



“Jožef Stefan” Institute



Department of Communication Systems



SensorLab

SensorLab VESNA open source development environment setup manual for Linux based development

version 0.12

by Zoltan Padrah, Tomaž Šolc

November 10, 2016

1 Introduction

This document describes the installation of a cross-compiler and associated development tools required to compile and debug software for VESNA wireless sensor node on a GNU/Linux operating system.

It offers two distinct approaches to development: one that focuses on graphical interaction and an integrated development environment (IDE) and one focused on command-line interfaces and scripting. A short overview of both is given below - they are identical in regard to functionality and it is up to the reader to choose the approach that he or she finds most convenient.

The graphical approach is recommended for new developers and those moving from a Windows development environment. Developers with a background in free open source software development will likely prefer the command-line way.

The exact steps are performed on a specific distribution mentioned, but the document should be applicable to any distribution. Besides distribution-specific instructions, generic steps will be also described.

1.1 Graphical way

1. Eclipse IDE with EGit, Zylind Embedded CDT
2. Cross-compiler provided by CodeSourcery Lite binary toolchain distribution
3. OpenOCD on-chip debugger
4. moserial or CuteCom serial terminal
5. Steps performed on Ubuntu 10.10, 32 bit distribution

1.2 Command-line way

1. Text editor and terminal emulator of choice, GNU make, GNU debugger
2. Cross-compiler compiled from official GNU sources
3. OpenOCD on-chip debugger
4. Steps performed on Debian 6.0 (Squeeze), amd64 architecture

Contents

1	Introduction	1
1.1	Graphical way	1
1.2	Command-line way	1
2	Graphical way	5
2.1	Preparing the system and installing custom software	5
2.1.1	Setting up bash as the default shell in the system	5
2.1.2	Installing the CodeSourcery Lite toolchain	5
2.1.3	Inserting the directory of the CodeSourcery toolchain in the PATH	6
2.2	Installing programs provided as packages by the distribution . . .	7
2.2.1	Eclipse	8
2.2.2	Eclipse Git plugin (EGit)	9
2.2.3	OpenOCD	9
2.2.4	CuteCom	10
2.2.5	Moserial	11
2.2.6	Git	12
2.2.7	Performing the actual installation	12
2.3	Installing Eclipse plugins	13
2.3.1	GNU ARM C/C++ development support	14
2.3.2	Zylin Embedded CDT	15
2.4	Setting up authentication for GitHub	16
2.5	Testing the setup	19
2.5.1	Creating a new project and importing the source code . .	19
2.5.2	Compiling the source	29
2.5.3	Uploading to the device	32
2.5.4	Testing debugging facilities	35
2.6	Troubleshooting	38
2.6.1	Error while starting the debugger: “Remote ‘g’ packet is too long: ...”	39
2.6.2	OpenOCD prints error about stm32.flash when attempt- ing to upload program, on Ubuntu 10.04 LTS	39
3	Command-line way	40
3.1	Installing dependencies	40
3.2	Installing on-chip debugger	40
3.3	Installing cross-compiler toolchain	41
3.4	Compiling libopencm3	42
3.5	Testing the setup	42
3.6	Debugging setup	43
3.7	Troubleshooting	44
3.7.1	unable to open ftdi device: device not found	44

List of Figures

1	Setting the default shell to bash, instead of dash	5
2	Opening hidden files for editing	6
3	Editing .bashrc	7
4	Selecting eclipse-cdt for installation	8
5	Extra packages required by eclipse-cdt	9
6	Selecting eclipse-egit for installation	9
7	Selecting openocd for installation	10
8	Selecting cutecom for installation	11
9	Selecting moserial for installation	12
10	Selecting openocd for installation	13
11	Selecting Eclipse workspace	14
12	Search for Eclipse plugins to install	14
13	Selecting GNU ARM C/C++ development for Eclipse	15
14	Selecting Zylind Embedded CDT plugin for Eclipse	16
15	SSH keys at $\sim /.ssh$	17
16	Generating SSH keys with Eclipse	18
17	Adding private keys to Eclipse	19
18	Import from Git	20
19	Select the Git repository	21
20	Parameters of the repository to clone	22
21	Selecting branches to clone	23
22	Selecting the location of local repository	24
23	Selecting the repository from which to import a project	25
24	Selecting the wizard that imports a project	26
25	Selecting the project type	27
26	Selecting the project name	28
27	Checking out the source as project	29
28	Open the C/C++ perspective	30
29	Make targets view	30
30	Adding a make target	31
31	Console window contents, after a successful compilation	31
32	Configuring moserial	32
33	Adding external tools	33
34	Console window contents, after uploading the code to the device	34
35	Hello world messages on serial terminal	35
36	Adding debug favorites	36
37	Starting OpenOCD debug server	37
38	Launching the program for debugging	37
39	Debug perspective	38

List of Abbreviations

CDT	C/C++ Development Tooling
SSH	Secure Shell

2 Graphical way

2.1 Preparing the system and installing custom software

2.1.1 Setting up bash as the default shell in the system

In order to be able to run in Eclipse external programs from special locations (in our case, the compiler and other needed tools), the default shell for script execution must be changed. Otherwise the default system shell (dash) will be launched by Eclipse, and the compiler won't be found.

This can be achieved by launching Applications → Accessories → Terminal, copying the following command in it:

```
sudo dpkg-reconfigure dash
```

and pressing Enter.

You will be asked for your password, and after that a question will be presented, asking if dash should be installed as default system shell. See Figure 1. Select *No*, and press enter.

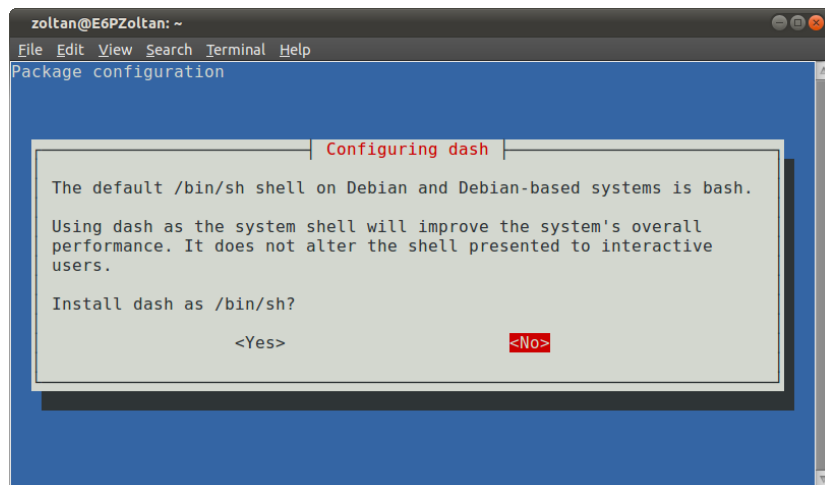


Figure 1: Setting the default shell to bash, instead of dash

Finally close the terminal window. In order to apply these changes, you must either log out and log in, or launch Eclipse from a terminal. Otherwise you will get errors when you are trying to compile the source code. It is recommended to log out and log in after completing section 2.1.3, in order to avoid one extra logout and login.

2.1.2 Installing the CodeSourcery Lite toolchain

The CodeSourcery Lite toolchain contains the compiler, linker, debugger, and other utilities needed for ARM development.

Go to <http://www.codesourcery.com/sgpp/lite/arm/portal/subscription?@template=lite>, select *EABI* as Target OS, and click on the *All versions* link.

As of date of writing, the latest version is 2010.09-51, but due to compatibility problems during debugging, the recommended version is *2010q1-188*. For details, see section 2.6.1 in Troubleshooting part. The direct link to the recommended version is <https://sourcery.mentor.com/GNUToolchain/package6493/public/arm-none-eabi/arm-2010q1-188-arm-none-eabi-i686-pc-linux-gnu.tar.bz2>.

On the webpage, select *IA32 GNU/Linux TAR*, and save the received file. Next, open the file, and extract its contents into your home folder. This way, you should have a folder named *arm-version* in your home directory. In this case, the folder is called *arm-2010q1*.

The next step is to make the system know about the location of the tools; this is achieved by setting up the *PATH* for the current user.

2.1.3 Inserting the directory of the CodeSourcery toolchain in the *PATH*

The *PATH* variable stores a list of directories, where various program executables can be found. In order to make Eclipse know about the ARM development tools, the directory containing the tools must be inserted into this list.

Select from the menu Applications → Accessories → Text Editor, and click open. Navigate to your home folder, then right click on the file list, and select from the menu *Show hidden files*. This operation is depicted on Figure 2.

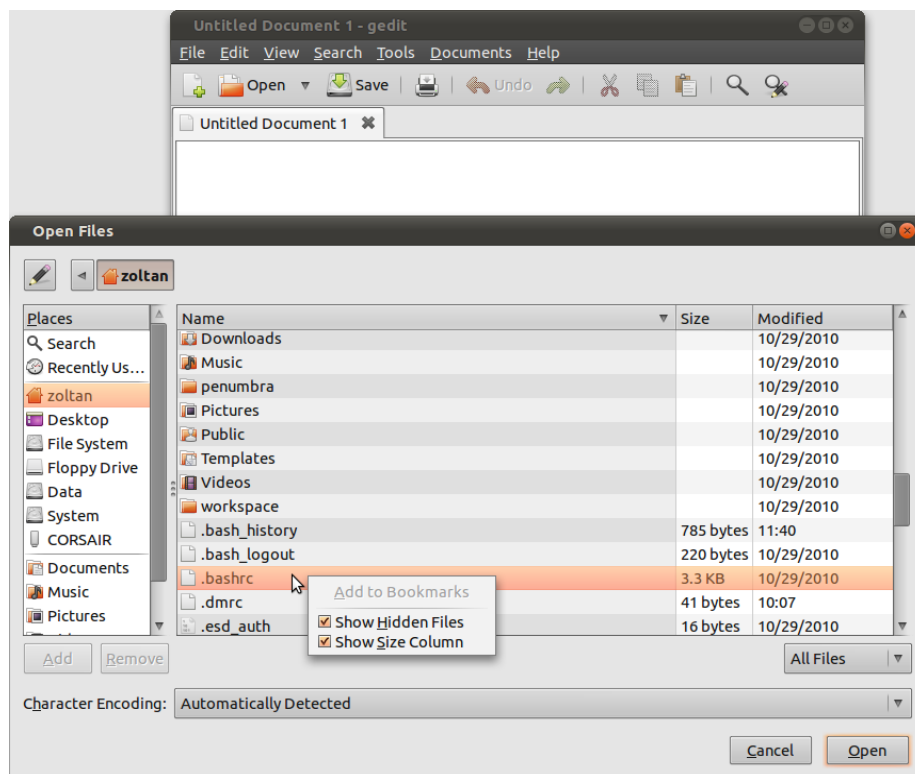


Figure 2: Opening hidden files for editing

From the list, select the file `.bashrc`. Note that hidden files in Linux have their name starting with a dot (`.`).

Select the file called `.bashrc` and insert the following text in it, close to the beginning:

```
export PATH=$PATH:$HOME/arm-2010q1/bin
```

See Figure 3 for example. If you selected different CodeSourcery version, then replace the text `2010q1` accordingly.

Note: the new line in the file needs to be placed before the block *# If not running interactively, don't do anything*, in order to change the `PATH` variable in all situations, including the case when compiling programs from Eclipse.

