Developing using C on iMX Developer's Kits

Embedded Artists AB
Rundelsgatan 14
SE-211 36 Malmö
Sweden

http://www.EmbeddedArtists.com

Disclaimer

Embedded Artists AB makes no representation or warranties with respect to the contents hereof and specifically disclaim any implied warranties or merchantability or fitness for any particular purpose. Information in this publication is subject to change without notice and does not represent a commitment on the part of Embedded Artists AB.

Feedback

We appreciate any feedback you may have for improvements on this document. Send your comments by using the contact form: www.embeddedartists.com/contact-us.

Trademarks

All brand and product names mentioned herein are trademarks, services marks, registered trademarks, or registered service marks of their respective owners and should be treated as such.

Table of Contents

1 Document Revision History

Revision	*Date*	*Description*
A	2017-01-10	First release
B	2017-02-15	Added section 4.1 Updates to the Yocto image
C	2021-05-21	Added link to prebuilt toolchain for 64-bit architectures in section 3.1

2 Introduction

When developing applications for Linux you have a large selection of programming languages, editors, development environments, libraries and toolchains to choose from. This document will provide you with instructions for how to get started with application development using the C programming language.

This document is not a course in C programming or Embedded Linux application development. Instead, it will guide you in setting up the tools that exist for building your first "Hello world" application.

If you have never worked with Embedded Linux a recommended course is *bootlin's* "Embedded Linux training". The slides are available for download on their site:

https://bootlin.com/training/embedded-linux/

Additional documentation you might need:

- The *Getting Started* document for the board you are using.
- The *Working with Yocto* document

2.1 Conventions

A number of conventions have been used throughout to help the reader better understand the content of the document.

`Constant width text` – is used for file system paths and command, utility and tool names.

```
$ This field illustrates user input in a terminal running on the
development workstation, i.e., on the workstation where you edit,
configure and build Linux
```

```
# This field illustrates user input on the target hardware, i.e.,
input given to the terminal attached to the COM Board
```

```
This field is used to illustrate example code or excerpt from a
document.
```

This field is used to highlight important information

3 Getting started

The instructions in this section have been tested on a virtual machine running **lubuntu 16.04**. The document "Working with Yocto to build Linux" has a chapter that explains how to create a VMware based virtual machine running lubuntu.

If you are an experienced Linux user it shouldn't be a problem using another Linux distribution with the instructions below as a guideline.

3.1 Install toolchain

To be able to build an application that will run on Embedded Artists iMX based COM boards a toolchain is needed. The toolchain contains cross compiler, linker, and needed libraries.

The toolchain can be built in Yocto using the **meta-toolchain** image. See the document "Working with Yocto to build Linux" for more information about Yocto.

```
$ bitbake meta-toolchain
```

The build will result in a file located at `<build directory>/tmp/deploy/sdk`. The exact name of the file depends on several parameters, but in our example, it is called:

`fsl-imx-fb-glibc-x86_64-meta-toolchain-cortexa9hf-vfp-neon-toolchain-4.1.15-1.2.0.sh`

Part of file name	Description
`fsl-imx-fb`	The distribution (DISTRO parameter) used when initializing the build.
`x86_64`	Architecture of the host computer. In this example a **64-bit** Intel x86 platform
`4.1.15-1.2.0`	BSP version

It is recommended to build this toolchain on your host computer where you will do the development, but if you have a 64-bit Intel x86 based host computer you can download a pre-built version from imx.embeddedartsits.com.

For iMX6 and iMX7 based boards (32-bit architecture)

```
$ wget imx.embeddedartists.com/common/fsl-imx-fb-glibc-x86_64-
meta-toolchain-cortexa9hf-vfp-neon-toolchain-4.1.15-1.2.0.sh
```

For iMX8 based boards (64-bit architecture)

```
$ wget imx.embeddedartists.com/common/fsl-imx-wayland-glibc-
x86_64-meta-toolchain-aarch64-5.4-zeus.sh
```

Install the toolchain. It is recommended to use the default settings (such as installation path) when installing. Replace `<installation file>` below with the name of the file you have downloaded or built.

```
$ chmod a+x <installation file>
$ sudo ./<installation file>
```

3.2 Hello world

To test the toolchain and make sure everything is working a simple "Hello world" application will be developed.

1. Open a terminal application on your host computer (the host computer in our example is lubuntu 16.04)
2. Setup the toolchain environment by running the `source` command below. A file was installed together with the toolchain that contains all environment variables needed to be setup. The instructions below use the default installation path.

```
$ source /opt/fsl-imx-fb/4.1.15-1.2.0/environment-setup-
cortexa9hf-vfp-neon-poky-linux-gnueabi
```

3. You can verify that the environment variables have been correctly setup by running the command below. This command shows the version of the GCC compiler.

```
$ $CC --version
arm-poky-linux-gnueabi-gcc (GCC) 5.2.0
```

4. Create the "Hello world" application using a text editor. In this example we are using `nano`. Create the file and copy the content of the example below to that file.

```
$ cd ~
$ mkdir hello_app
$ cd hello_app
$ nano hello.c
```

```
#include <stdio.h>

int main(int argc, char **argv)
{
        printf("Hello world\n");
        return 0;
}
```

5. Save the file (in `nano` use Ctrl+X followed by Y and Enter)
6. From the terminal where you setup the toolchain environment compile the application. We are giving the application the name "hello" using the –o argument to the compiler.

```
$ $CC -o hello hello.c
```

7. You will now have a file named "hello". Go to the next section for instructions of how to run this application on target.

3.3 Run the application on target

In the previous section the application was built. In this section the application will be copied to the target via a USB memory stick.

1. Connect a USB memory stick to your host computer and copy the file "hello" to that memory stick. No instructions are given here since it normally is only a drag-and-drop procedure.
2. Unmount the USB memory stick from your host computer and insert it into the developer's kit. You should see output in the terminal connected to the target that looks similar to below.

```
usb 1-1.3: new high-speed USB device number 7 using ci_hdrc
usb-storage 1-1.3:1.0: USB Mass Storage device detected
scsi host4: usb-storage 1-1.3:1.0
scsi 4:0:0:0: Direct-Access     SanDisk  U3 Cruzer Micro  2.18 PQ:
0 ANSI: 2
scsi 4:0:0:1: CD-ROM            SanDisk  U3 Cruzer Micro  2.18 PQ:
0 ANSI: 2
sd 4:0:0:0: [sda] 8015505 512-byte logical blocks: (4.10 GB/3.82
GiB)
sd 4:0:0:0: [sda] Write Protect is off
sd 4:0:0:0: [sda] No Caching mode page found
sd 4:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 4:0:0:0: [sda] Attached SCSI removable disk
```

3. The important part is the device name "sda1". Mount the USB memory stick.

```
# mount /dev/sda1 /mnt
```

4. Copy the application to the target. It is assumed that the file was copied to the root of the USB memory stick (in step 1 above). The '~' character means that we are copying the file to the home directory.

```
# cp /mnt/hello ~
```

5. Run the application

```
# cd ~
# ./hello
Hello world
```

6. If you get a "permission denied" message instead of "Hello world" add execution permissions to the application and then run it again

```
# chmod a+x hello
```

4 Eclipse

Eclipse is a popular software development kit that can be used with many different programming languages. This chapter describes how to get started with Eclipse when developing C applications.

4.1 Updates to the Yocto image

The default Yocto images provided by Embedded Artists are missing some functionality needed when following the instructions in this chapter. More specifically it is a GDB server – needed for debugging and a SFTP server – needed when downloading an application to target that are missing.

The servers can be added by modifying the `local.conf` file in your build. See the document "Working with Yocto to build Linux" for more details about building images.

1. Open `local.conf`. Replace `<build dir>` with your build directory.

```
$ nano <build dir>/conf/local.conf
```

2. Find the `IMAGE_INSTALL_append` variable and add the lines below.

```
   gdbserver \
   openssh-sftp-server \
```

3. Save the file and exit the editor: CTRL+X followed by Y and Enter.
4. Now build your image. In this example we are using a "core-image-base" build, but replace this with the image you are building.

```
$ bitbake core-image-base
```

5. When the image has been built don't forget to deploy the image on the target. For more information see the "Working with Yocto" document.

4.2 Install Eclipse

If you haven't already got Eclipse on your host computer follow these instructions to install it. Please note that we are using lubuntu 16.04 when writing these instructions.

NOTE: It is Eclipse **3.8.1** that was installed when writing these instructions. If you have another version of Eclipse there could be minor differences.

1. Eclipse can be installed using `apt-get`

```
$ sudo apt-get install eclipse
```

2. Answer Y to any question. It takes a couple of minutes to install eclipse.
3. Since we are doing C development we also need to install CDT (C/C++ Development Tooling)

```
$ sudo apt-get install eclipse-cdt
```

4. To be able to connect to the target from within Eclipse we are going to use a plugin called Remote System Explorer.

```
$ sudo apt-get install eclipse-rse
```

4.3 Create and configure a project

Start Eclipse

Start Eclipse either from the menu (normally under "Programming") or by writing `eclipse` in a terminal window.

```
$ eclipse &
```

You will be asked to select a workspace as shown in Figure 1.

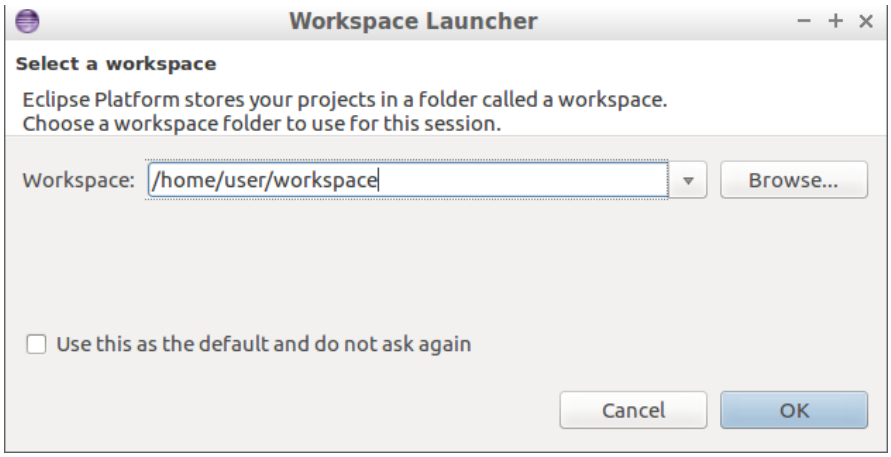

Figure 1 - Select a workspace

Click Ok and if the workspace is new or has no projects you will be presented with the "Welcome screen" that can look as shown in Figure 2. Click on the "Workbench" button at the bottom of the screen.

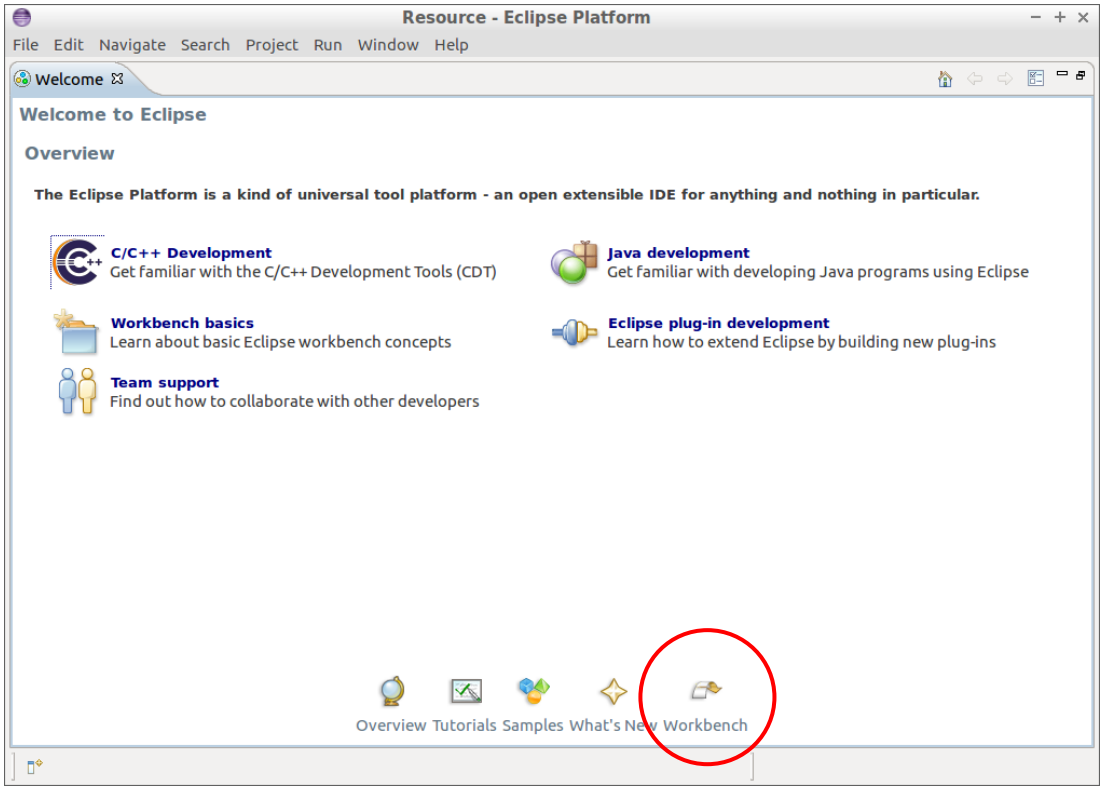

Figure 2 - Eclipse welcome screen

Create a project

Create a new project by going to File → New → Project in the menu. Choose a "C Project" as shown in Figure 3 and then click "Next".

