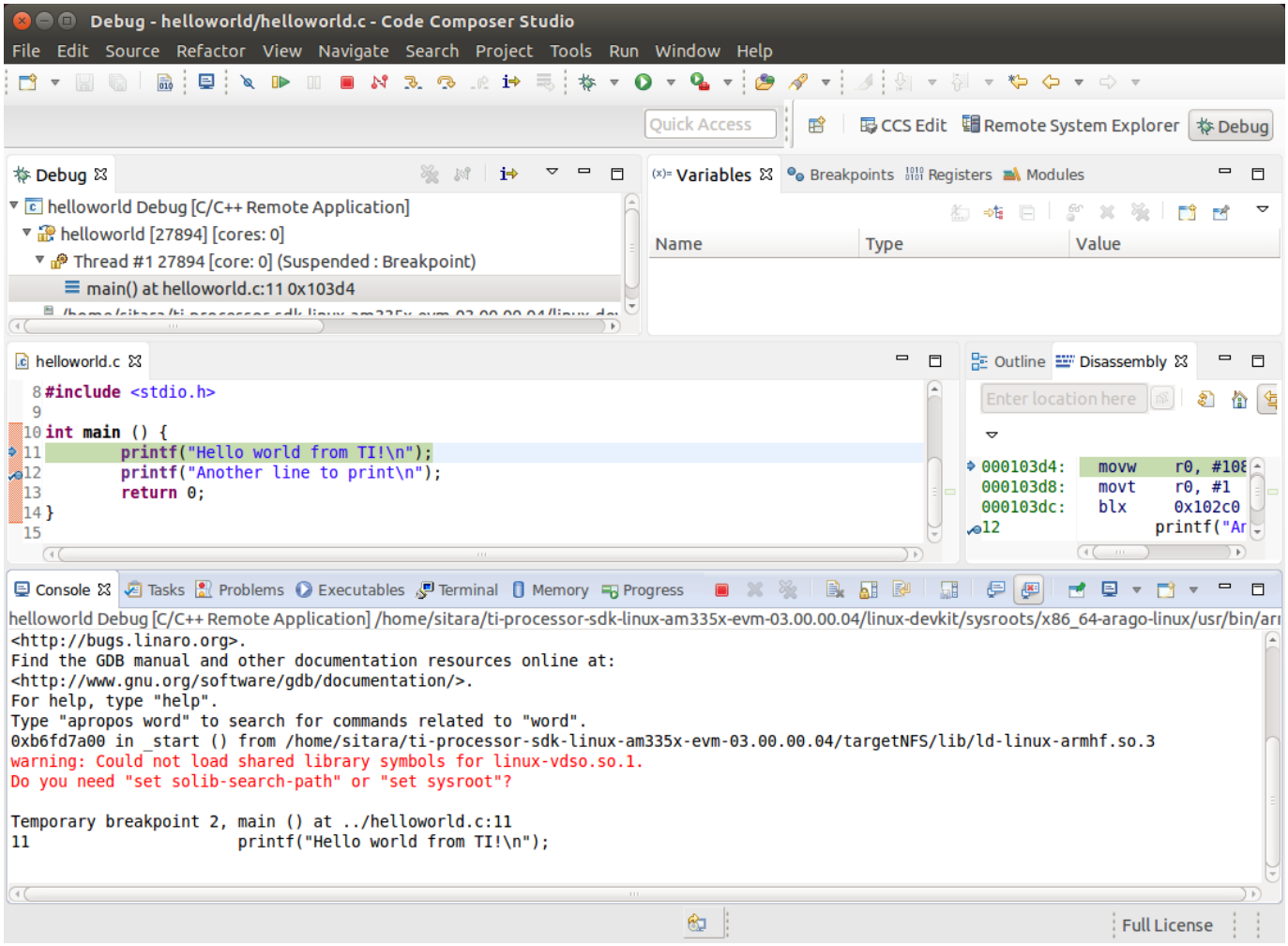
19. Click the arrow next to the bug icon in the top toolbar and select the **1 helloworld Debug** configuration.