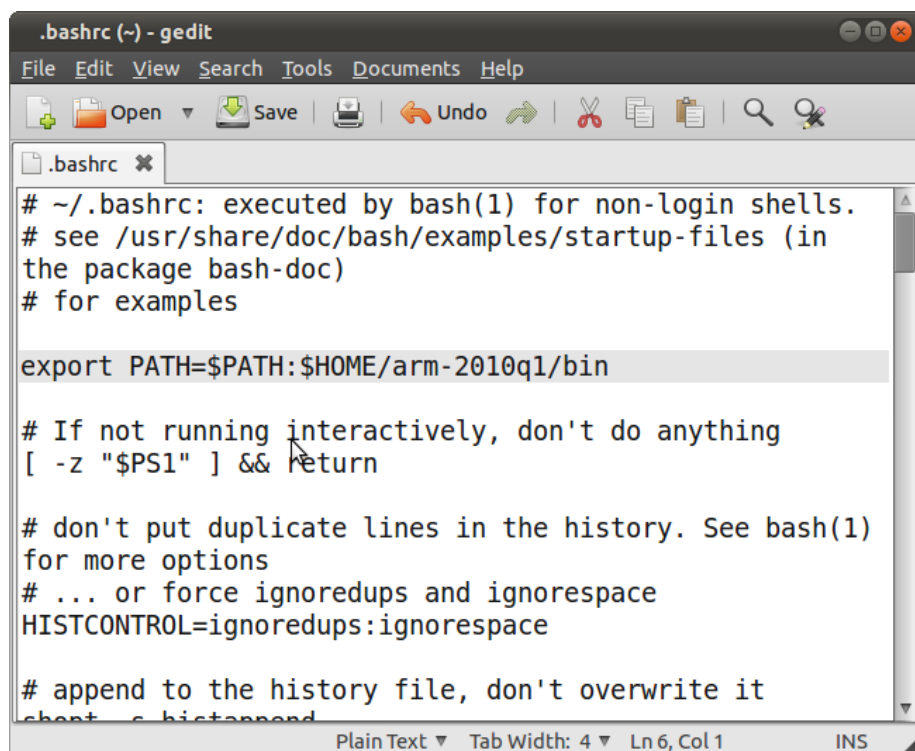


Figure 3: Editing `.bashrc`

Save the file and close the text editor. After completing these operations, and before launching Eclipse, it is needed to log out and log in back, in order to apply the performed changes.

2.2 Installing programs provided as packages by the distribution

Programs provided by Linux distributions can be installed by using the package management system. On Ubuntu, the Synaptic package manager is recommended. Other distributions have equivalent tools for package management. If

Synaptic is not installed, then you can install it first by using Ubuntu Software Center.

Open Synaptic by selecting in the menu System → Administration → Synaptic Package Manager, or Dash home, and write Synaptic in the search field.

2.2.1 Eclipse

In Synaptic select Edit → Search, or Ctrl+F, and enter `eclipse` in the search box. Right click the package called `eclipse-cdt` and select *Mark for installation*. See Figure 4.

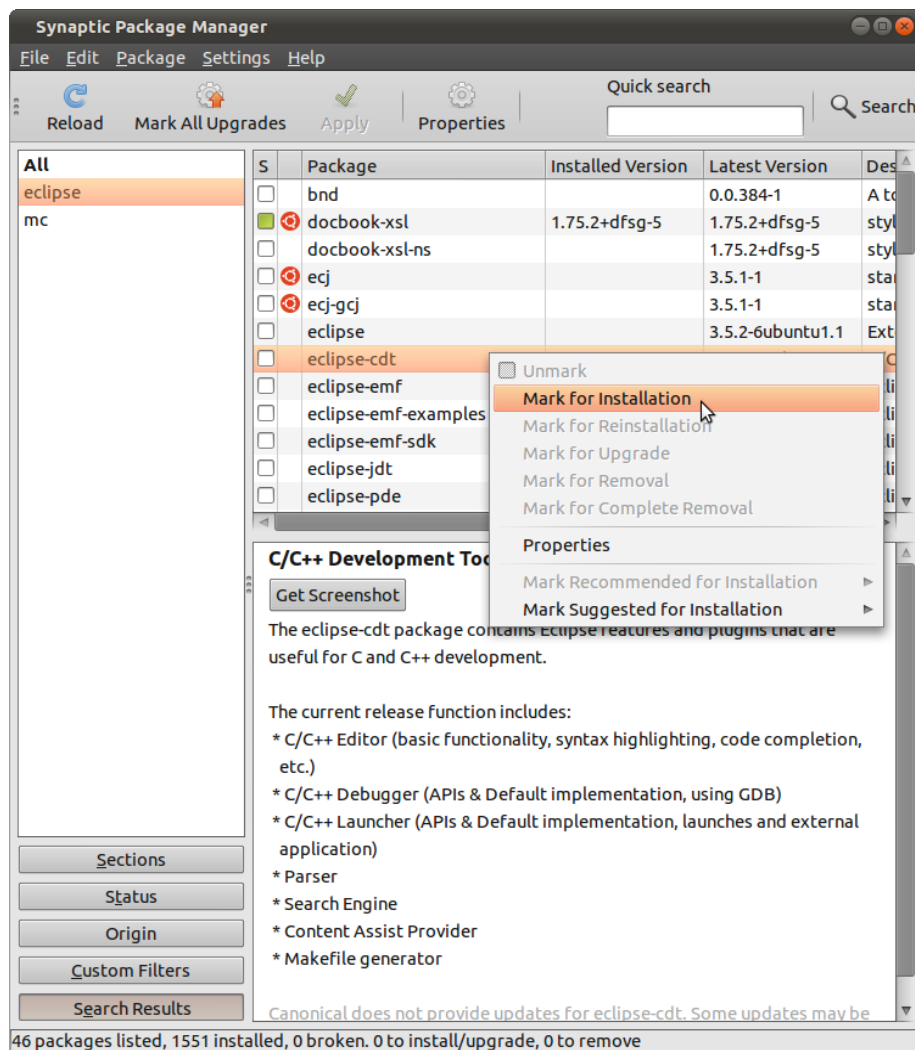


Figure 4: Selecting eclipse-cdt for installation

After this a dialog should pop up, asking if the other, needed packages should be installed, see Figure 5. Click *Mark*.

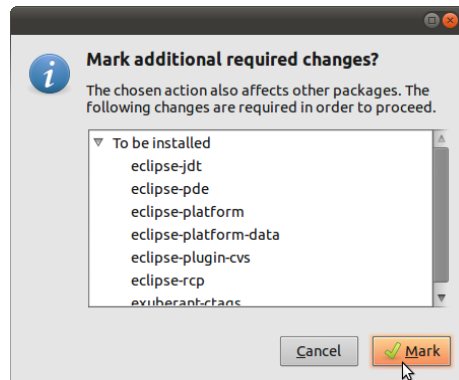


Figure 5: Extra packages required by eclipse-cdt

2.2.2 Eclipse Git plugin (EGit)

From the search results in synaptic, also select the package `eclipse-egit` for installation, see Figure 6. EGit is used in Eclipse for supporting the Git version control system.

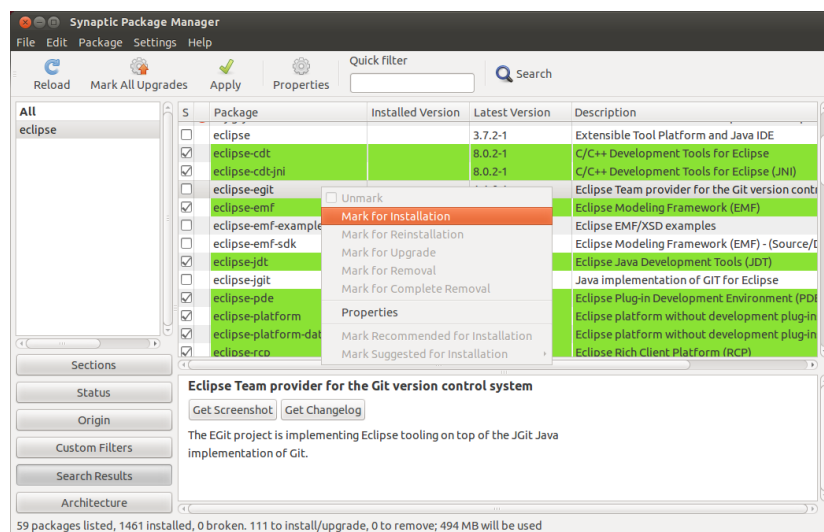


Figure 6: Selecting eclipse-egit for installation

2.2.3 OpenOCD

Still in Synaptic, search for package called `openocd`, and mark it for installation, as shown on Figure 7.

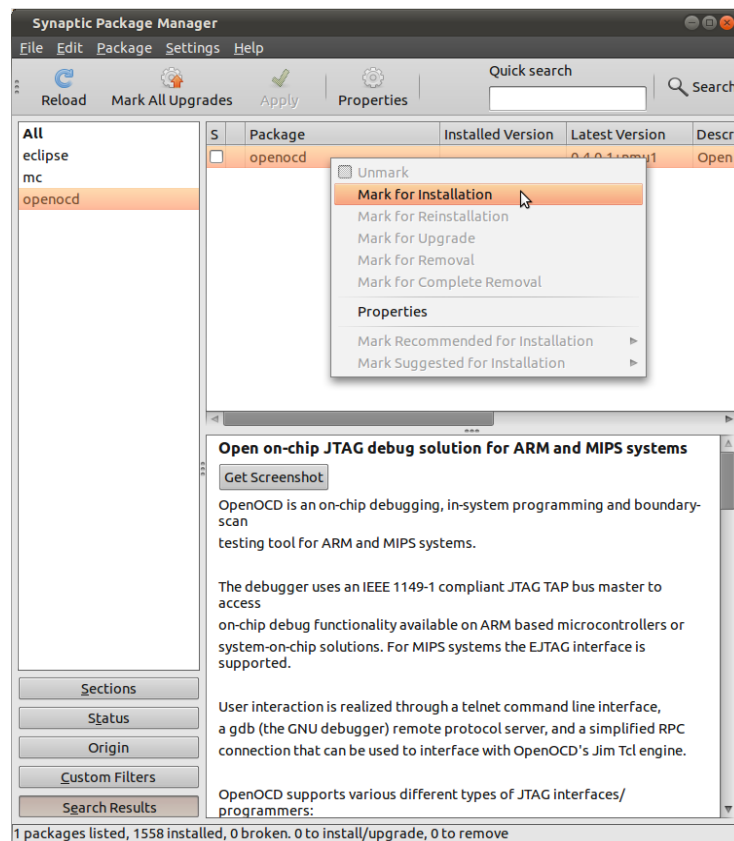


Figure 7: Selecting openocd for installation

2.2.4 CuteCom

Optionally, install a program for communicating on the RS232 port, connected to VESNA. CuteCom is one viable option, Moserial is another one. You can use any of these programs; note that CuteCom handles long RS232 captures much better than Moserial.

Search for `cutecom` on Synaptic, and select it for installation. See Figure 8.

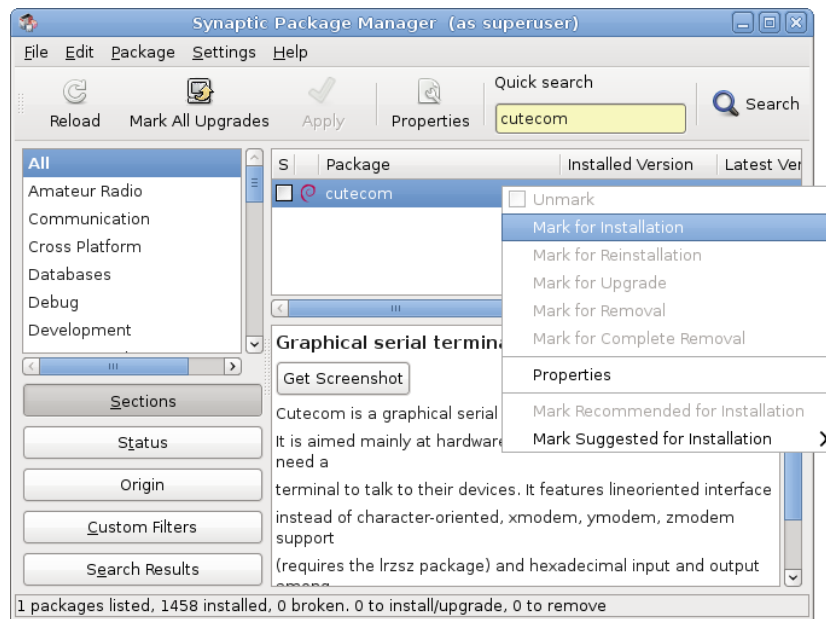


Figure 8: Selecting cutecom for installation

2.2.5 Moserial

Search for `moserial` on Synaptic, and select it for installation. See Figure 9.