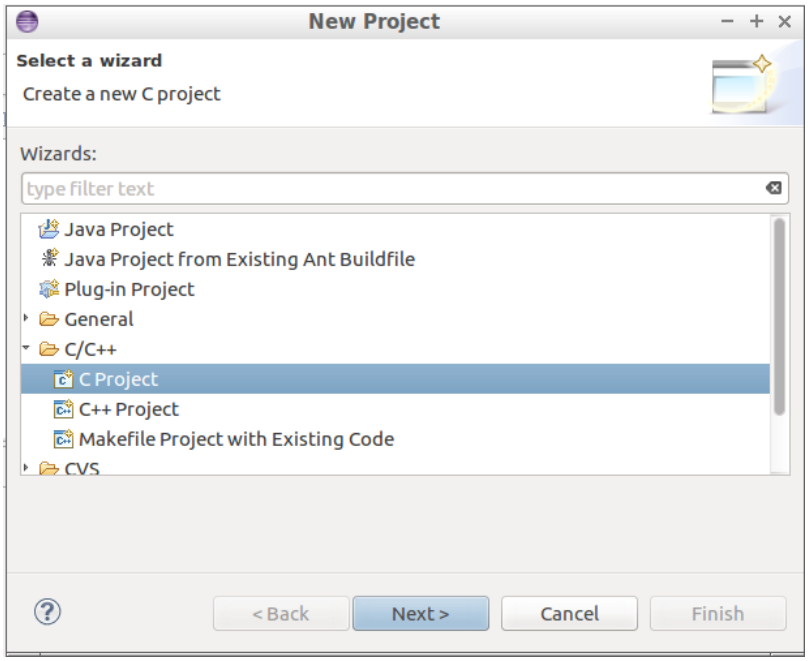

Figure 3 - New Project wizard

Enter a project name ("hello" in this example). Select project type as an "Empty project" and set toolchain to be "Cross GCC" as shown in Figure 4.

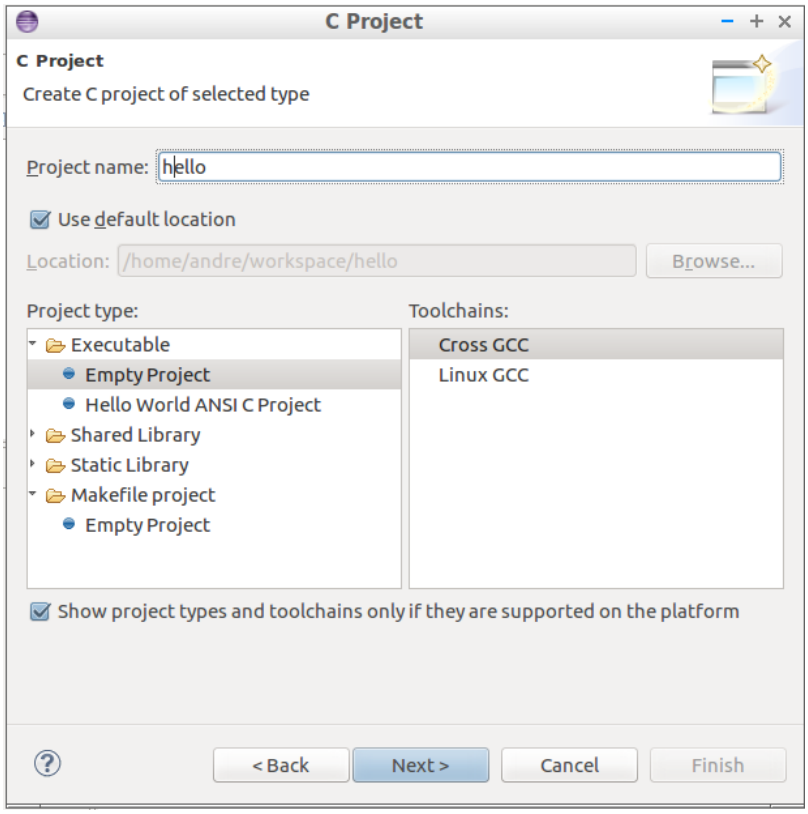

Figure 4 - Project name and type

Click "Next" and then just use the default settings in the "Select configurations" dialog as shown in Figure 5.

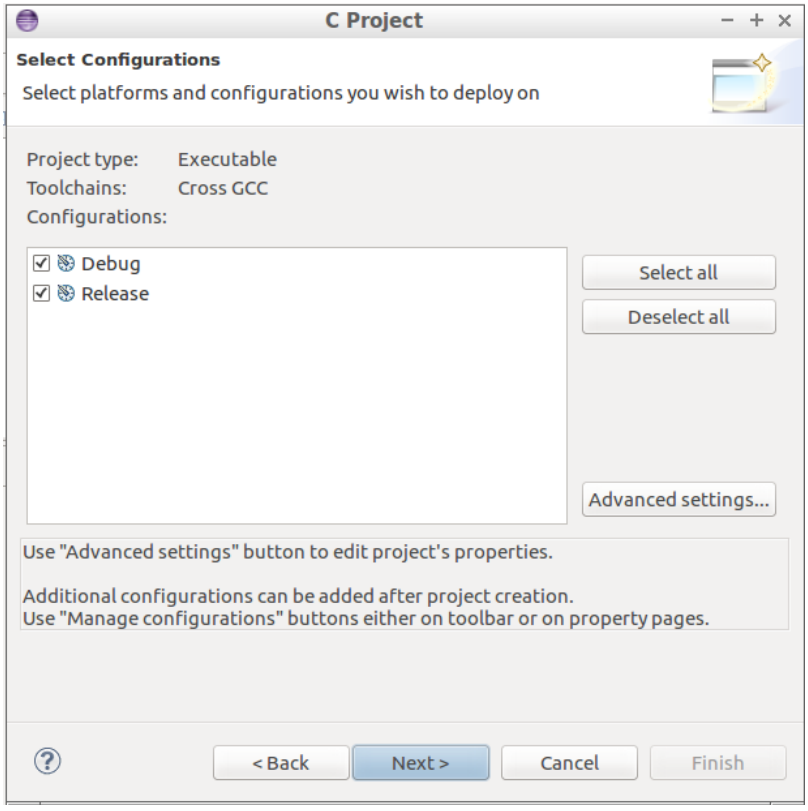

Figure 5 - Select configurations

Click "Next". Now it is time to set the path to the cross-compiler. Set the prefix to `arm-poky-linux-gnueabi-`. Set the path to (change if you have installed the toolchain elsewhere) `/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi`.

When you are done click the "Finish" button as shown in Figure 6.

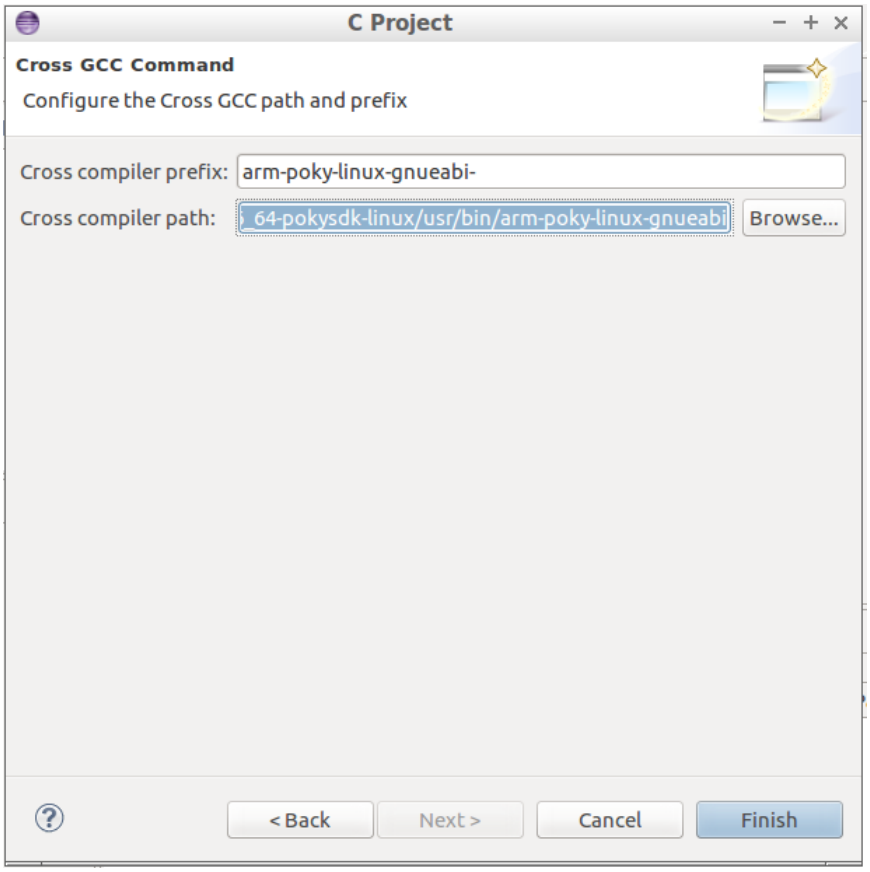

Figure 6 - Cross compiler

Create the application

Create a new file by going to File → New → Source File in the menu. Enter a file name as shown in Figure 7 (`hello.c` in this example) and then click "Finish".

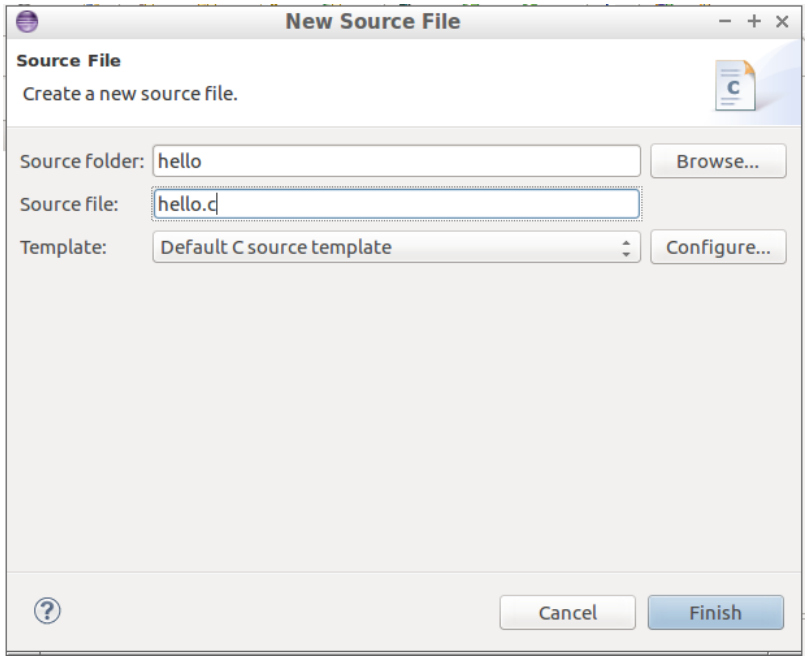

Figure 7 - New source file

Copy the "Hello world" application (source code) from section 3.2 to the newly created file as shown in Figure 8.