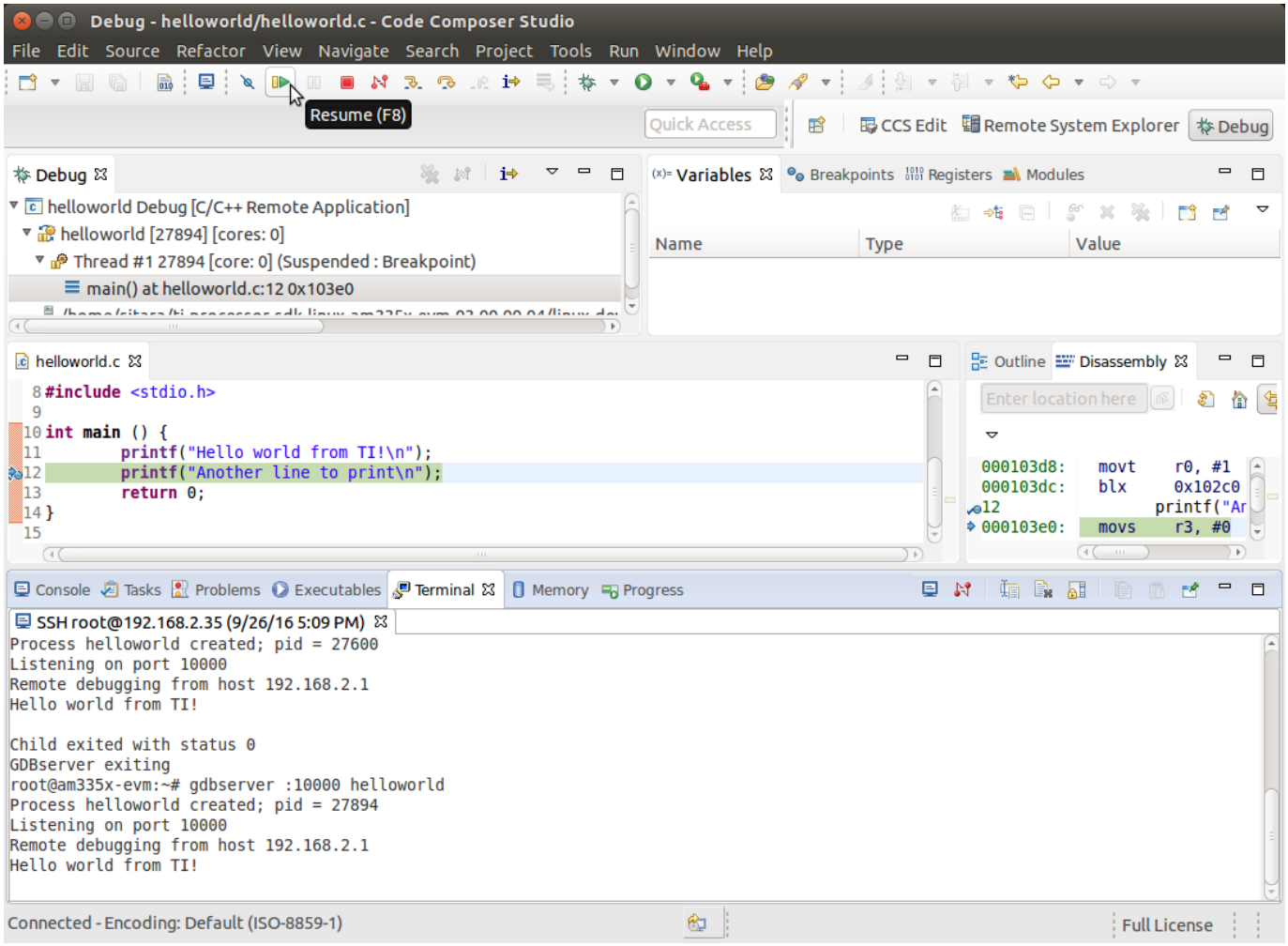
20. When prompted to save your changed to *helloworld.c* select **OK**.