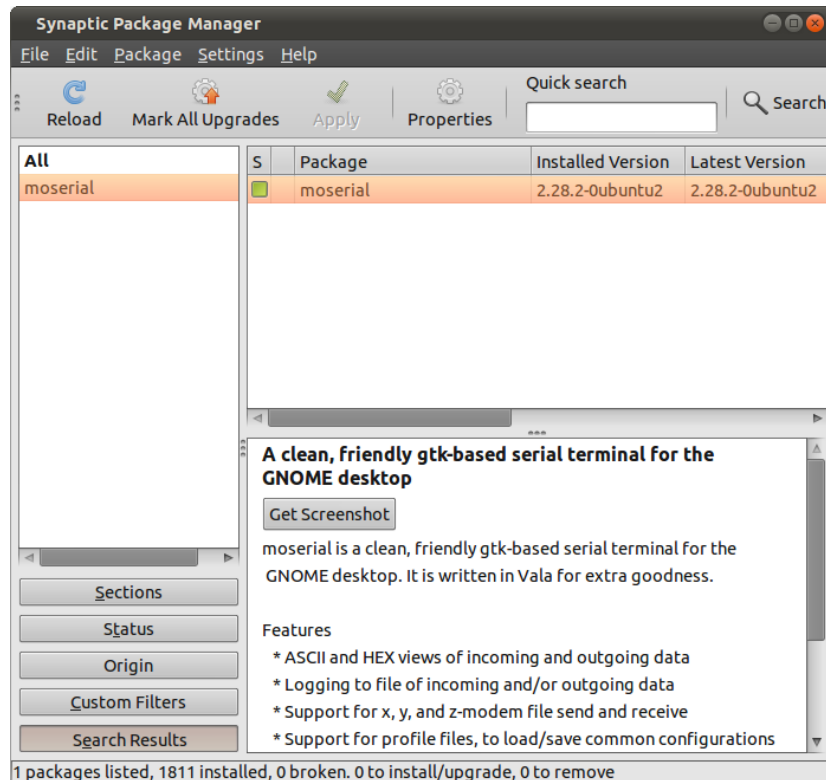


Figure 9: Selecting moserial for installation

The next step is to actually perform the installation of the above mentioned programs.

2.2.6 Git

Optionally you can install the programs to work with Git version control system. Inside Eclipse Git is handled by the EGit plugin, but sometimes it can be useful to manipulate Git repositories from outside Eclipse.

To install git, search in synaptic for `git gui` and install the `git-gui` package for installation. This package will install as dependencies all of the commonly useful Git tools.

2.2.7 Performing the actual installation

Finally, click *Apply* in Synaptic. A window as the one shown on Figure 10 should appear.

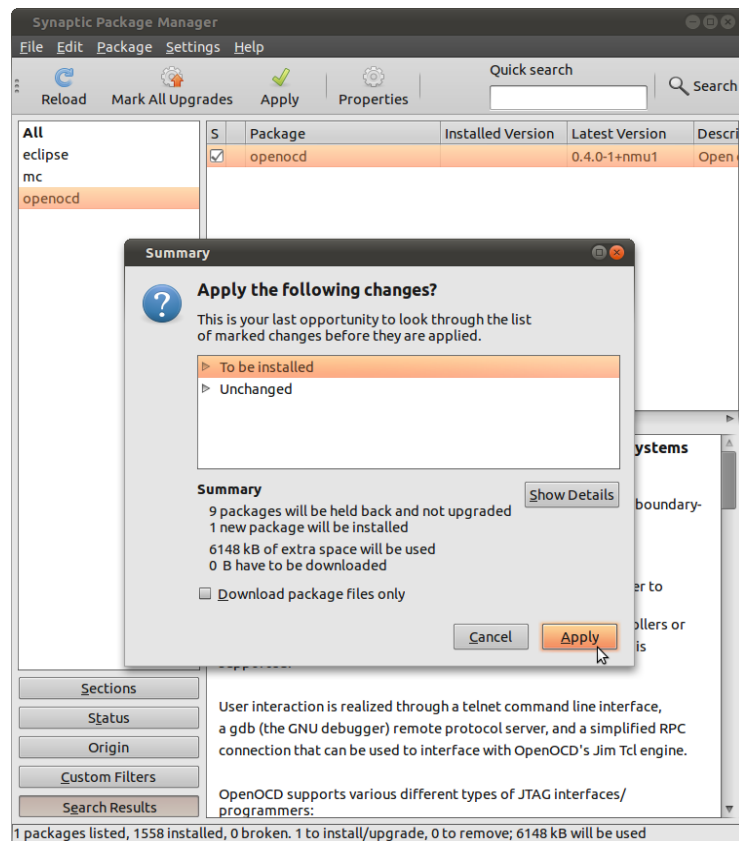


Figure 10: Selecting openocd for installation

Click *Apply*, wait for the installation to finish, and close Synaptic.

2.3 Installing Eclipse plugins

First, start Eclipse, by selecting in the menu Applications → Programming → Eclipse, or Dash home, and search for Eclipse.

After Eclipse starts, it will ask for a workspace, as shown on Figure 11.

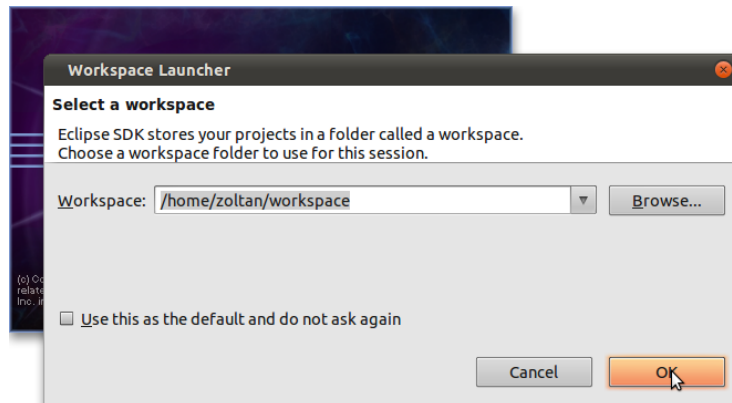


Figure 11: Selecting Eclipse workspace

The default workspace location should be good in most cases, so click *OK*. In Eclipse, select in the menu *Help* → *Install new Software...*, as shown in Figure 12.

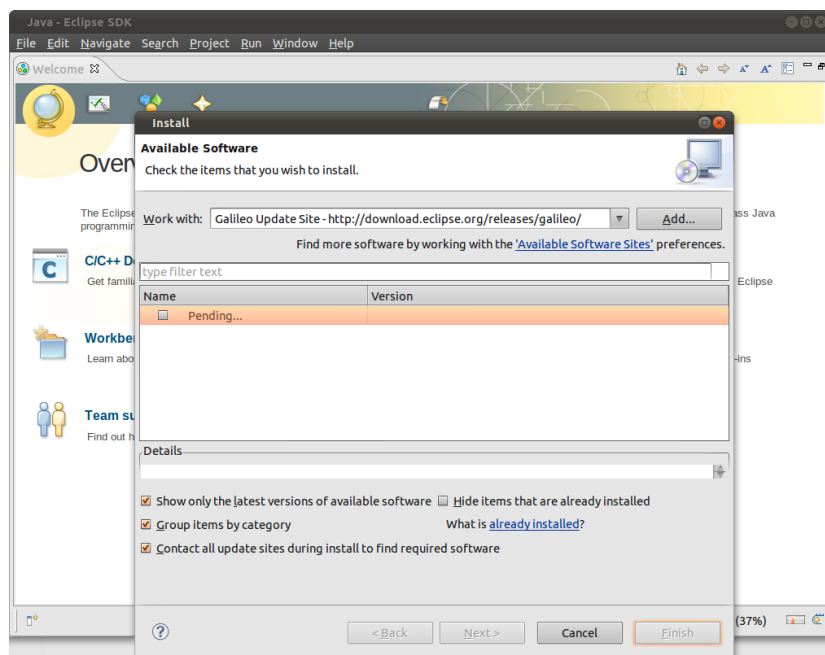


Figure 12: Search for Eclipse plugins to install

2.3.1 GNU ARM C/C++ development support

Again, select in the menu *Help* → *Install new Software...* as shown in Figure 12.

In the install window, paste in the *Work with* field the following URL: `http://gnuarmclipse.sourceforge.net/updates` and press Enter. The list in the middle of the window should have an item, as shown on Figure 13.

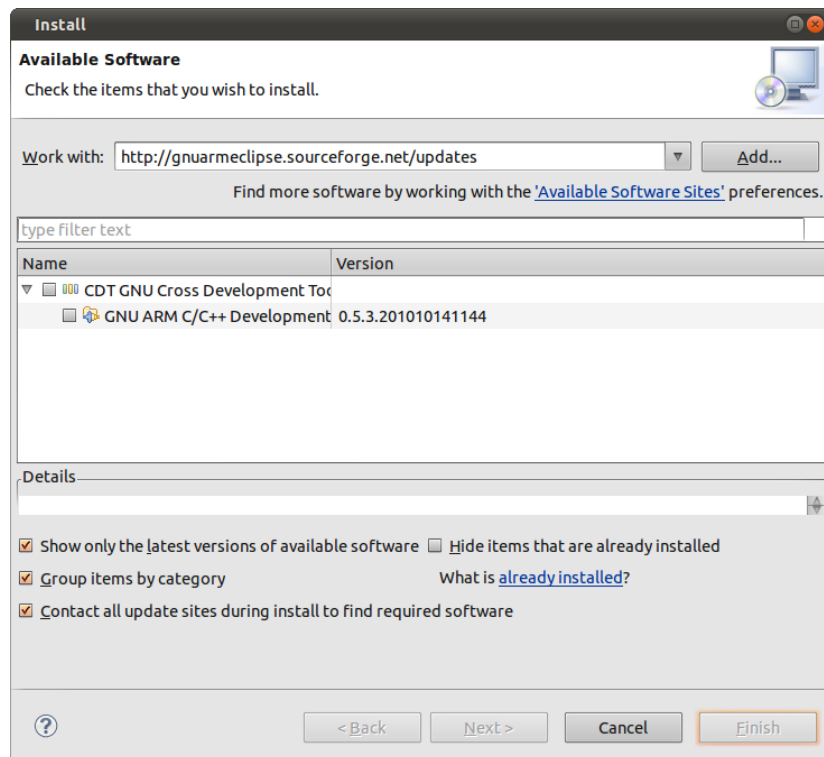


Figure 13: Selecting GNU ARM C/C++ development for Eclipse

Check the *GNU ARM C/C++ Development Plugin* and click *Next*, *Next*, select that you accept the license and click *Finish*.

Finally approve the installation of unsigned software, and restart Eclipse when it's asking.

2.3.2 Zylin Embedded CDT

In Eclipse, select again in the menu Help → Install new Software...

In the *Work with* field, enter the following URL: <http://opensource.zylin.com/zylincdt>.

Select the *Zylin Embedded CDT* plugin for installation, as depicted on Figure 14.

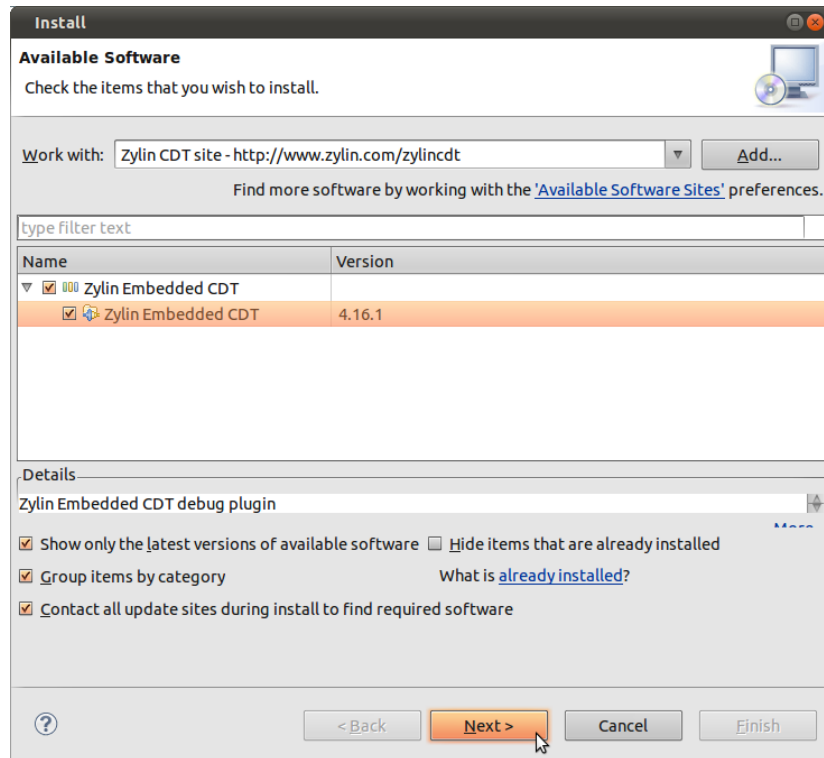


Figure 14: Selecting Zylin Embedded CDT plugin for Eclipse

Proceed with the installation as for GNU ARM C/C++ development support plugin.