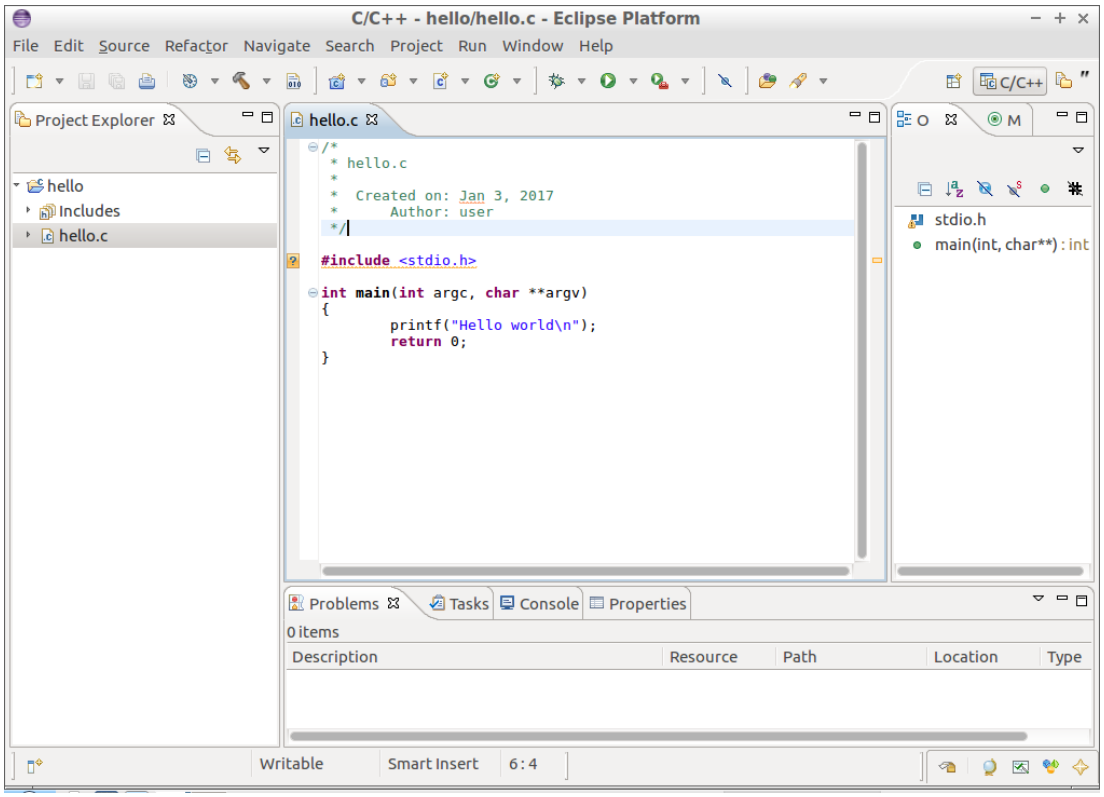

Figure 8 - Hello world application

Configure the project

It is now time to configure the project. The path to "sysroot" is needed in several places so the first step is to create an environment variable specifying this path.

Go to Project → Properties in the menu and then C/C++ Build → Environment. Click the "Add" button and create a variable named `SDKTARGETSYSROOT` with the path `/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi` as shown in Figure 9.

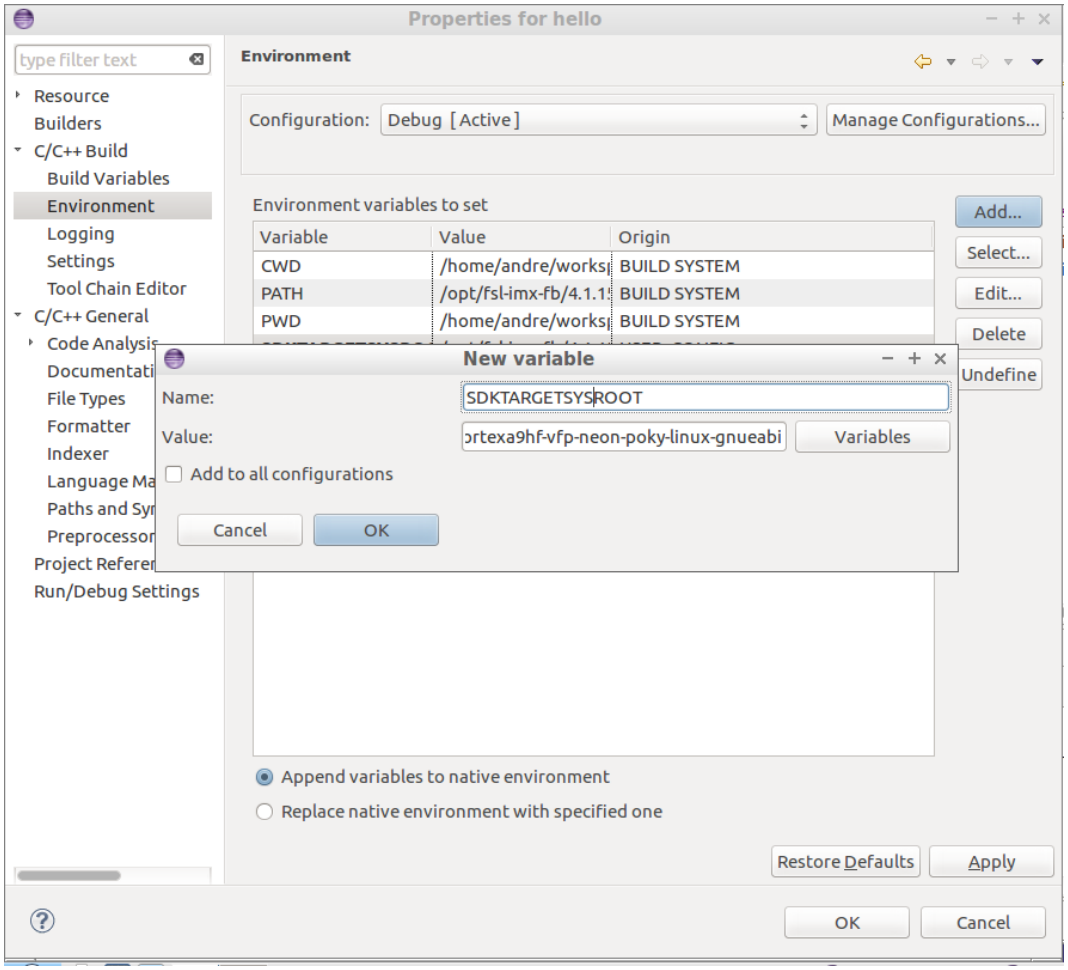

Figure 9 - Sysroot as environment variable

A number of compiler options must be specified in order to correctly compile the application. Go to C/C++ Build → Settings and then click on Cross GCC Compiler →Miscellaneous. Add the line below to the "Other flags" field in this window as shown in Figure 10.

```
-march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --
sysroot=${SDKTARGETSYSROOT}
```

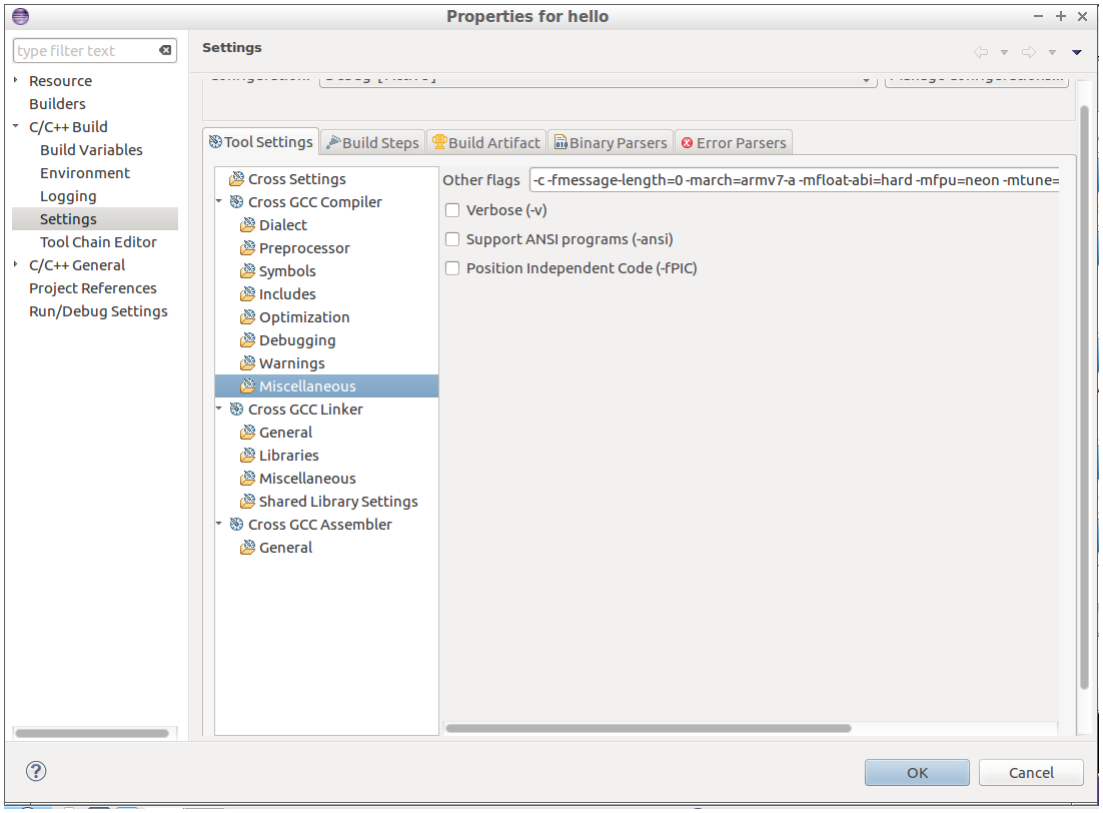

Figure 10 - Compiler options

Options must also be given to the linker. Go to C/C++ Build → Settings and then Cross GCC Linker →Miscellaneous. Add the line below to the "Linker flags" field as shown in Figure 11.

```
--sysroot=${SDKTARGETSYSROOT} -mfloat-abi=hard
```

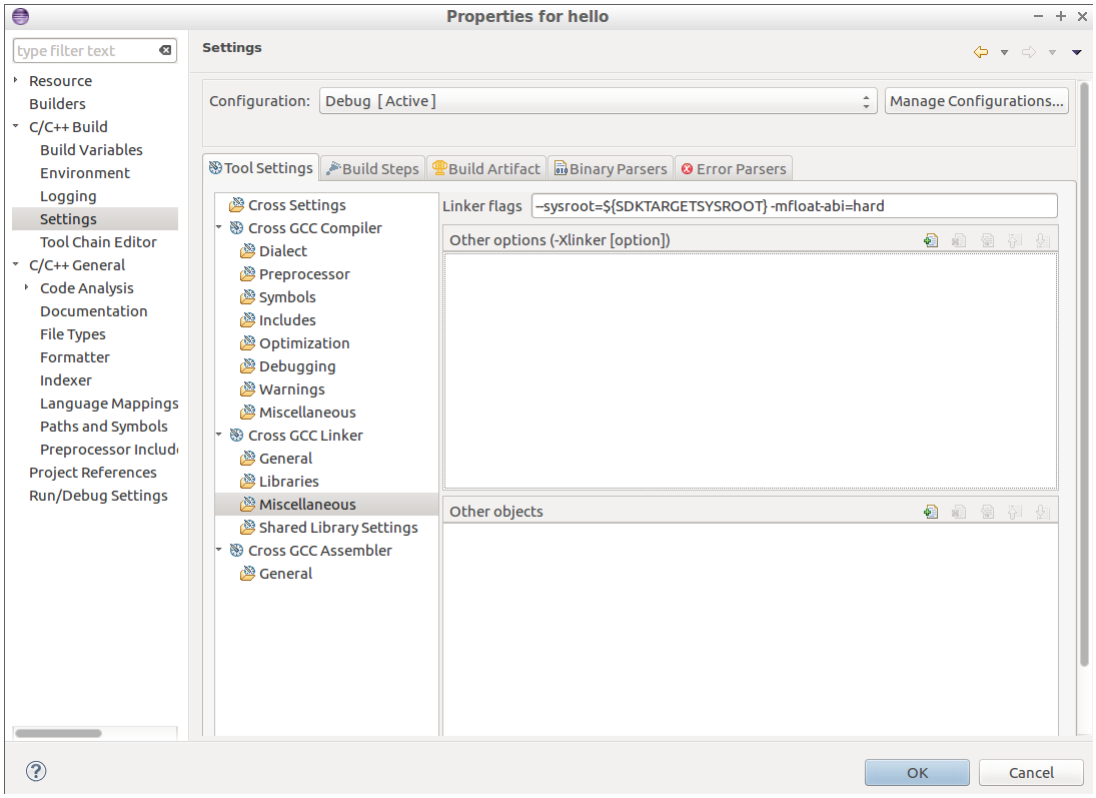

Figure 11 - Linker options

When creating the application you might have noticed that `stdio.h` was underlined (see Figure 8). Holding the mouse cursor above the question mark shows you that `stdio.h` cannot be found. We need to add a path to the "header files" to get rid of this warning. Go to C/C++ General → Paths and Symbols and then add the directory `${SDKTARGETSYSROOT}/usr/include`. Click the "Apply" button and then "OK".