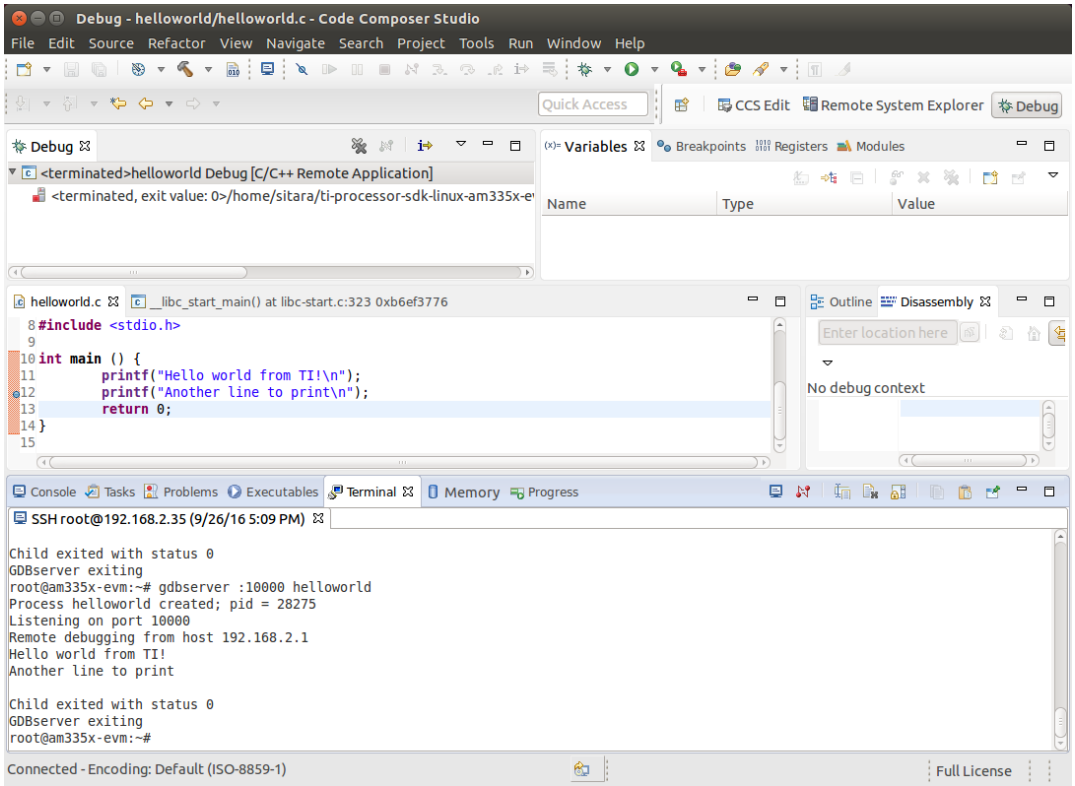
21. You will see the sources get compiled and the debug view will open automatically. Once again the current location will be the first line in the *main* function, but this time there will be a breakpoint shown in the sources as well. **Don't forget to copy the executable to the target using RSE. Look back in the lab for a reminder, if needed.**



22. Click the **Run** icon and notice that the program automatically stops at the breakpoint. The console output should say *Hello World from Sitara!*



23. Click the **Step Over** icon and notice that the second printf line which was just added is now executed, so the program on the target file system was updated with your changes.



24. You can now stop the debugger by pressing the **Stop** button.
25. To restart the debugger click the **Bug** icon on the top toolbar to relaunch the application for debug.

Navigation

Return to Processor SDK Linux Software Developer's Guide for additional information.

Archived Versions

- ***Processor SDK Linux 02.00.01.07*** (http://processors.wiki.ti.com/index.php/?title=Processor_SDK_Linux_Training:_Hands_on_with_the_Linux_SDK&oldid=221388)
- ***Processor SDK Linux 01.00.00.03*** (http://processors.wiki.ti.com/index.php/?title=Sitara_Linux_Training:_Hands_on_with_the_SDK&oldid=208944)
- ***Sitara Linux SDK 06.00.00.00*** (http://processors.wiki.ti.com/index.php/?title=Sitara_Linux_Training:_Hands_on_with_the_SDK&oldid=206006)
- SDK 5.07 (http://processors.wiki.ti.com/index.php?title=Sitara_Linux_Training:_Hands_on_with_the_SDK&oldid=150470)

{{ 1. switchcategory:MultiCore= • For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum • For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **Processor SDK Linux Training: Hands on with the Linux SDK** here.	Keystone= • For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum • For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **Processor SDK Linux Training: Hands on with the Linux SDK** here.	C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **Processor SDK Linux Training: Hands on with the Linux SDK** here.*	DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **Processor SDK Linux Training: Hands on with the Linux SDK** here.*	MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **Processor SDK Linux Training: Hands on with the Linux SDK** here.*	OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Processor SDK Linux Training: Hands on with the Linux SDK** here.*	OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Processor SDK Linux Training: Hands on with the Linux SDK** here.*	MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **Processor SDK Linux Training: Hands on with the Linux SDK** here.*	*For technical s please post yo questions at http://e2e.ti.cor Please post on comments abo article **Process SDK Linux Tra Hands on with Linux SDK** her* }}

Links

Amplifiers & Linear	DLP & MEMS	Processors	Switches & Multiplexers
Audio	High-Reliability	• ARM Processors	Temperature Sensors & Control ICs
Broadband RF/IF & Digital Radio	Interface	• Digital Signal Processors (DSP)	Wireless Connectivity
Clocks & Timers	Logic	• Microcontrollers (MCU)	
Data Converters	Power Management	• OMAP Applications Processors	