2.4 Setting up authentication for GitHub

In order to download the source code from GitHub, one needs to have a GitHub account and at least one SSH key pair set up for GitHub.

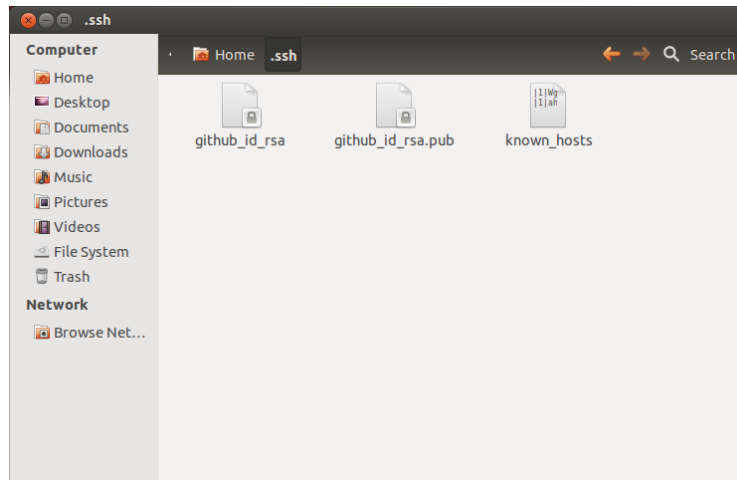


Figure 15: SSH keys at $\sim / .ssh$

In case you have the SSH key pair set up, then you can copy the key to the $\sim / .ssh /$ folder, add the keys to EGit's list of keys and you are done. Note that you need to set up the correct permissions on the key files: the files must be owned by your user, and they can be readable only to your user.

If you need to create an account and generate keys, then you can use the tool available in Eclipse, and proceed with account creation on GitHub. The key generator tool can be found at *Window* \rightarrow *Preferences...* and search for *SSH2*, then select *General* \rightarrow *Network Connection* \rightarrow *SSH2* \rightarrow *Key Management*.

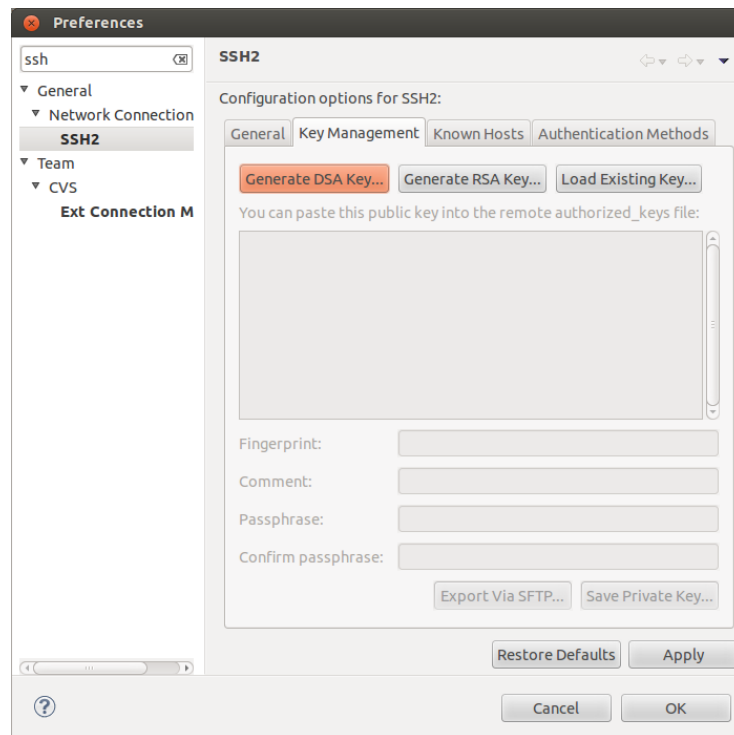


Figure 16: Generating SSH keys with Eclipse

For adding private SSH keys to Eclipse, go to *Window* → *Preferences...* and search for *SSH2*, then select *General* → *Network Connection* → *SSH2* → *General*. There click *Add private key...* and select the private key that you will use with GitHub.

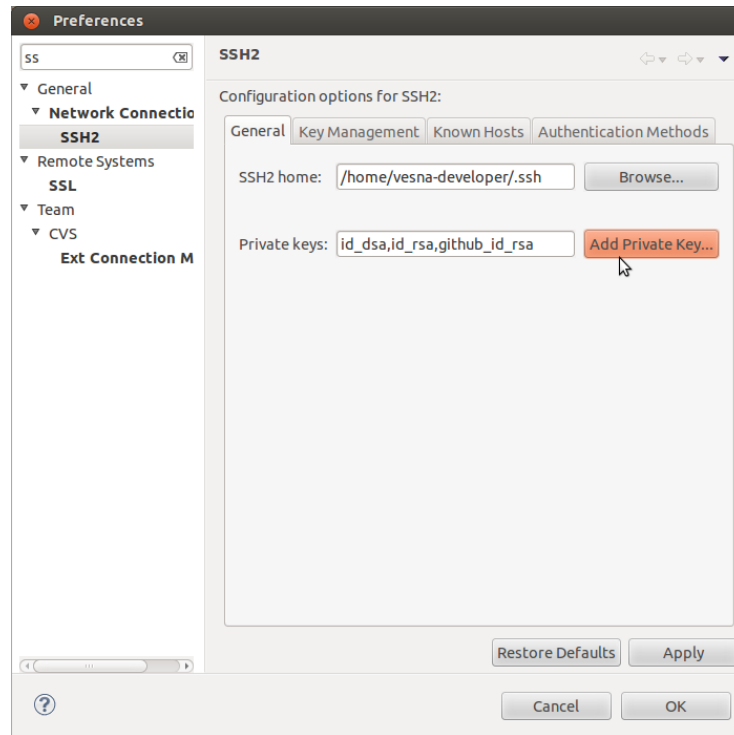


Figure 17: Adding private keys to Eclipse

2.5 Testing the setup

Now everything should be installed for software development. This section guides through the testing of the installed programs and gives a short introductions to their functioning.

Note that after installing the compiler tools, you will need to log out and log in before launching Eclipse. Otherwise you might get errors saying that the executable `arm-none-eabi-gcc` cannot be found.

2.5.1 Creating a new project and importing the source code

In Eclipse, select *File* → *Import...* For the import source select *Projects from Git*, as shown on Figure 18.

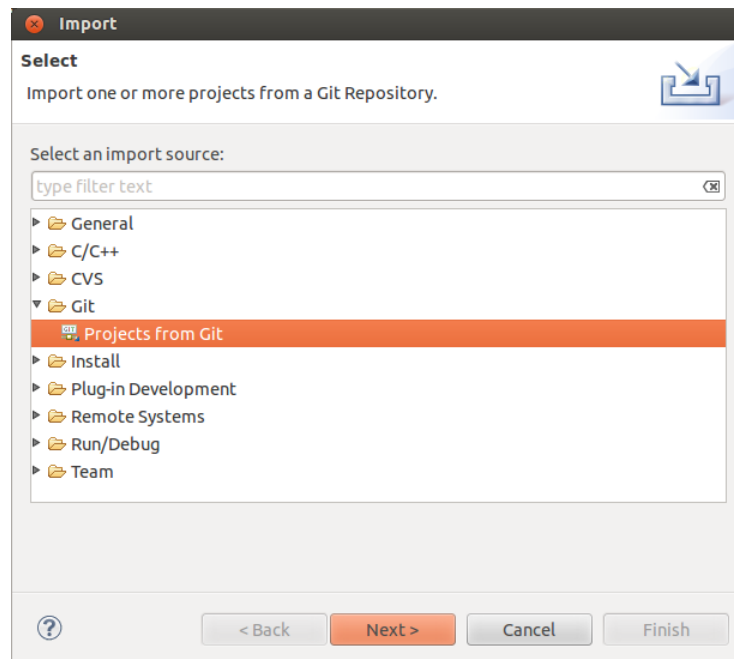


Figure 18: Import from Git

Click *Next*.

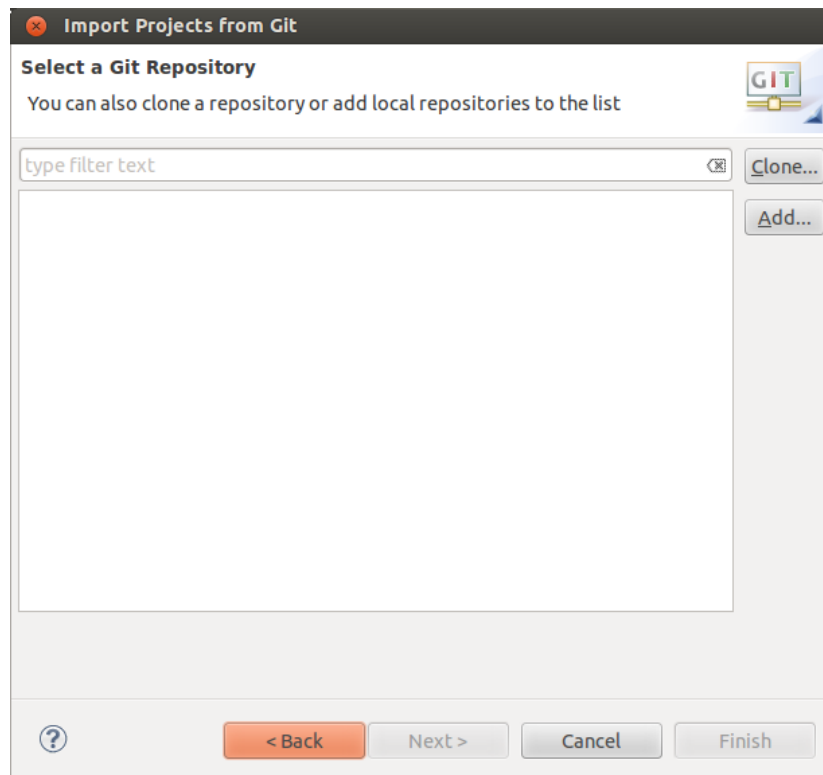
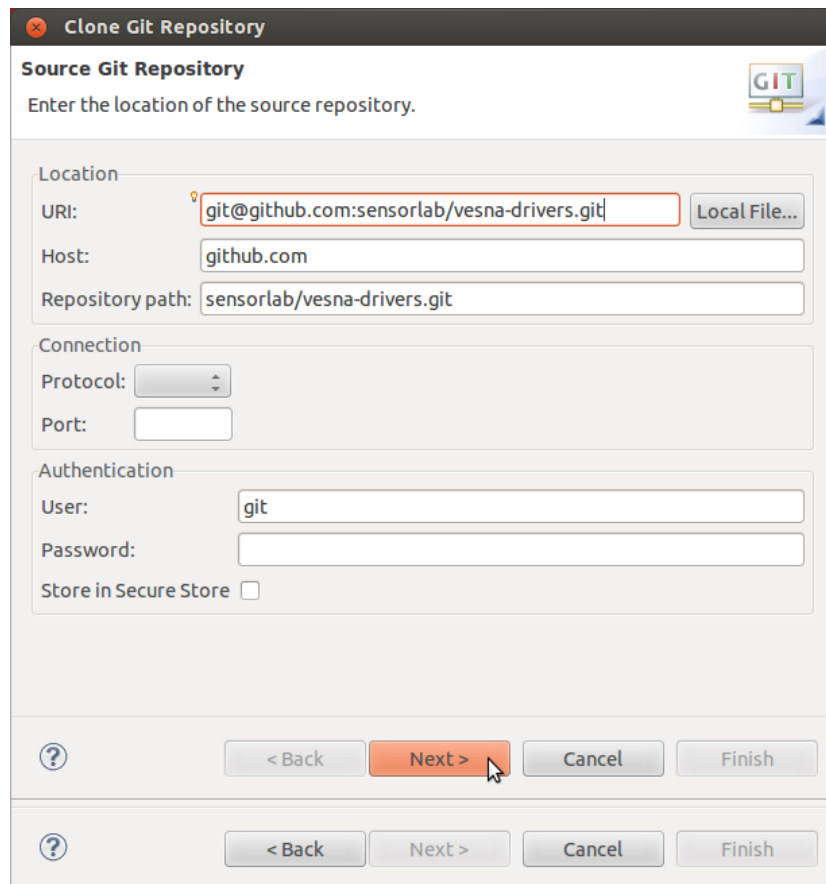


Figure 19: Select the Git repository

Because the source code is not available in a local git repository, first download the source code by cloning the **vesna-drivers** repository.