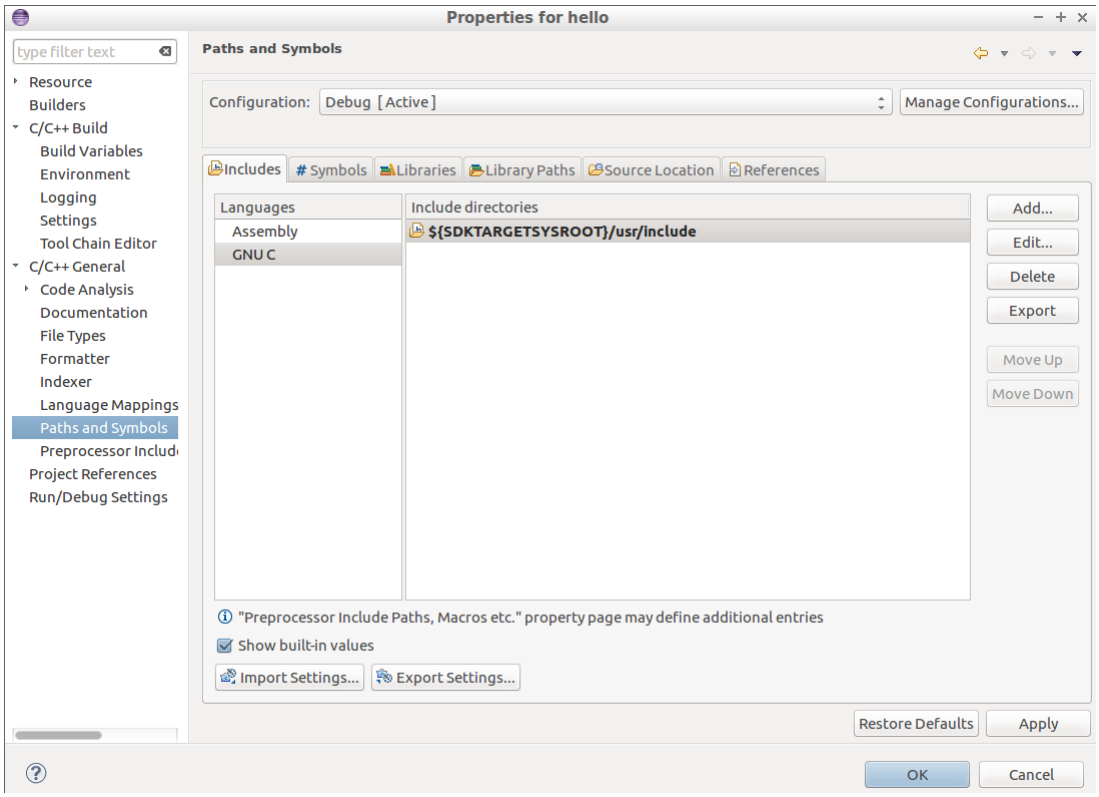

Figure 12 - Path to headers

Build the application

Now it is time to build the application. Right-click on the project ("hello") and then select "Build Project" as shown in Figure 13 below.

You can see the output of the build in the "Console window" as shown in Figure 14.

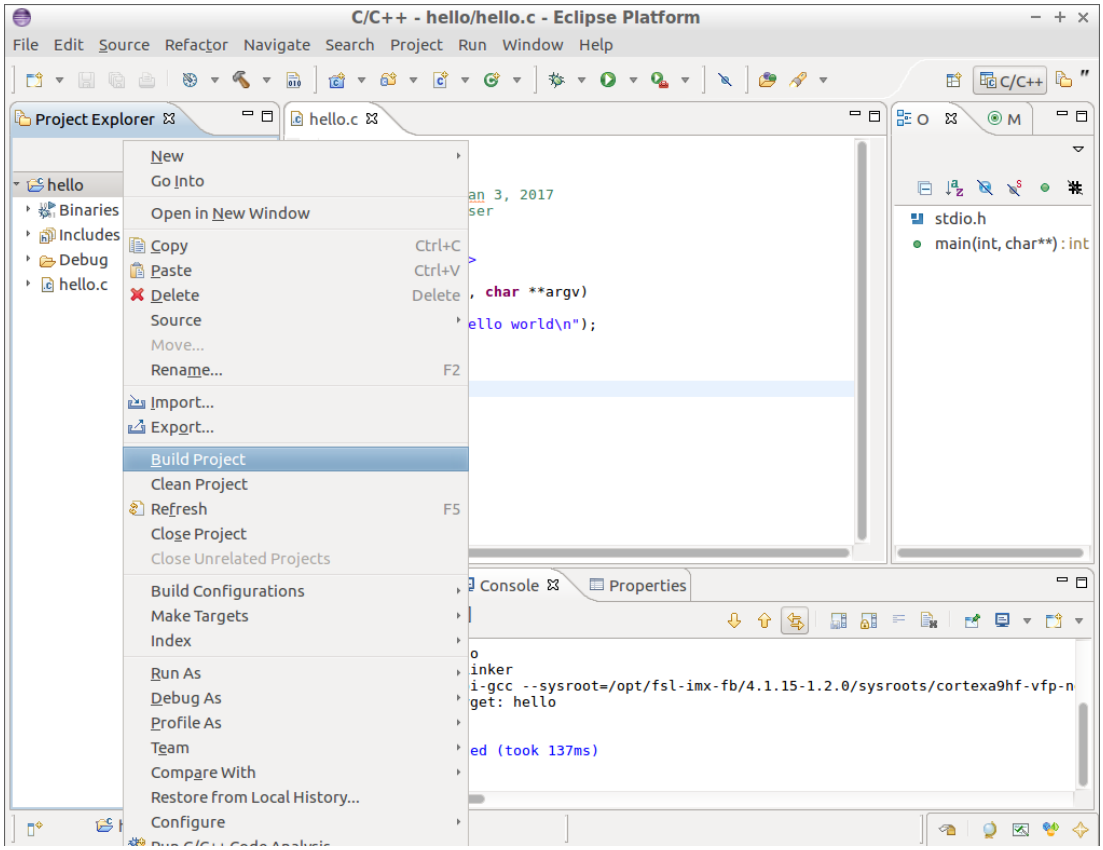

Figure 13 - Build Project

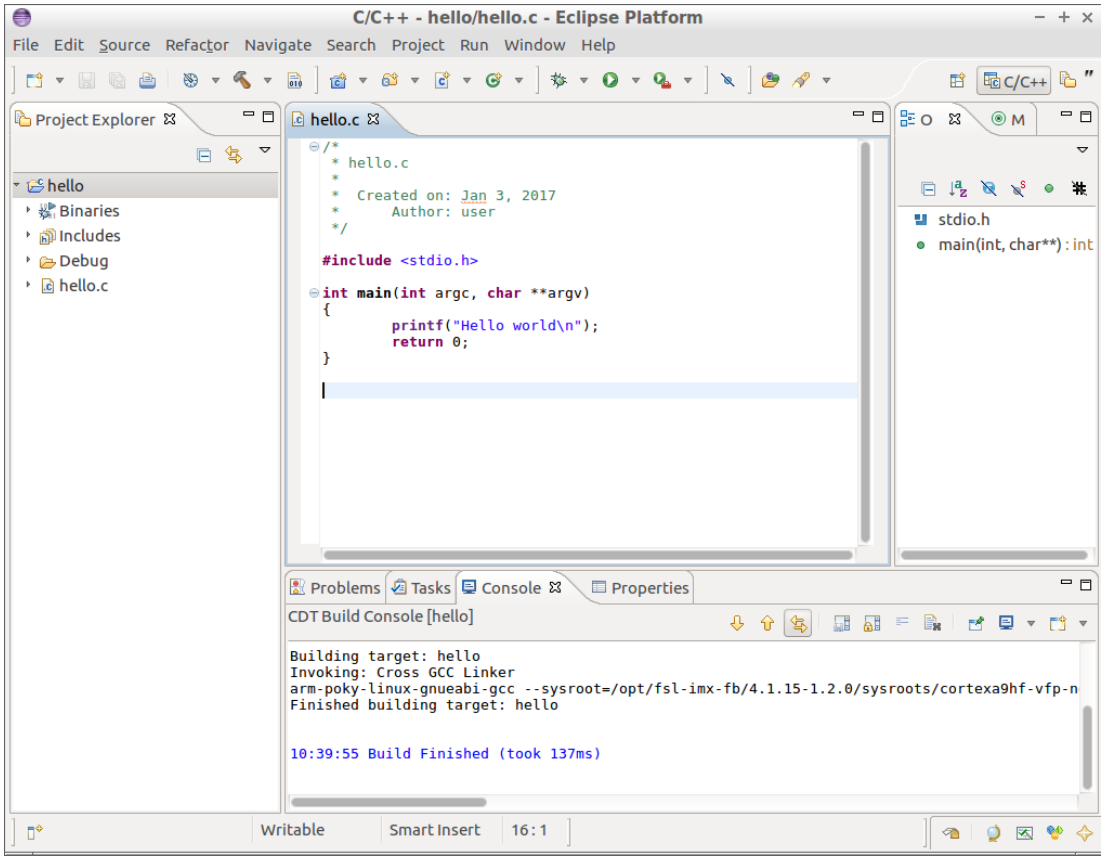

Figure 14 - Console output from a build

4.4 Run the application on target

The application is now located in your workspace under the `Debug` directory. For these instructions that would be in the directory `/home/user/workspace/hello/Debug/`. You can use the same instructions as in section 3.3 to copy the application to a USB memory stick and then to the target.

Another alternative is to use the plugin "Remote System Explorer", but first it must be configured.

Open the Remote System Explorer Perspective

Go to Window → Open Perspective → Other as shown in Figure 15.

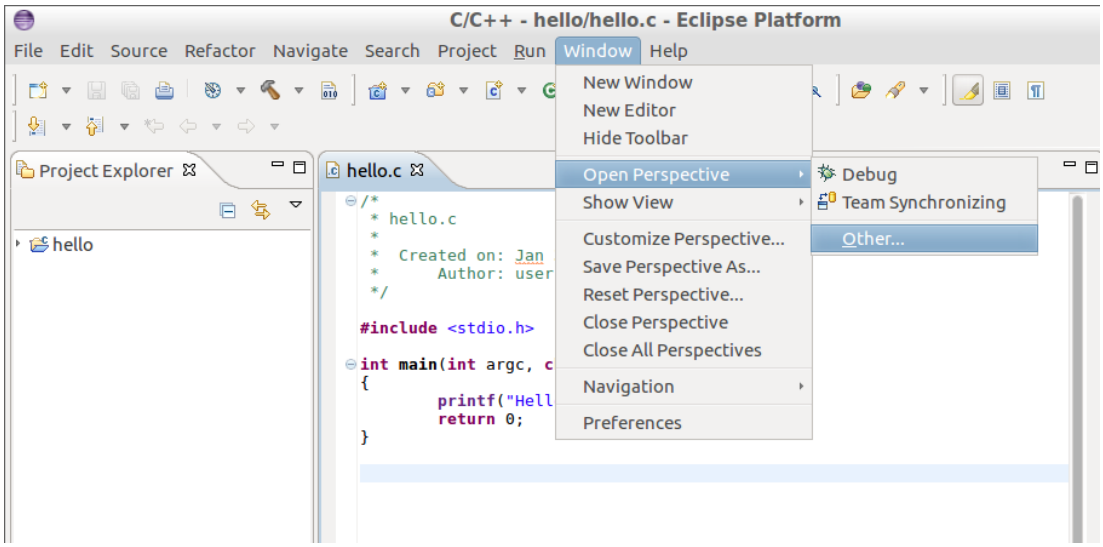

Figure 15 - Change perspective

Select "Remote System Explorer" as shown in Figure 16.

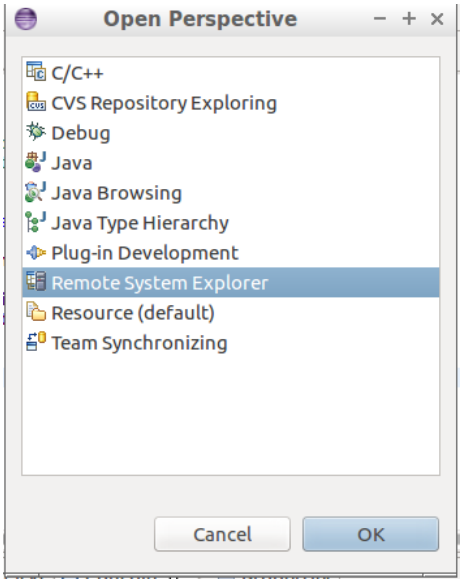

Figure 16 - Select Remote System Perspective

Create connection to remote system

Now it is time to configure the connection to the target. Click on the icon shown in Figure 17.