Click *Clone...* and fill in the URI field with `git@github.com:sensorlab/vesna-drivers.git`



The image shows a 'Clone Git Repository' dialog box with a dark title bar. The main area is titled 'Source Git Repository' and contains the instruction 'Enter the location of the source repository.' The 'Location' section has a 'URI' field with the text 'git@github.com:sensorlab/vesna-drivers.git', a 'Host' field with 'github.com', and a 'Repository path' field with 'sensorlab/vesna-drivers.git'. A 'Local File...' button is next to the URI field. The 'Connection' section has a 'Protocol' dropdown and a 'Port' field. The 'Authentication' section has a 'User' field with 'git', a 'Password' field, and a 'Store in Secure Store' checkbox. At the bottom, there are two rows of buttons: the first row has a help icon, '< Back', 'Next >', 'Cancel', and 'Finish'; the second row has a help icon, '< Back', 'Next >', 'Cancel', and 'Finish'. A mouse cursor is pointing at the 'Next >' button in the first row.

Clone Git Repository

Source Git Repository
Enter the location of the source repository.

Location

URI: Local File...

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

Password:

Store in Secure Store ☐

? < Back Next > Cancel Finish

? < Back Next > Cancel Finish

Figure 20: Parameters of the repository to clone

Click *Next*.

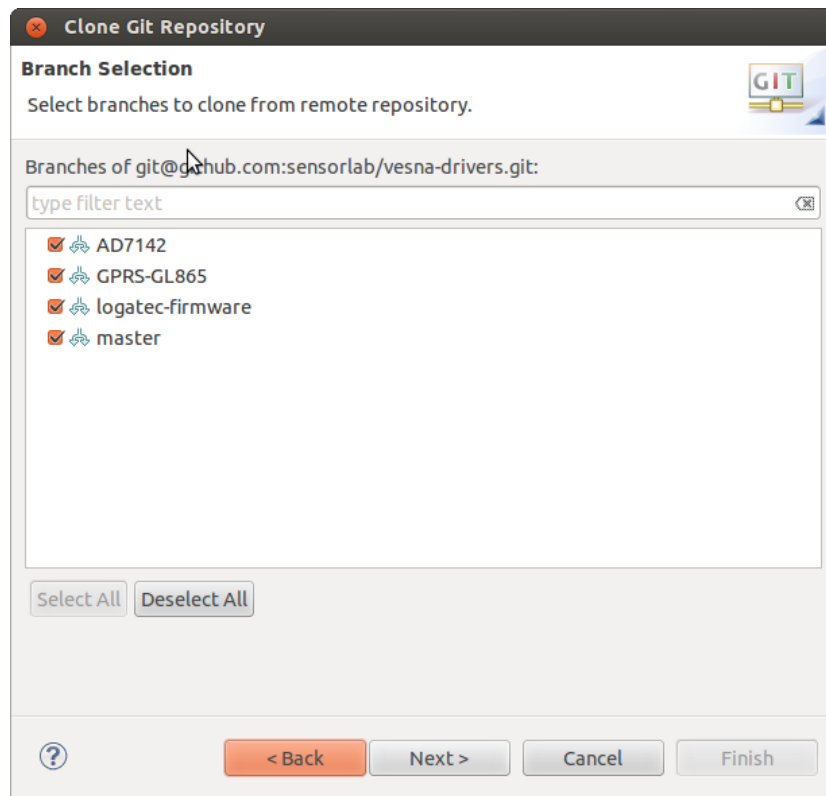


Figure 21: Selecting branches to clone

Click *Next*.

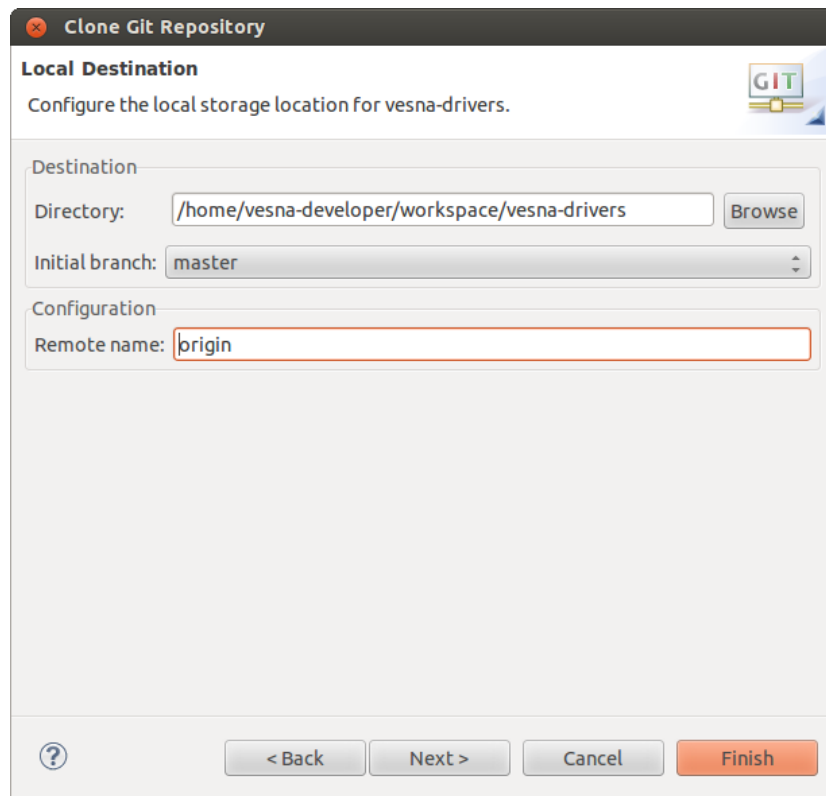


Figure 22: Selecting the location of local repository

For the destination directory select the vesna-drivers directory inside eclipse workspace. This way it's easier to manage the project. Click *Finish*

Now you can select the cloned repository, from which the project can be imported.

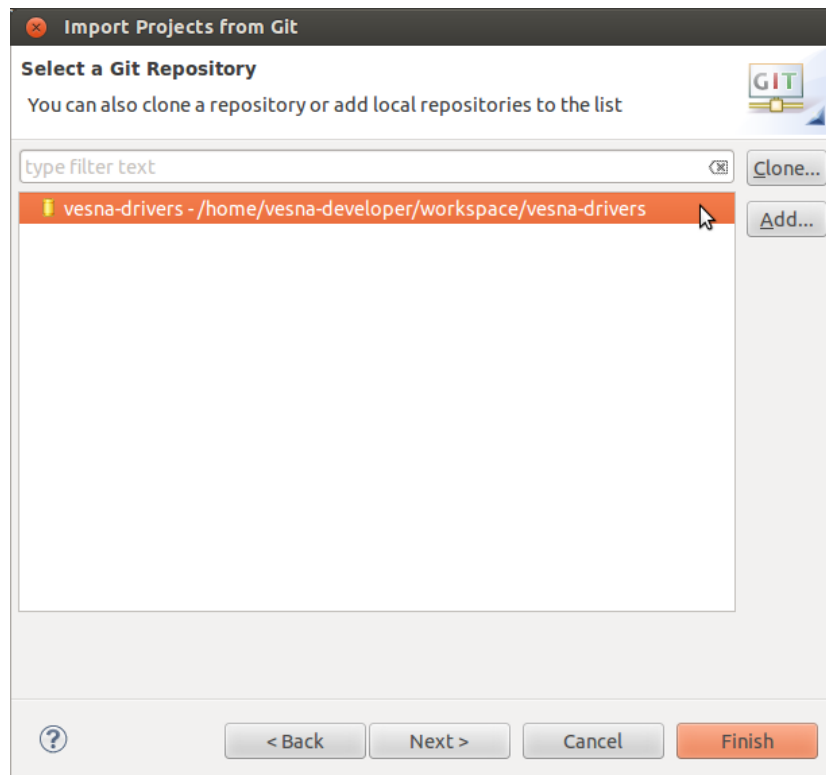


Figure 23: Selecting the repository from which to import a project

Click *Next*. Select the *Use new projects wizard*.

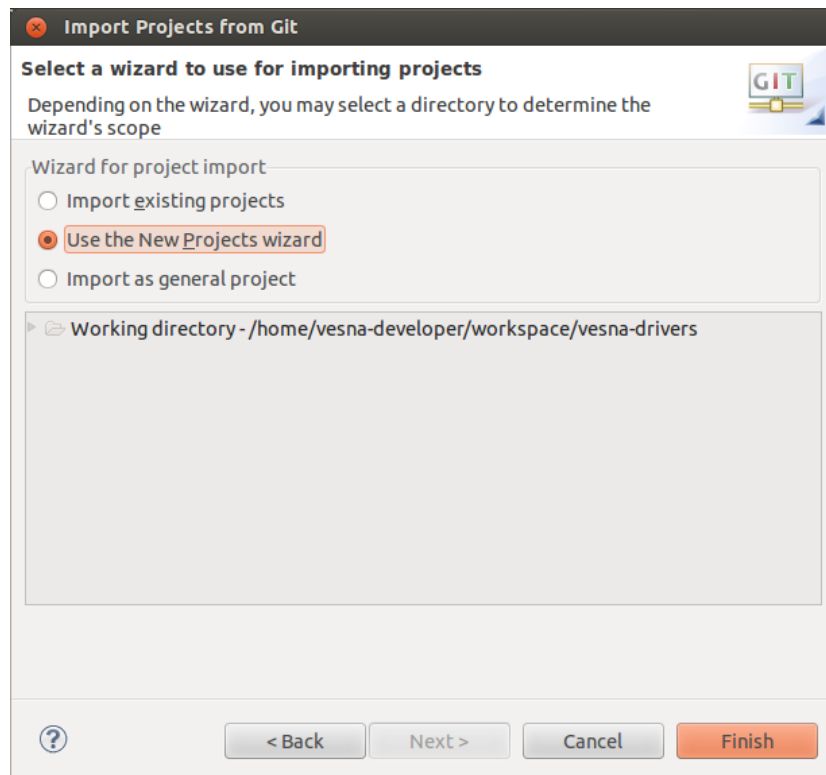


Figure 24: Selecting the wizard that imports a project

Click *Finish*. From the project types, select *C/C++* → *Makefile Project with Existing Code*.

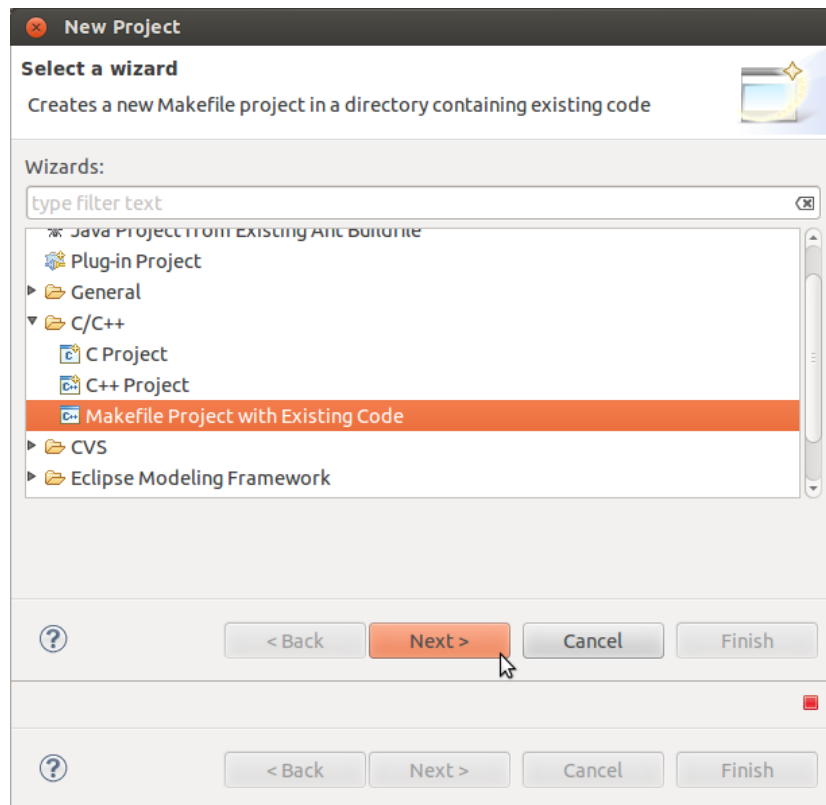


Figure 25: Selecting the project type

Click *Next*. Set the name of the project to `vesna-drivers`.