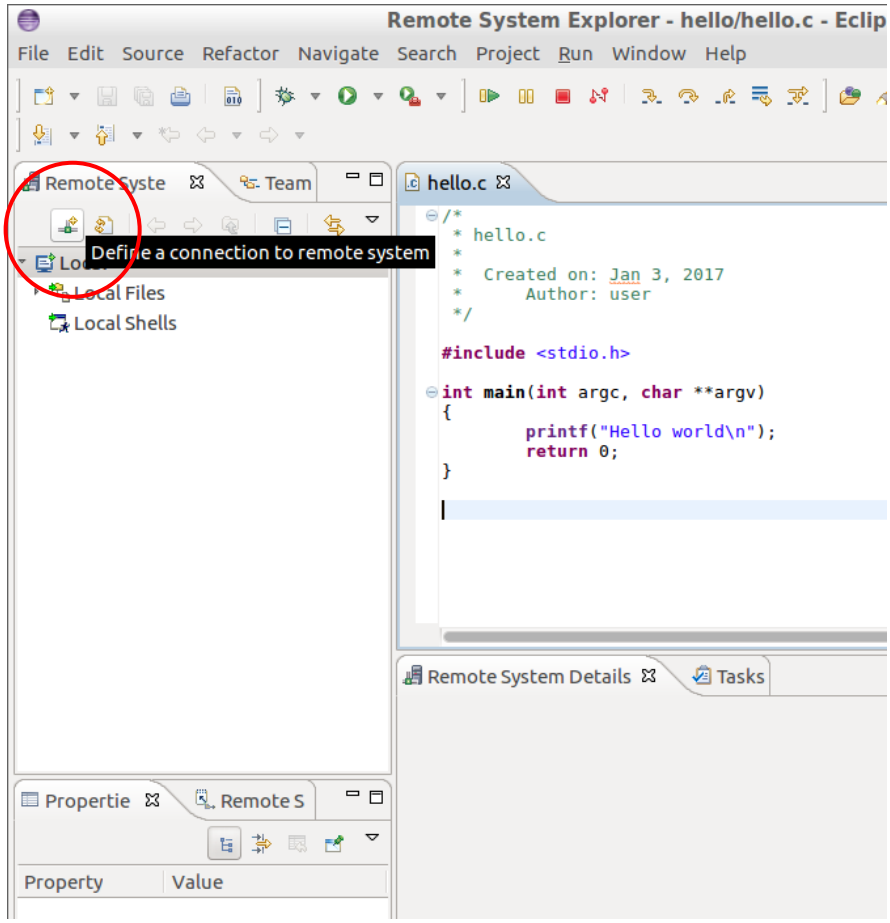

Figure 17 - Create connection to remote system

Select "Linux" as the remote system type as shown in Figure 18.

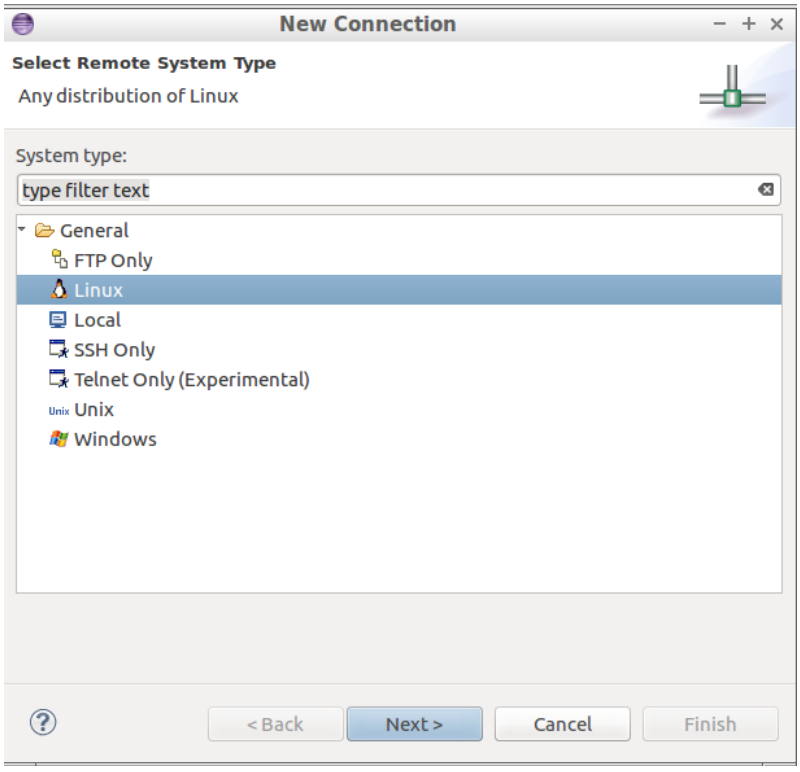

Figure 18 – Remote system type

Specify the IP address of the remote target and give it a description as shown in Figure 19. Please note that the IP address will most likely be different on your target.

You can get the IP address of the target by using the `ifconfig` command in a terminal attached to the target.