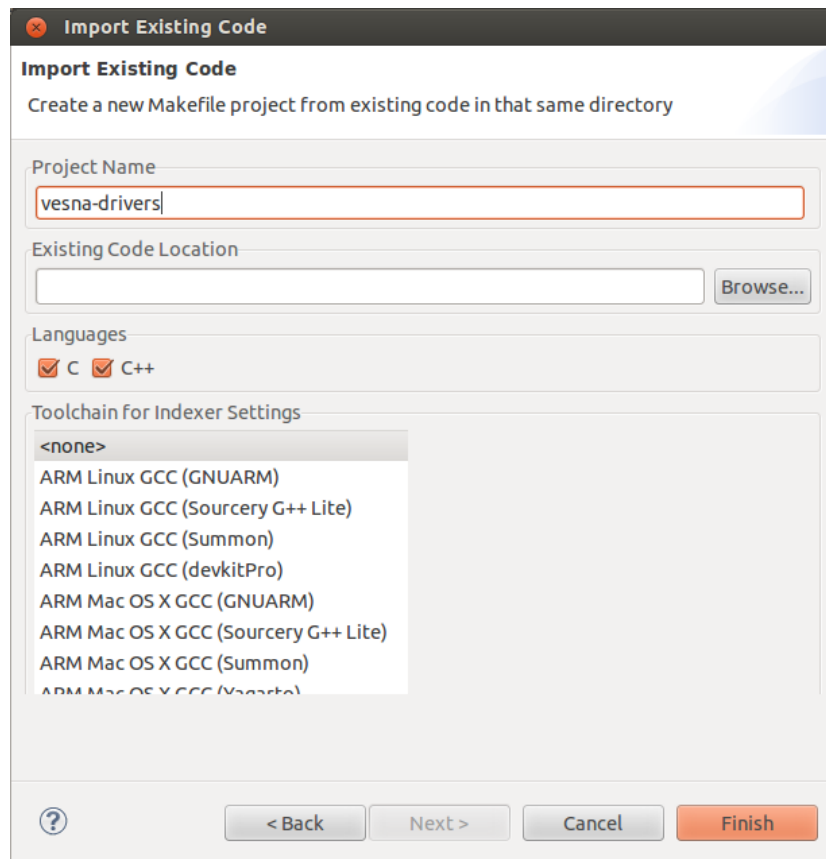


Figure 26: Selecting the project name

Click *Finish*.

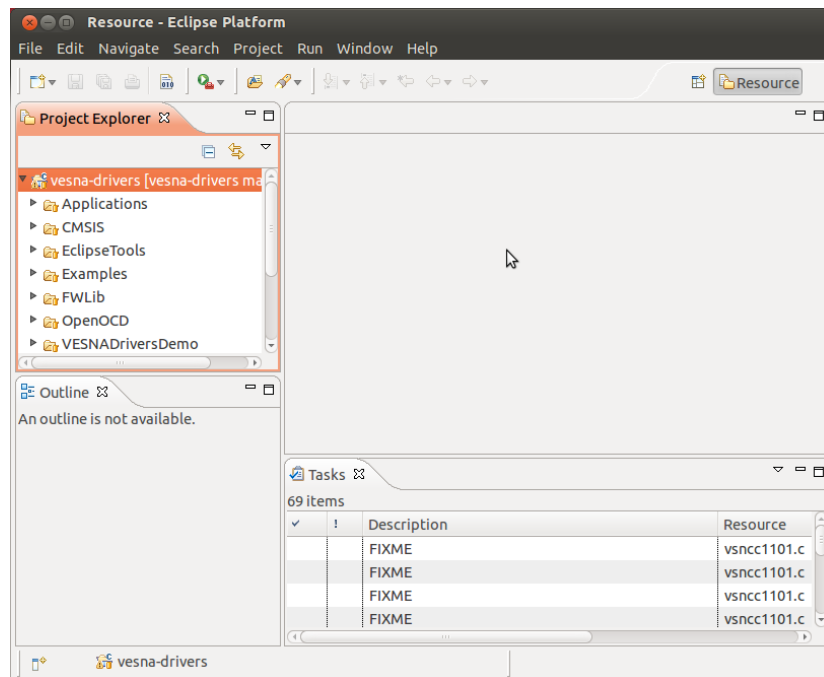


Figure 27: Checking out the source as project

Now you should have the sample project in Eclipse, see Figure 27.

2.5.2 Compiling the source

In Eclipse, open the C/C++ perspective, by selecting *Window* → *Open perspective* → *Other...*, and clicking *C/C++* and *OK*. See Figure 28.

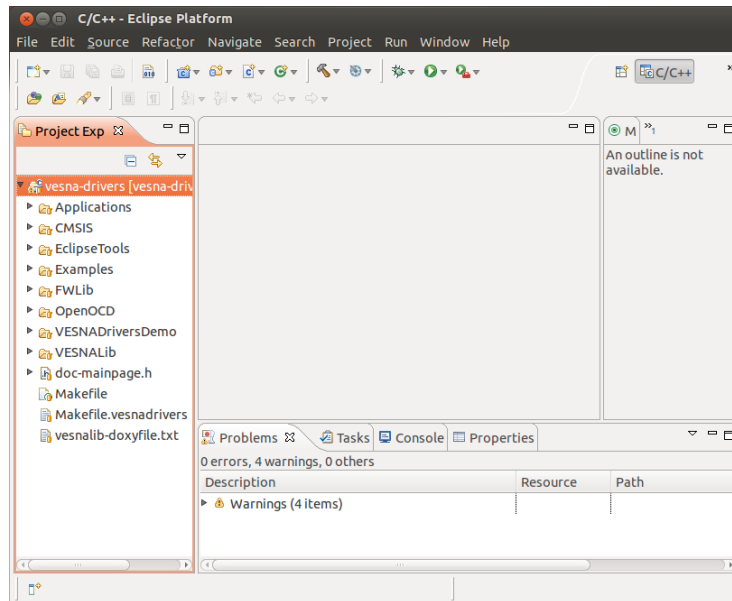


Figure 28: Open the C/C++ perspective

In this perspective, on the right side, there should be a *Make targets* view.

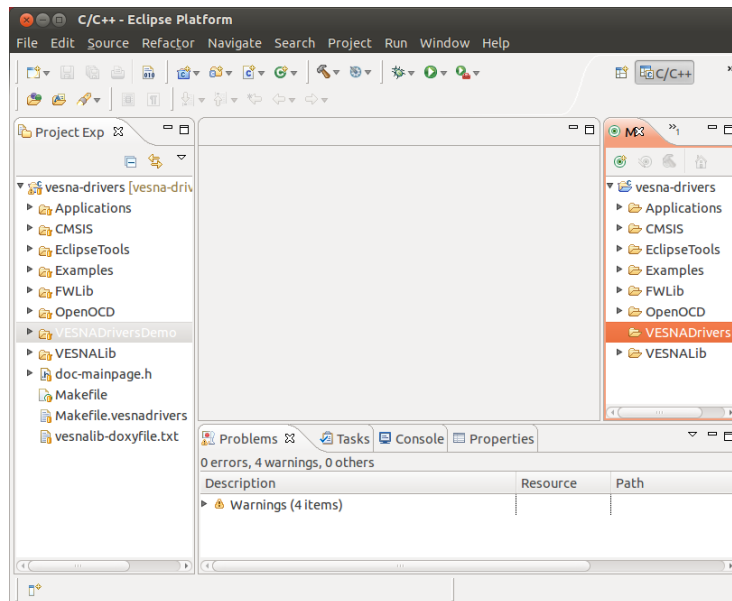


Figure 29: Make targets view

Select the *VESNADriversDemo* folder in the Make targets view, and right-click on it and select *New* in the menu. Set the make target name to *all*.

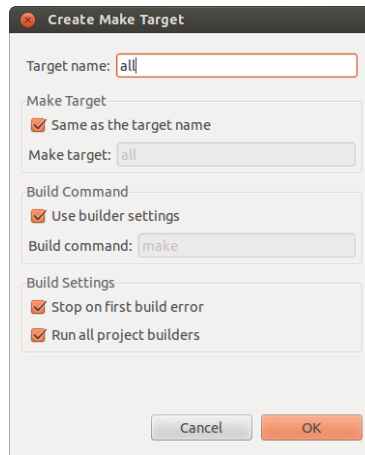


Figure 30: Adding a make target

Other useful targets are `clean` for removing the build output files from the source code and `drivers_demo.load5` for loading the application to VESNA.

Click *OK*, then double-click the newly created make target. If everything goes well, you should see similar lines in the console, as the text below:

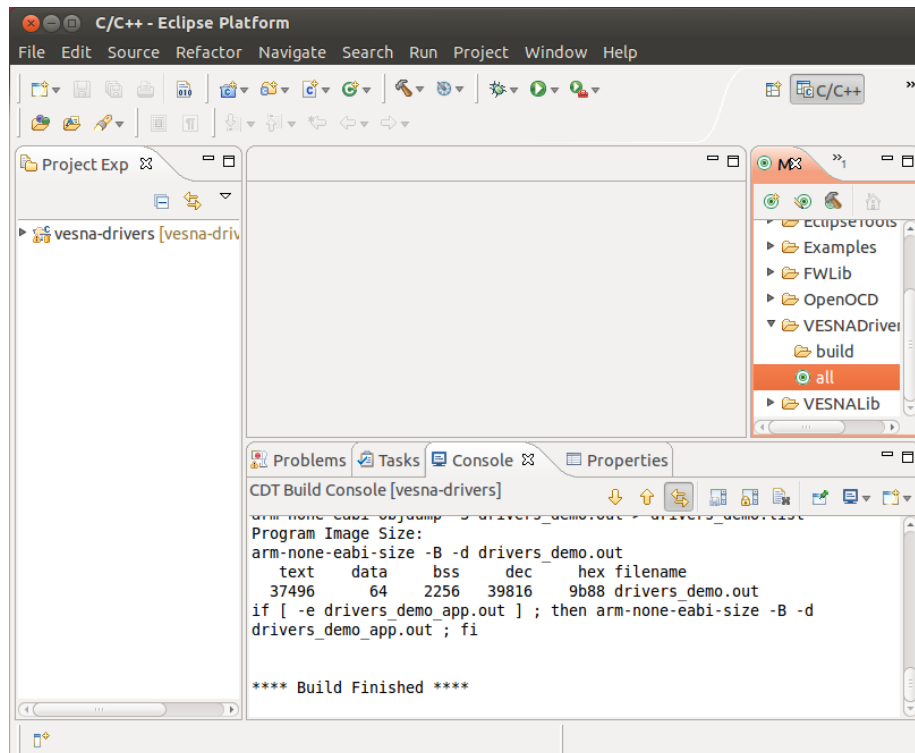


Figure 31: Console window contents, after a successful compilation

Now the example project has compiled, and it's ready to be loaded on

VESNA.

2.5.3 Uploading to the device

First, connect all the necessary cables to the debugger:

- RS232 cable
- JTAG cable
- power supply

and plug the debugger into a USB slot on the PC.

Then, it's recommended to launch the RS232 monitoring program. In order to do this, select from the menu Applications → Accessories → moserial Terminal. Inside the program, click on the serial port configuration icon. See Figure 32.

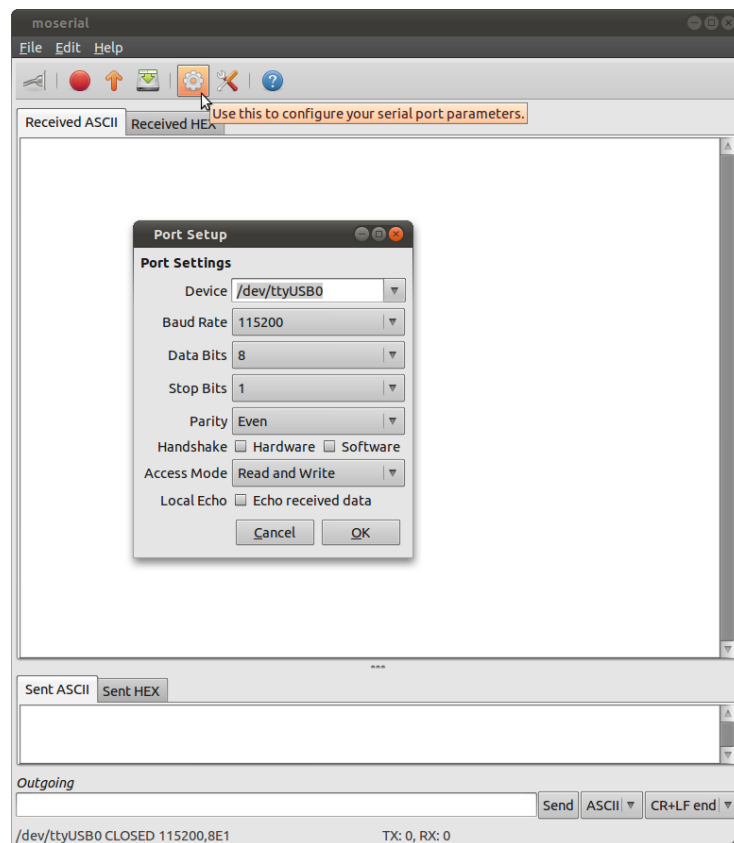


Figure 32: Configuring moserial

The usual settings are:

Port	/dev/ttyUSB0
Speed	115200
Parity	even
Bits	8
Stop bits	1
Flow control	none

Note that the port might be different if you are using serial ports on the PC (in that case, the port could be /dev/ttyS0, /dev/ttyS1, ...) or if you have more serial devices attached on USB (/dev/ttyUSB1, /dev/ttyUSB2, ...).

After applying appropriate settings, click the first button on the toolbar for activating the serial port connection.

Now, in Eclipse, select in the menu Run → External Tools → Organize Favorites, select *Add*, check both items and click *OK* and *OK*. See Figure 33.

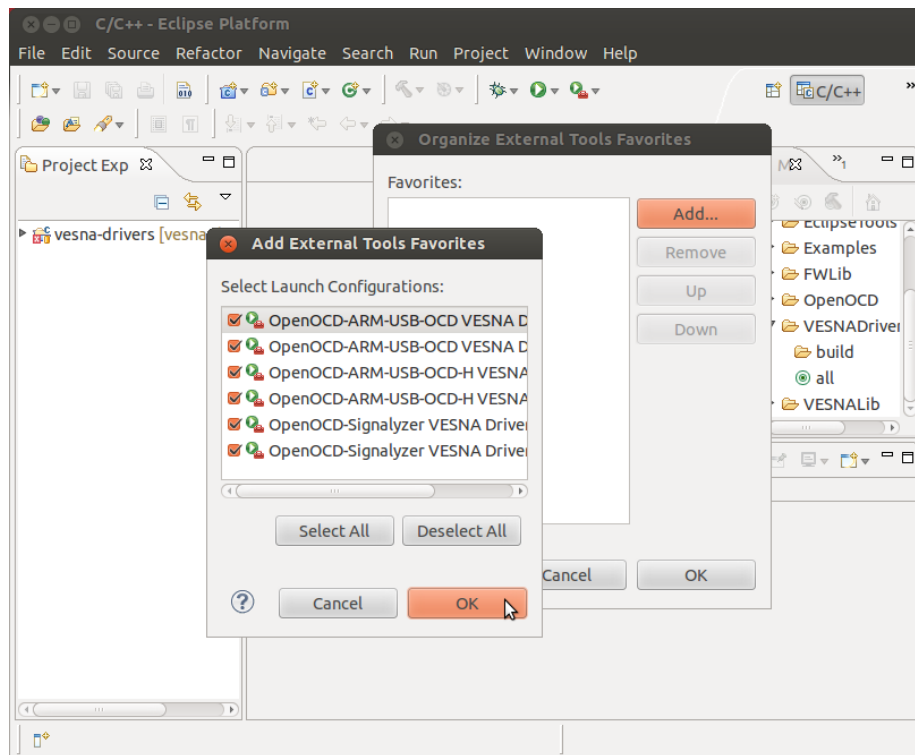


Figure 33: Adding external tools

In order to load the program select Run → External Tools → OpenOCD-ARM-USB-OCD Load.

In the console in Eclipse, you should see the messages depicted on Figure 34, after about 5 seconds

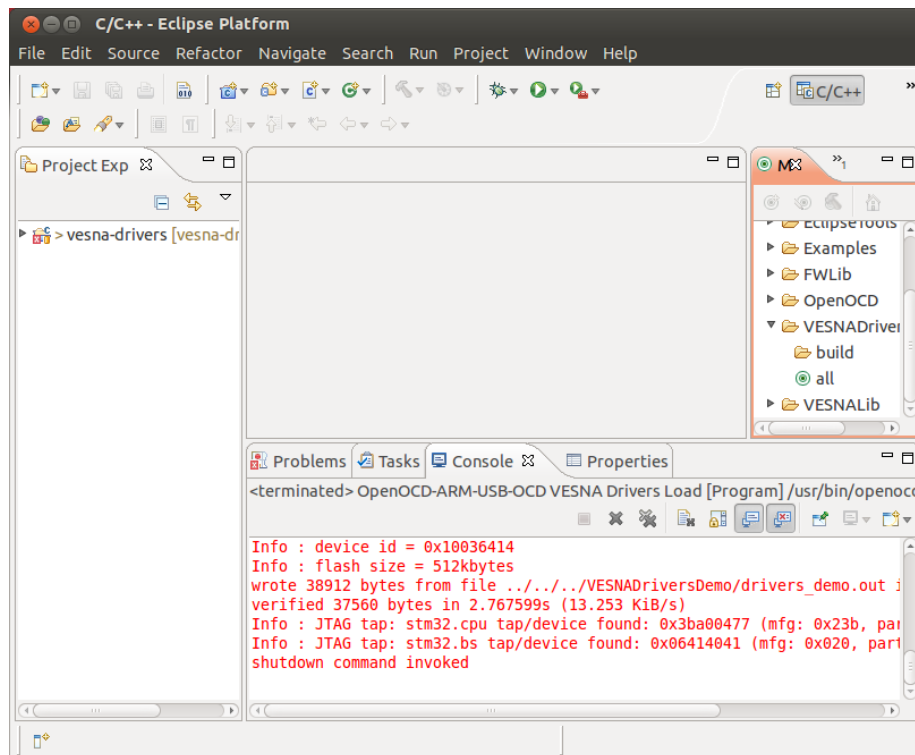


Figure 34: Console window contents, after uploading the code to the device

After these messages appear, VESNA should start printing hello world messages on the serial terminal, as shown on Figure 35.

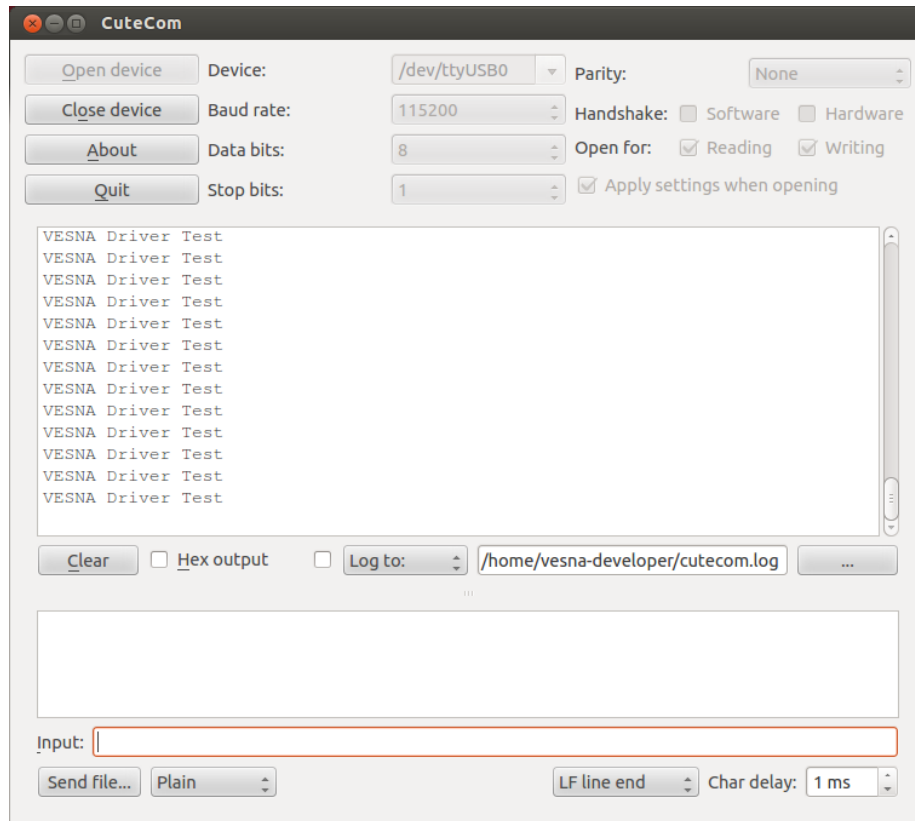


Figure 35: Hello world messages on serial terminal

By following these steps, you should be able to compile and load program to VESNA.

2.5.4 Testing debugging facilities

In order to debug in Eclipse, first the debug configurations have to be added. In Eclipse, select on the toolbar *Debug* → *Organize favorites....*

In the window that appears, click *Add...*, select both items, and click *OK* and *OK*. Illustration can be found on 36.

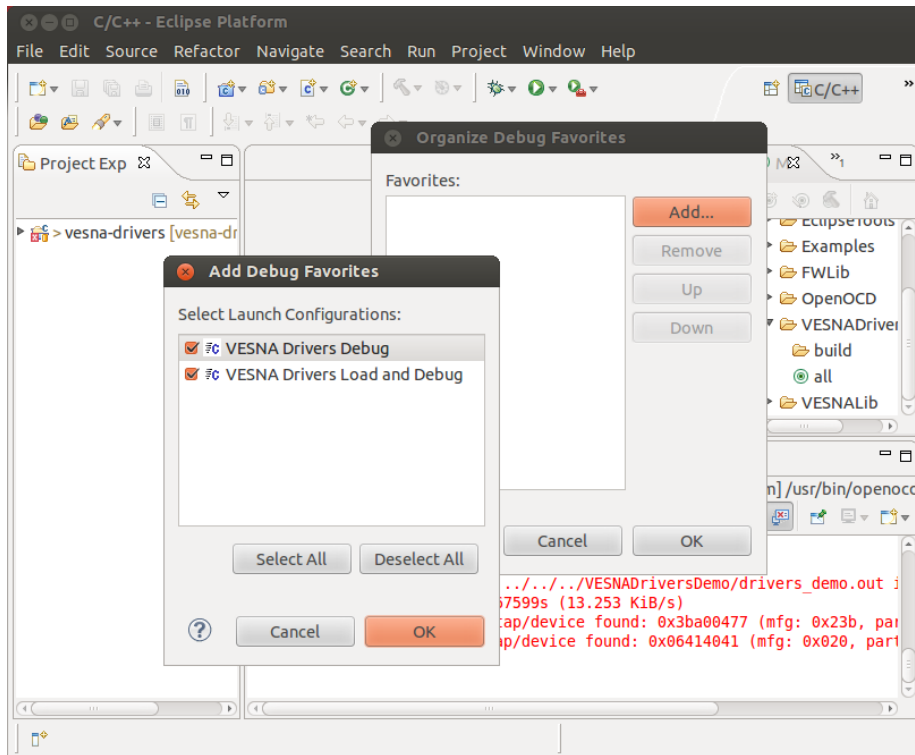


Figure 36: Adding debug favorites

In order to be able to debug programs running on VESNA, a debug server must be started. In this case, the debug server is called OpenOCD. Note that while the debug server is running, VESNA cannot be reprogrammed. Starting the debug server can be performed by clicking on the *Debug* icon on the toolbar, and selecting *OpenOCD-ARM-USB-OCD Debug*. See Figure 37. Launching the debug server is necessary every time a new debug session is started, and it must be stopped before reprogramming the VESNA.