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr CA:71:64:BD:1A:20
          inet addr:192.168.1.130  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80:
```

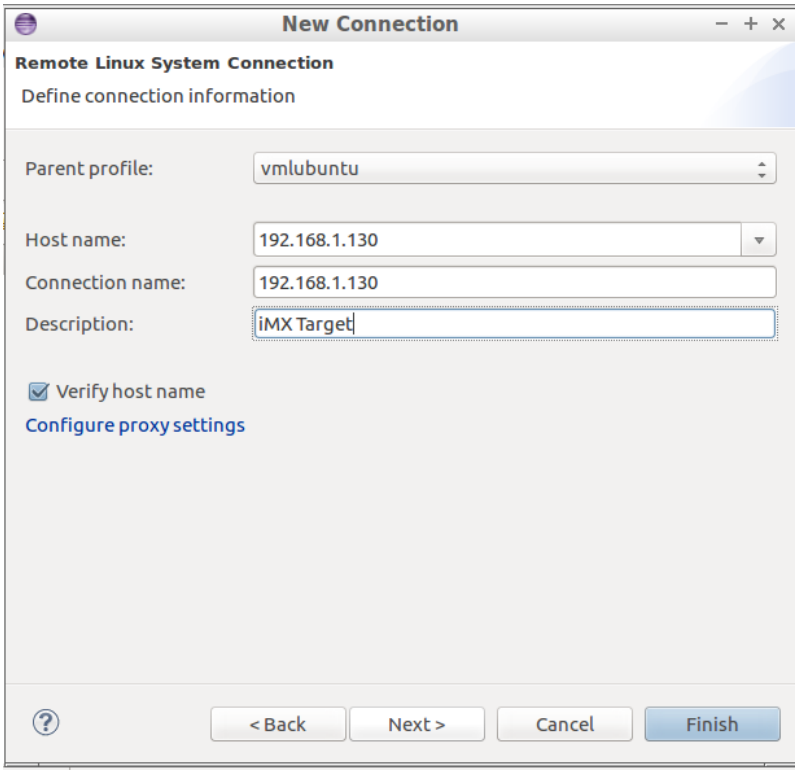

Figure 19 - Host name (IP address)

Go through the wizard (click the "Next" button) and choose "ssh"-related settings as shown in Figure 20, Figure 21, Figure 22, and Figure 23.

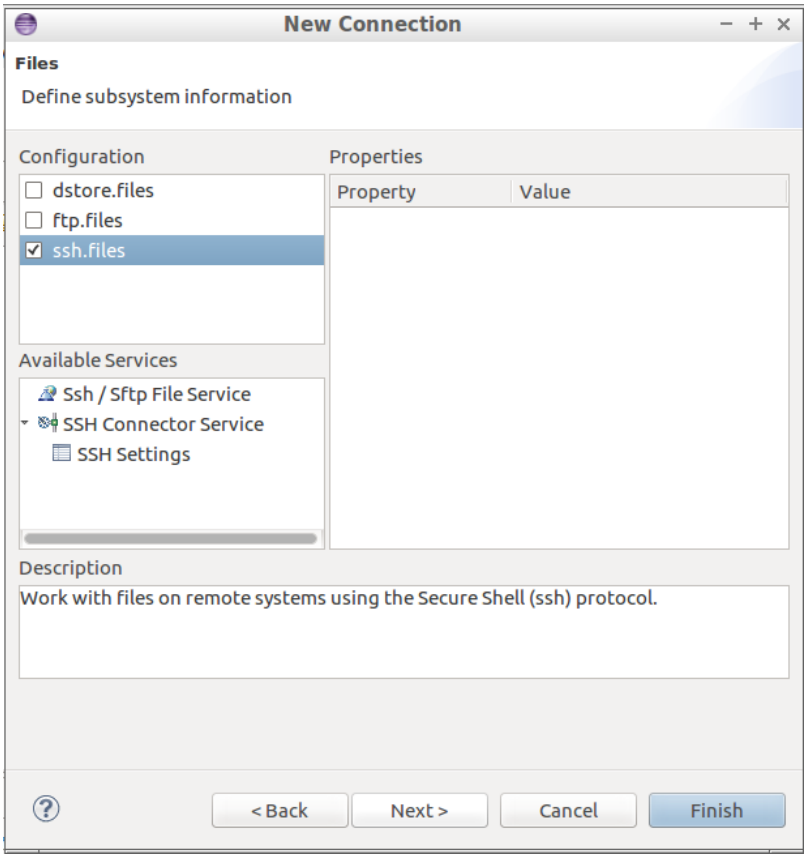

Figure 20 - Subsystem information part 1

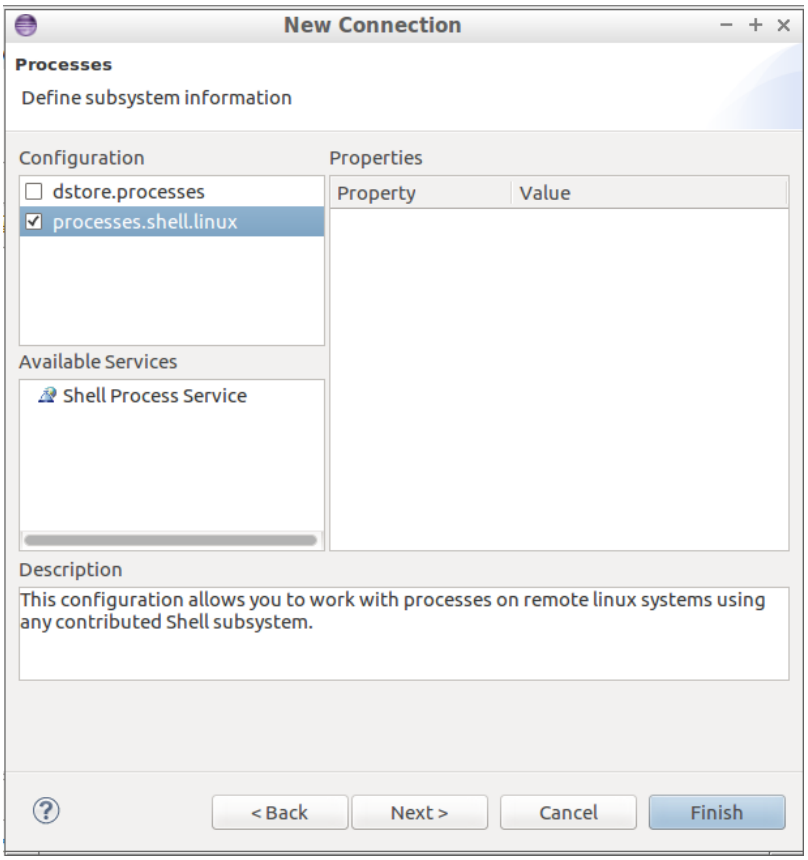

Figure 21 - Subsystem information part 2

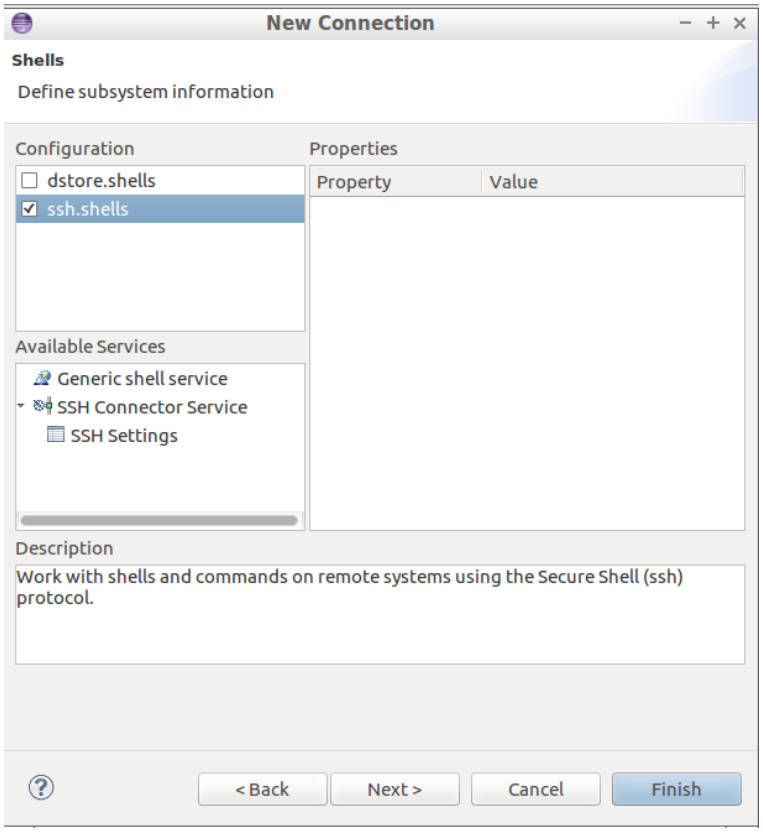

Figure 22 - Subsystem information part 3

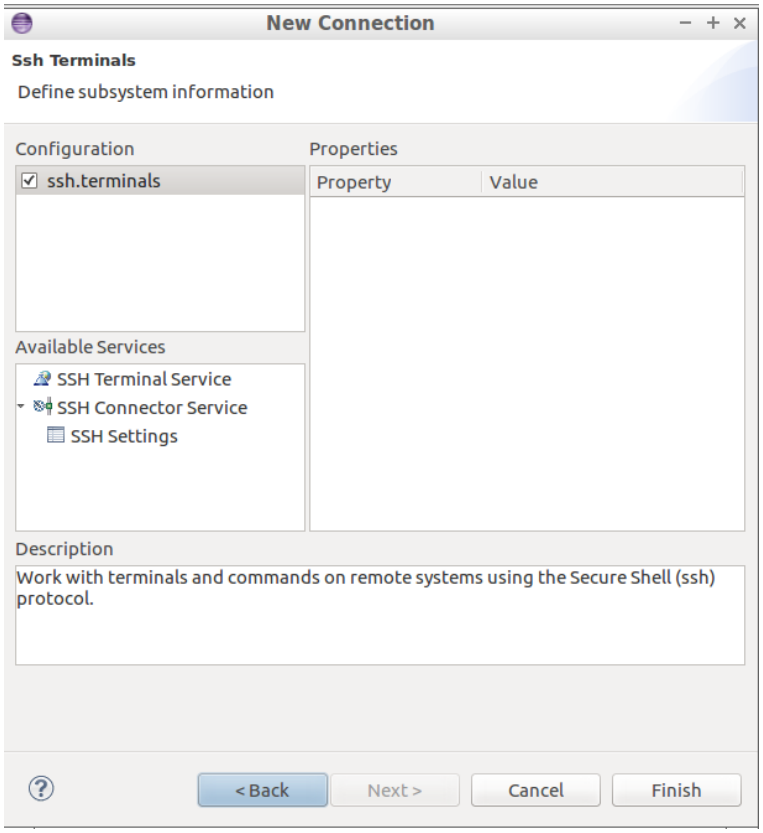

Figure 23 - Subsystem information part 4

Click Finish. The last step before establishing the connection is to specify which user that should login. Right-click on the connection and select "Properties" as shown in Figure 24.

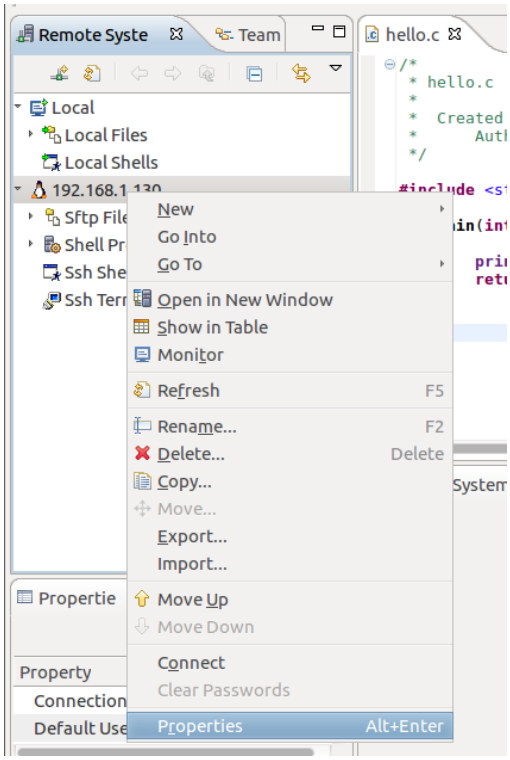

Figure 24 - Connection properties

Select "Host" and then change the "Default User ID" to "root" (or another user if you want to login with a different user) as shown in Figure 25.

NOTE: By default, the user "root" is not permitted to use an SSH connection. See section 5.1 how to permit the user "root" to login.

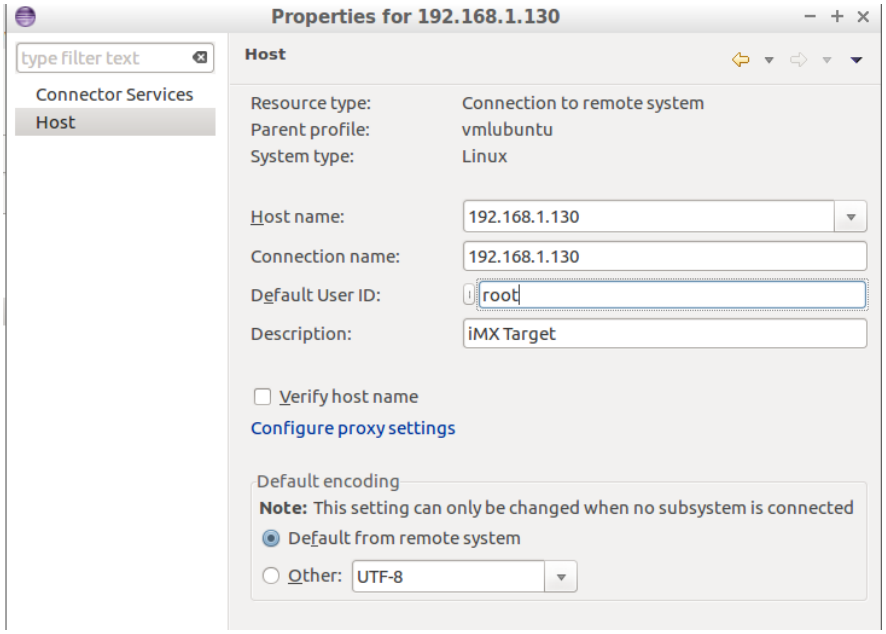

Figure 25 – User ID

Connect to the target by right-click and select "Connect" as shown in Figure 26.

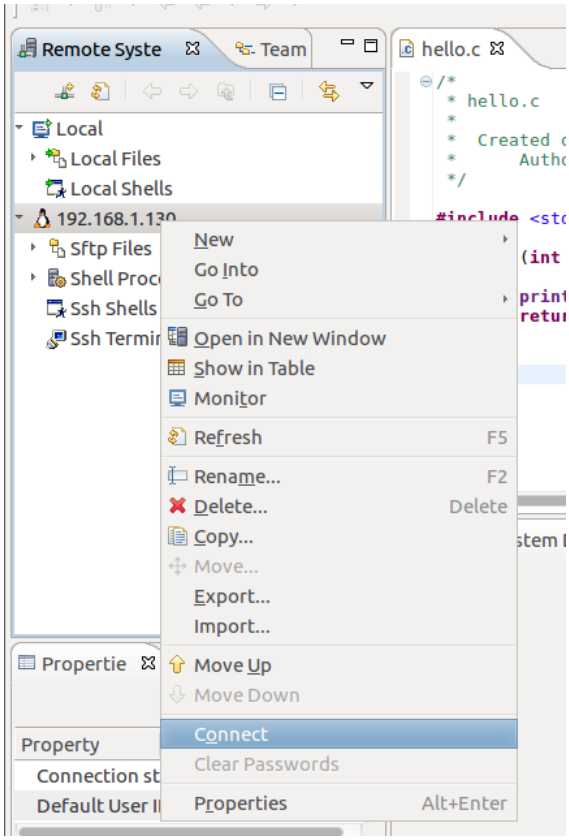

Figure 26 - Connect to target

If you are using the user "root" when logging in the default password is "pass".

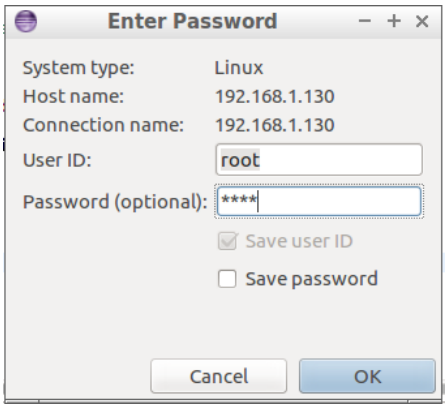

Figure 27 - Enter password

Copy application to target

Go to the location of the compiled application. You will find it under Local Files → My Home → workspace → hello → Debug as shown in Figure 28.

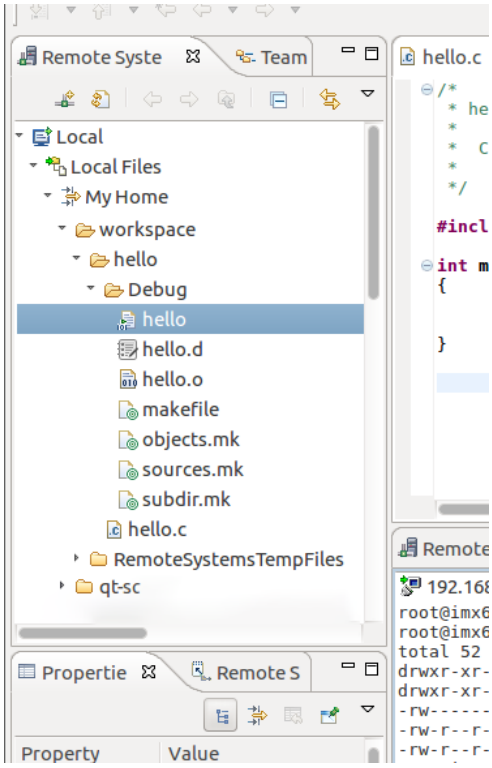

Figure 28 - Local files

Right-click on the application file (`hello`) and select "Copy".

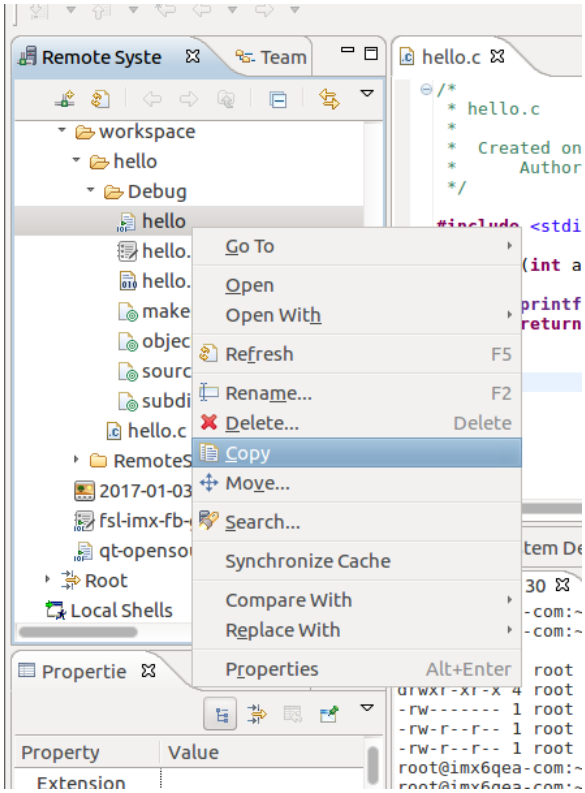

Figure 29 - Copy application

Go to the remote system and paste the application under Sftp Files → My Home as shown in Figure 30.

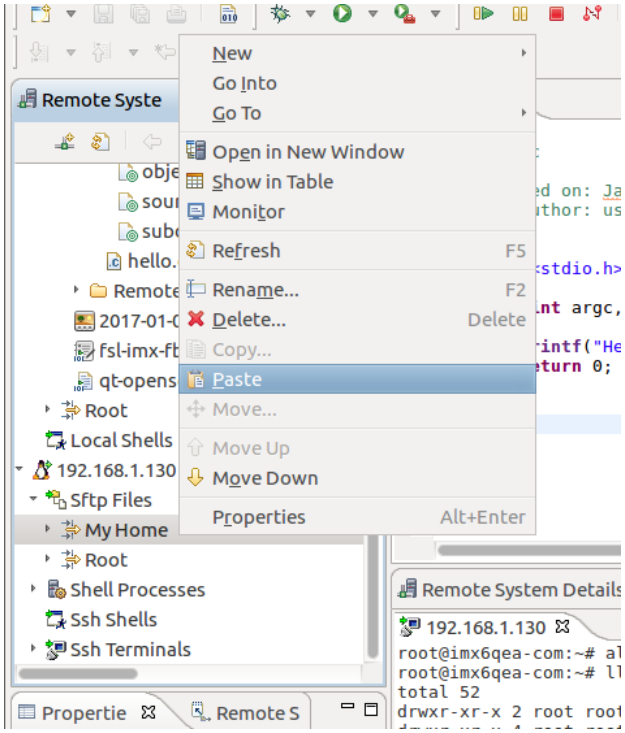

Figure 30 - Copy/paste application to target

Figure 31 shows how it looks when the application has been copied to the target.

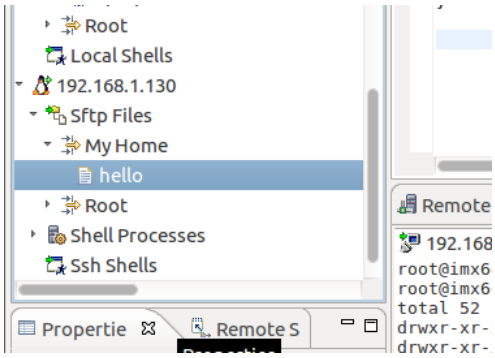

Figure 31 - Application file on the target

Start the application

To start the application we first need to start an SSH terminal. Right-click on "Ssh Terminals" and select "Launch Terminal" as shown in Figure 32.

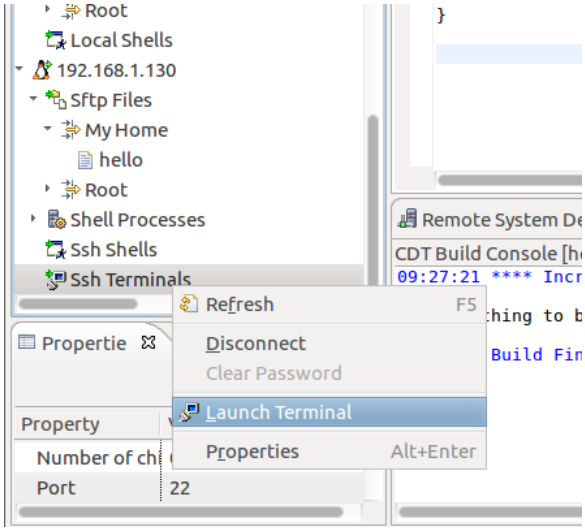

Figure 32 - Launch a terminal

Figure 33 shows the terminal window.

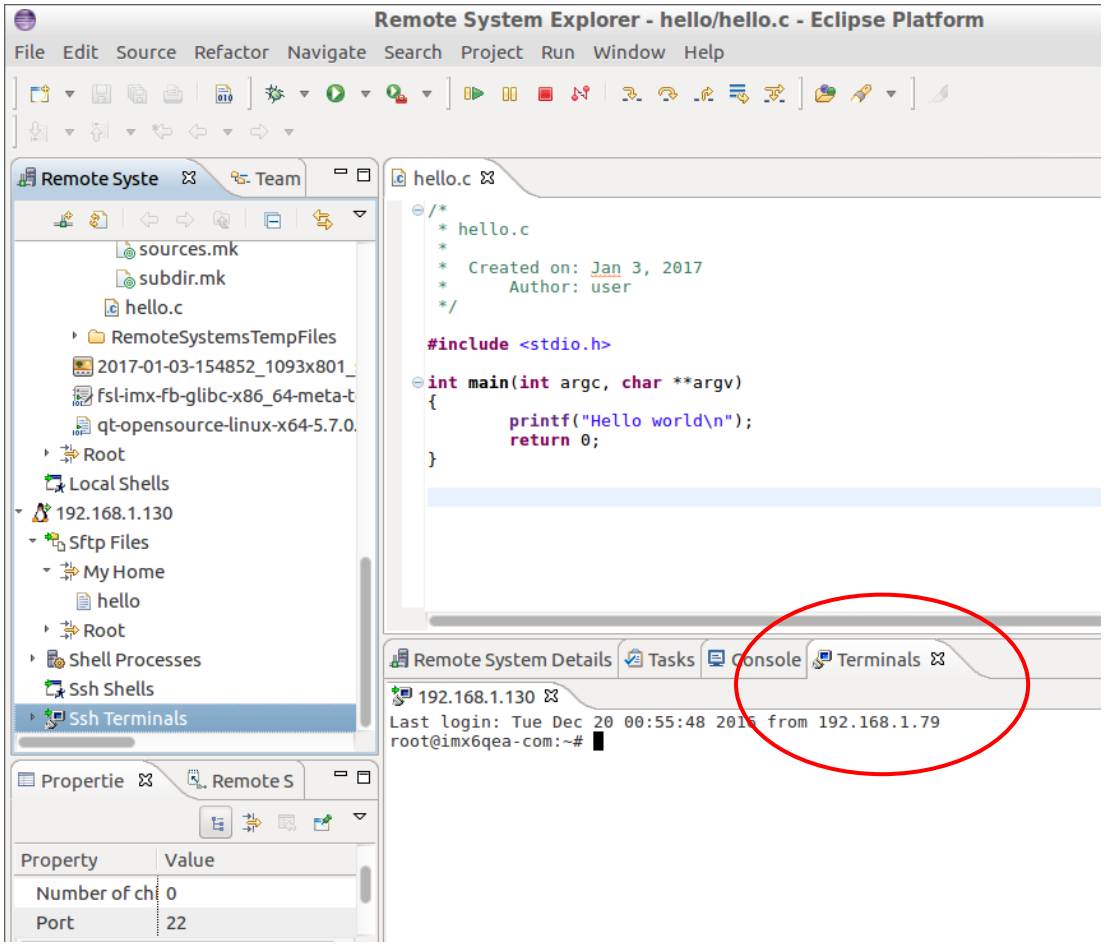

Figure 33 - Terminal

Set execution permissions on the application file

```
# chmod a+x hello
```

Run the application

```
# ./hello
```

All of the above is shown in Figure 34.

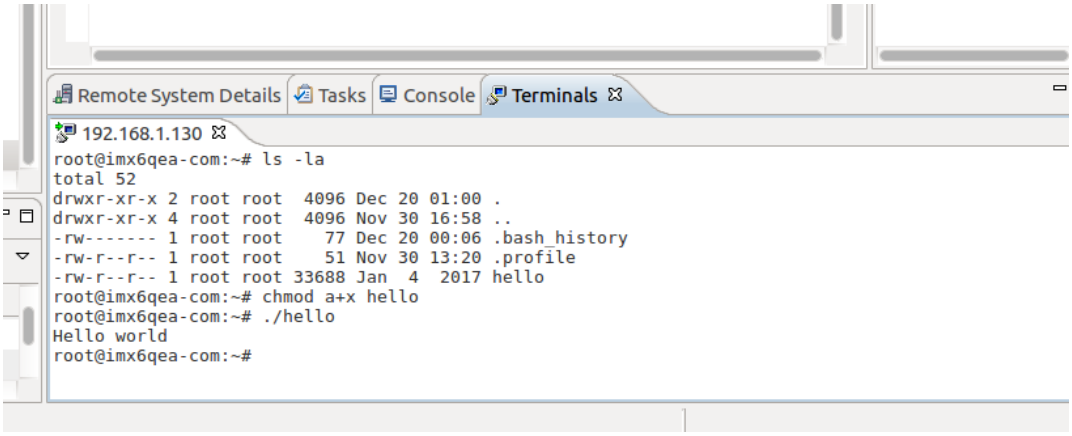

Figure 34 - Run application on target

4.5 Debug the application

For more complicated applications it is really useful to being able to debug the application, that is, single step through the code and inspect variables. This section describes how to debug your application from Eclipse using GDB.

First create the GDB command file (`.gdbint`) in the project directory. You can do this by right-clicking on the project and then New → File.

We need to set the path to the sysroot in this file in order for GDB to load shared libraries. The path should be the same as set in the `SDKTARGETSYSROOT` environment variable. Add the line below to the file.

```
set sysroot /opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi
```

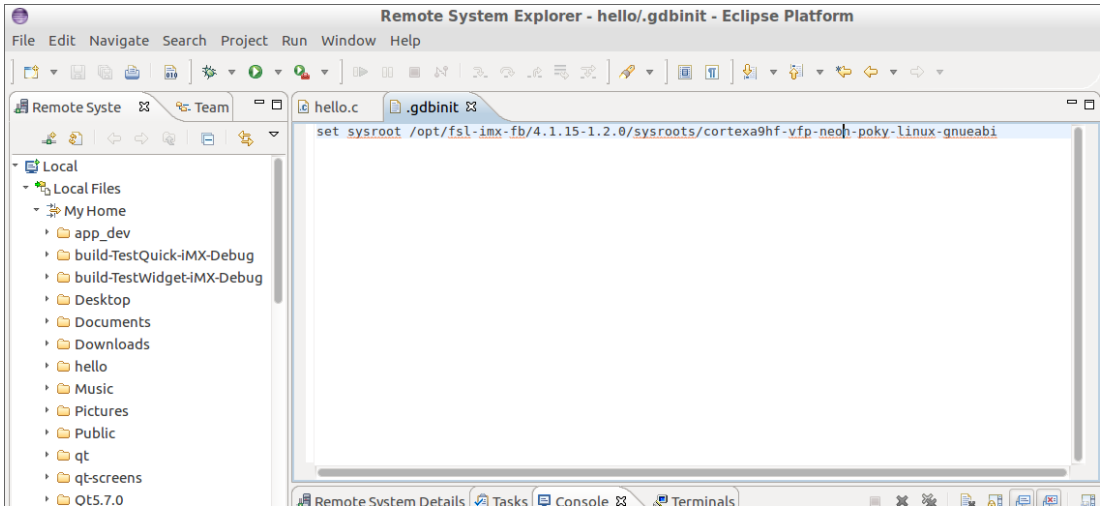

Figure 35 - GDB command file

Go to Run → Debug Configurations in the menu and then right-click on "C/C++ Remote Application" and select "New" as shown in Figure 36.

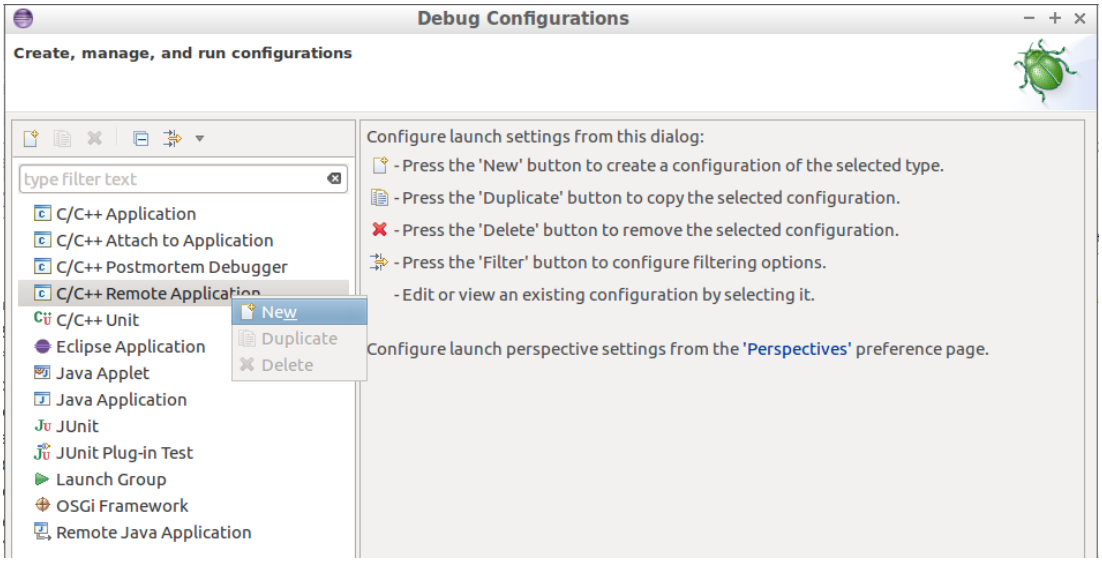

Figure 36 - New debug configuration

On the "Main" tab the name of the project is specified as shown in Figure 37.

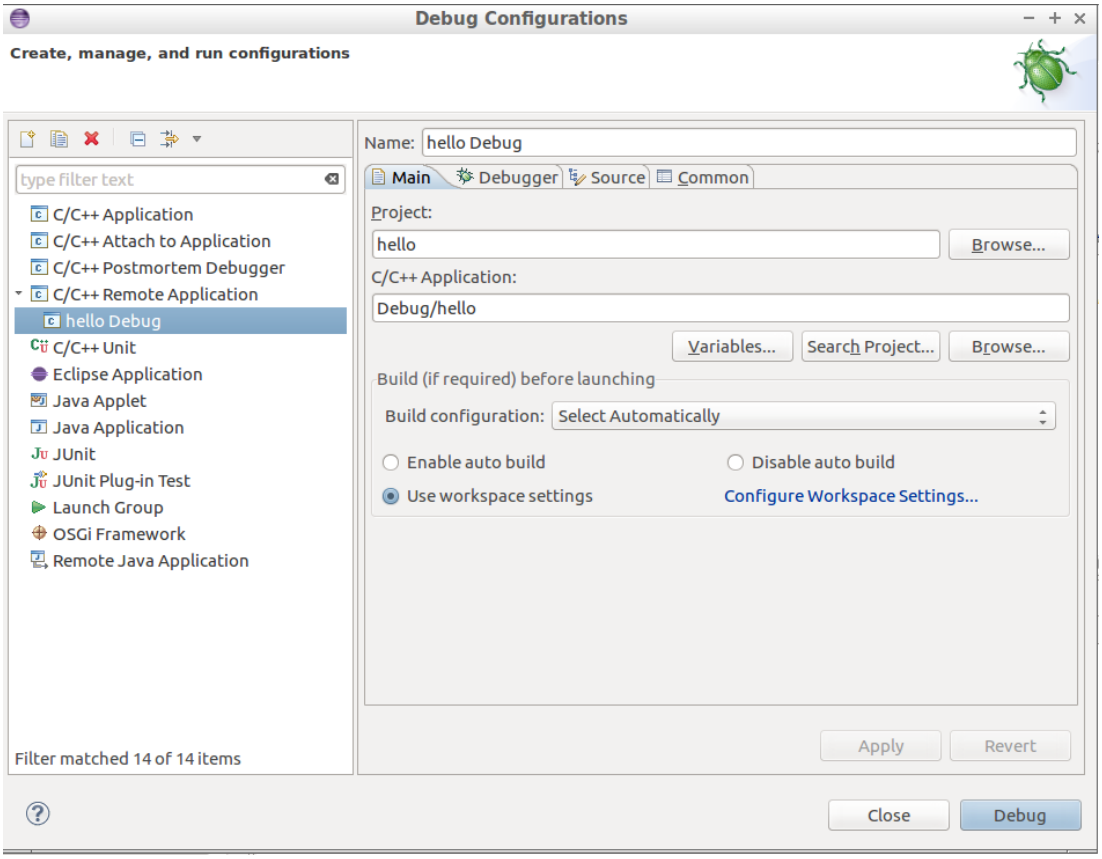

Figure 37 - Debug configuration, Main tab

Go to the "Debugger" tab and set the path to the GDB debugger and GDB command file.

```
/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-
linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb
```

```
/home/user/workspace/hello/.gdbinit
```

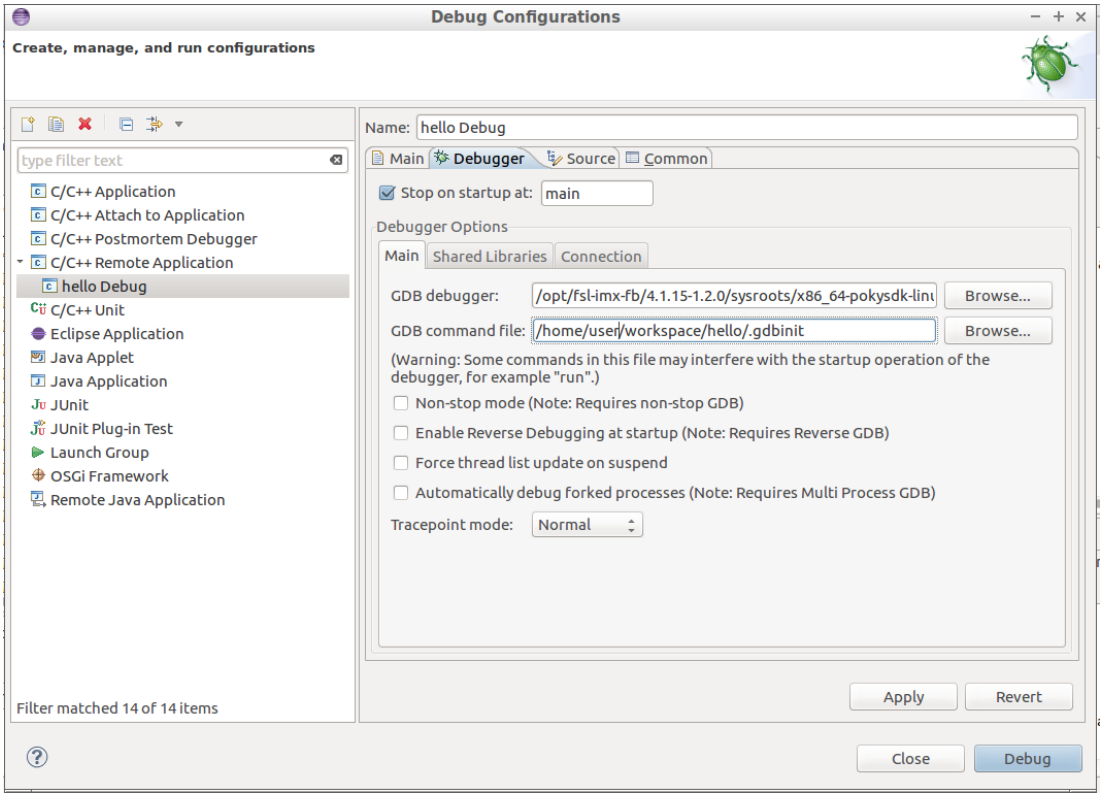

Figure 38 -Debug configuration, Debugger tab

Within the "Debugger" tab select "Connection". Enter the IP address of the target and port number as show in Figure 39. The port number is used in the next step and is by default set to 10000.

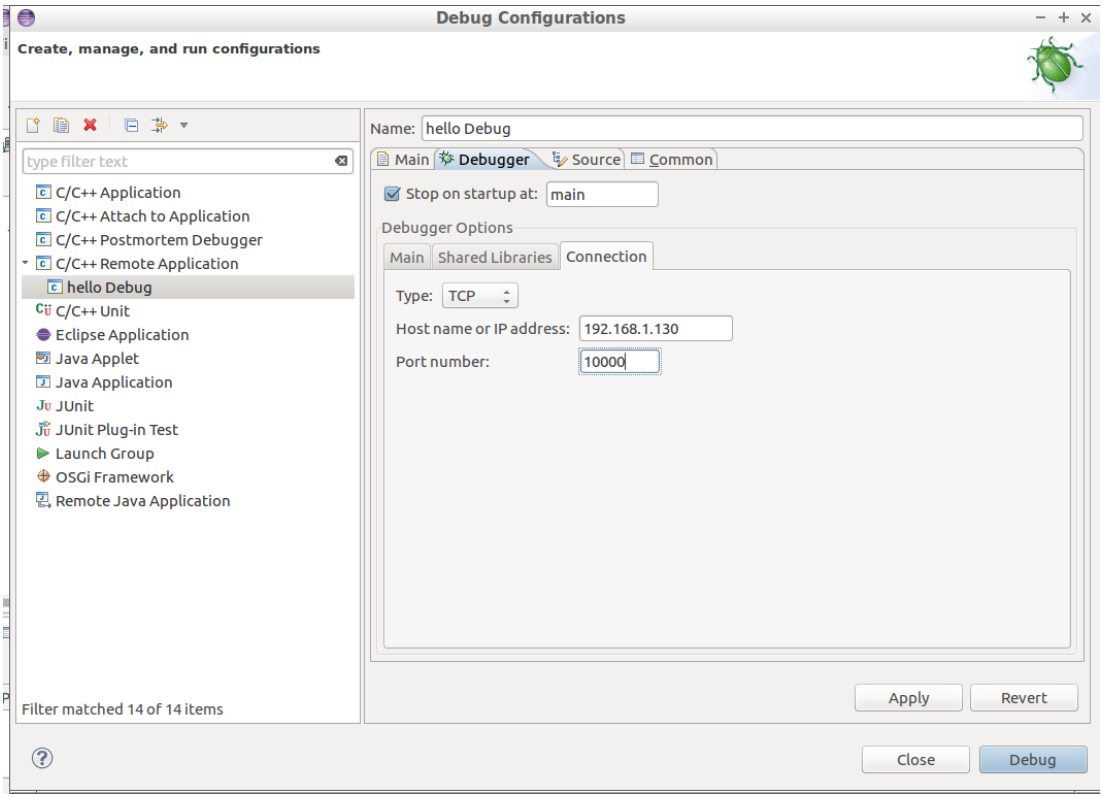

Figure 39 - Connection settings for debug configuration

Go back to the SSH terminal and start a GDB server on port 10000 debugging the application "hello" as shown in Figure 40.

```
# gdbserver :10000 hello
```

Figure 40 - GDB Server

Now it is possible to start to debug the application. Click on the debug icon and then "hello Debug" as shown in Figure 41.

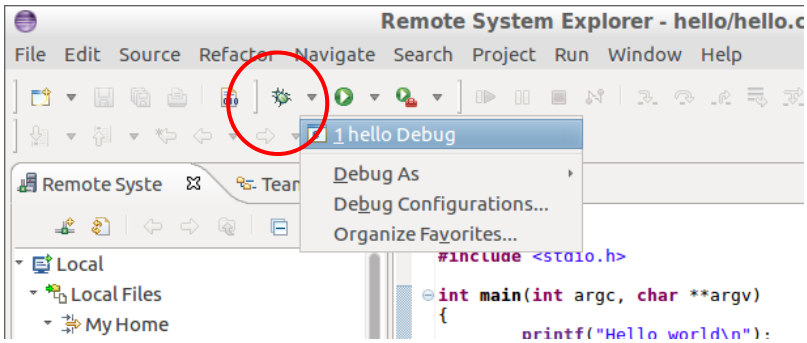

Figure 41 - Start a debug session

If you are asked to change perspective click the "Yes" button as shown Figure 42.

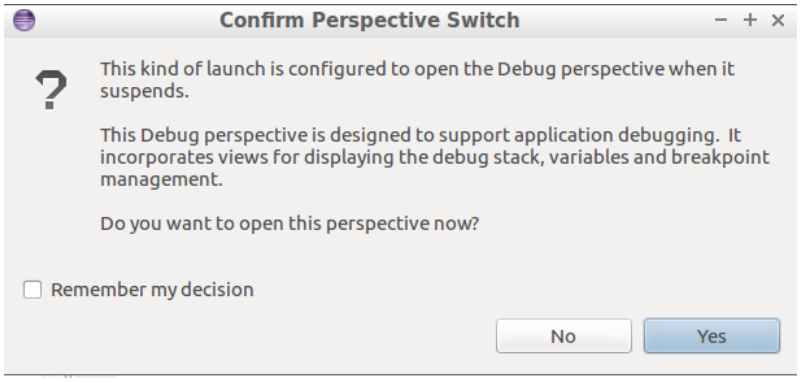

Figure 42 - Change perspective

A debug session is now started and will break at the main function as shown in Figure 43.

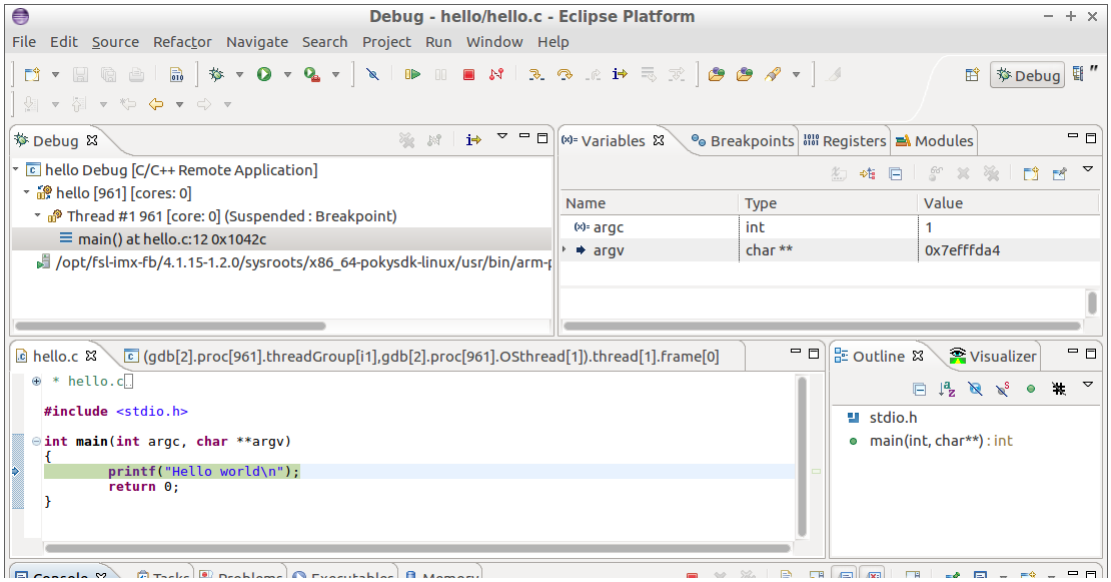

Figure 43 - Debug session

5 Troubleshooting

5.1 Allow user "root" to use an SSH connection

By default, the user "root" is not permitted to login via an SSH connection. By following these instructions "root" will be permitted to login through an SSH connection. It is, however, not recommended to use on a final application, but during development it can be permitted.

1. Open the configuration file for the SSH server

```
# nano /etc/ssh/sshd_config
```

2. Find the line that starts with #PermitRootLogin and remove the '#' (hash) character. If you cannot find this line just add it to the file (without the hash)

```
PermitRootLogin yes
```

3. Save the file and exit the editor (in `nano` it is Ctrl-X followed by Y and Enter).
4. Restart the SSH server

```
# /etc/init.d/sshd restart
```