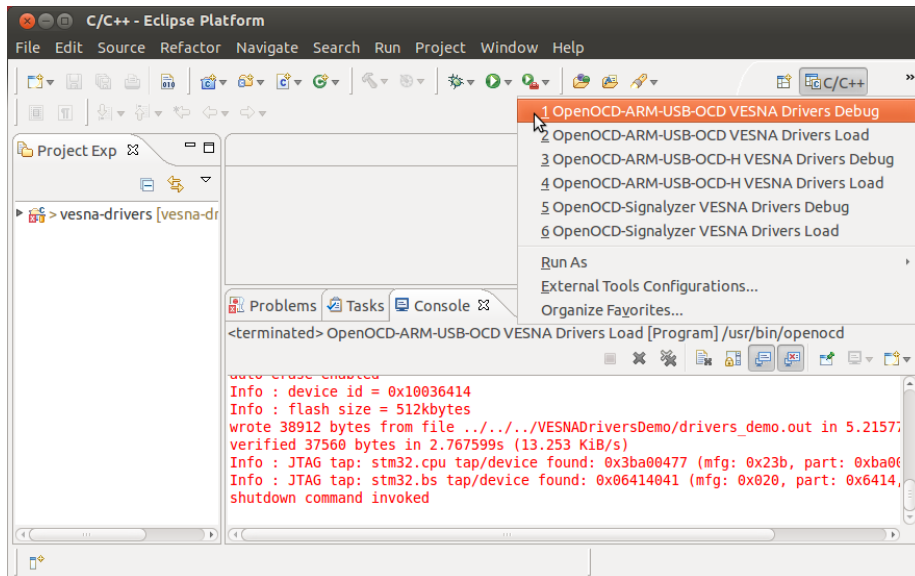


Figure 37: Starting OpenOCD debug server

Next, start the actual program that will be debugged. For this, click on the *Debug* icon on the toolbar, and select *OpenOCD-ARM-USB-OCD VESNA Drivers Debug* (Figure 38)

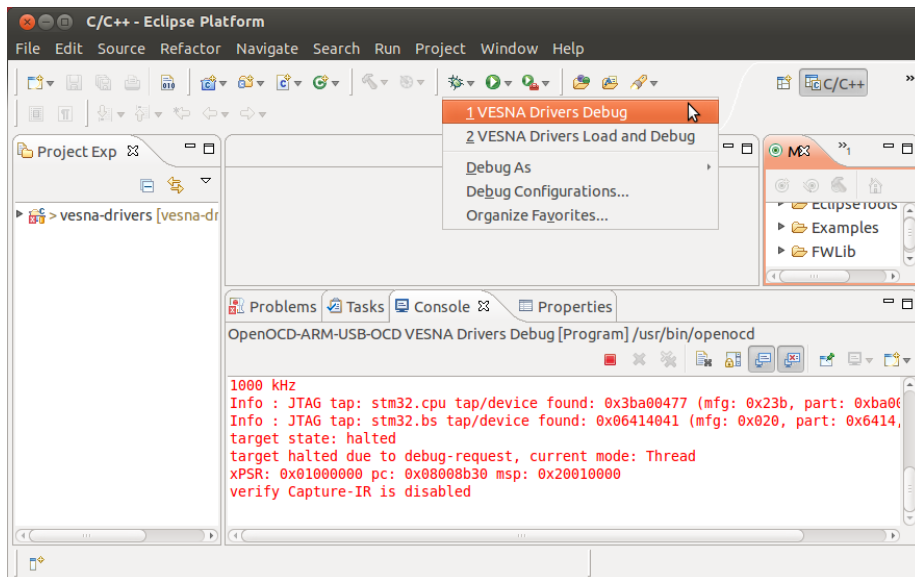


Figure 38: Launching the program for debugging

Now it's recommended to switch to the debug perspective, for efficient debugging. Click the *Debug* icon on the upper right corner of the Eclipse window. The debug perspective is depicted on Figure 39.

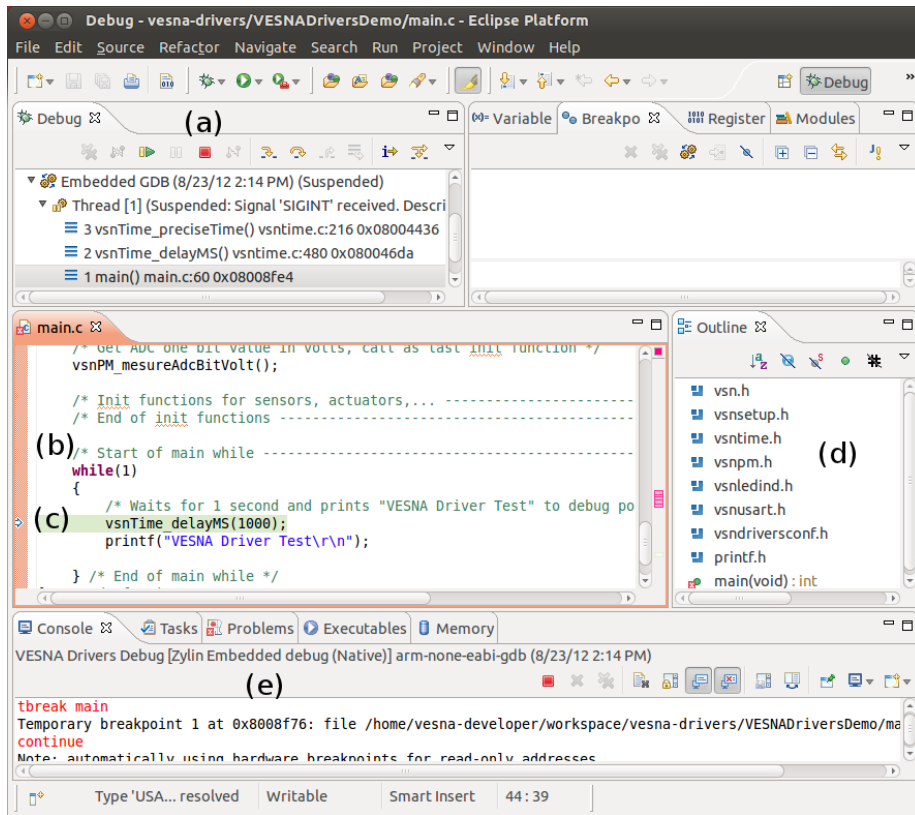


Figure 39: Debug perspective

The marked elements on Figure 39 are:

- Program flow control and the running processes. Note that the list also contains OpenOCD, the debug server, and the program being debugged, including the stack(s).
- Source code view
- The line being executed.
- The outline of the current source code. In the upper region there are the lists of breakpoints, register values, variables.
- Debugging console.

At the end of debugging, the executed program and also the debug server should be stopped.

2.6 Troubleshooting

This section tries to give fixes to common problems. The list might be incomplete, so feel free to contact the author for including new entries in this section.

2.6.1 Error while starting the debugger: “Remote ‘g’ packet is too long: ...”

This problem is caused by incompatible versions of OpenOCD and CodeSourcery debugger. See a discussion about this topic at OpenOCD mailing list: <https://lists.berlios.de/pipermail/openocd-development/2010-December/017349.html>

Below is a table of OpenOCD and CodeSourcery toolchain versions that are known to work:

CodeSourcery version	OpenOCD version
2010q1-188	0.4.0-1+nmul (Ubuntu 10.10 package)

Table of not working combinations:

CodeSourcery version	OpenOCD version
2010.09-51	0.4.0-1+nmul (Ubuntu 10.10 package)

Combinations not appearing in any of the tables have not been tested.

2.6.2 OpenOCD prints error about stm32.flash when attempting to upload program, on Ubuntu 10.04 LTS

This problem is caused by the old version of OpenOCD (v3.1) supplied with Ubuntu 10.04. A simple workaround is to temporarily add the Ubuntu 10.10 package repositories to the system update sources and install version 4 of OpenOCD from there. After installing OpenOCD, remove the extra repositories, because installing software from different version of the same distribution might cause problems.

3 Command-line way

The following instructions assume you are working in a terminal emulator, either a text-mode virtual console or a graphical emulator. If you are using the default GNOME environment, you can open one from the Applications, Accessories, Terminal menu entry.

Command lines starting with dollar sign (\$) are to be executed under your normal user privileges. Lines starting with a hash sign (#) should be executed as root using `su` or `sudo`, depending on your system configuration.

A backslash (\) at the end of the line means the command continues on the next line without a carriage return.

3.1 Installing dependencies

All of the software that we will compile from source will be installed into your home directory. This approach was chosen so that the time required to work with elevated privileges is minimal and hence minimize the chances of the installation affecting other users or software on your system. However, before we can do that, we need to install some prerequisite software through the system's package manager.

Perform the following command as root and confirm installation of packages when prompted:

```
# aptitude install libgmp3-dev \  
libncurses5-dev \  
libmpc-dev \  
autoconf \  
build-essential \  
git-core  
# aptitude build-dep gcc-4.4
```

3.2 Installing on-chip debugger

At the time of writing, the `openocd` version in the Debian stable distribution (0.3.1) is too old to support VESNA. We require at least version 0.5.0, so we will compile and install the package from source.

If your distribution already ships with `openocd` version 0.5.0 or later (for instance Ubuntu 12.10), you can skip this step and simply install `openocd` through the package manager (e.g. `aptitude install openocd`—).

Begin by downloading and unpacking the Debian source package (if the URLs below no longer work, you can find a suitable version at <http://packages.debian.org/wheezy/openocd>).

```
$ wget http://ftp.de.debian.org/debian/pool/main/o/openocd/openocd_0.5.0-1.dsc \  
http://ftp.de.debian.org/debian/pool/main/o/openocd/openocd_0.5.0.orig.tar.bz2 \  
http://ftp.de.debian.org/debian/pool/main/o/openocd/openocd_0.5.0-1.debian.tar.gz  
$ dpkg-source -x openocd_0.5.0-1.dsc
```

Now build the package. `dpkg-buildpackage` might display a message about missing build dependencies. In that case, install the missing packages that are listed in the error message using `aptitude install` and re-run `dpkg-buildpackage`.

```
$ cd openocd-0.5.0
$ dpkg-buildpackage -uc -us
$ cd ..
```

Install the compiled package as root:

```
# dpkg -i openocd_0.5.0-1_amd64.deb
```

Check the setup by running `openocd -v`. It should return the following:

```
Open On-Chip Debugger 0.5.0 (2011-08-09-08:45)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.berlios.de/doc/doxygen/bugs.html
```

Note that your user account usually needs to be a member of the group `plugdev` in order to have access to the OCD programmer and other USB devices. You can ensure this by running the following command as root (replace `foo` with your user name):

```
# adduser foo plugdev
```

3.3 Installing cross-compiler toolchain

We will use a customized version of the `summon-arm-toolchain` script to build a GNU toolchain targeting ARM Cortex M3 architecture.

First we need to download the latest version of the script:

```
$ git clone https://github.com/avian2/summon-arm-toolchain.git
```

Now run the script. This will download, compile and install the toolchain (binutils, gcc, gdb and newlib C library) into `~/local`. If you would like to use a different directory, pass a different path to the `-t` command-line argument.

```
$ cd summon-arm-toolchain
$ ./summon-arm-toolchain -t ~/local
```

You should now add `~/local/bin` (or the directory you installed the toolchain in) to your `PATH`. This is best done by editing `~/.bashrc` and adding a line like the following at the top:

```
export PATH="$HOME/local/bin:$PATH"
```

Remember that this setting will take effect only after you open a new terminal window or a new login.

To test the setup, try running `arm-none-eabi-gcc --version`. It should have the following effect:

```
$ arm-none-eabi-gcc --version
arm-none-eabi-gcc (GCC) 4.6.3
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

3.4 Compiling libopencm3

Libopencm3 is a standard peripherals support library for ARM Cortex M3 microcontrollers. The version used below contains patches for VESNA that have not yet been accepted up-stream.

```
$ git clone https://github.com/avian2/libopencm3.git
```

Now compile and install the library. Adjust the DESTDIR value to the same prefix you used when installing the toolchain.

```
$ cd libopencm3
$ make
$ DESTDIR=~/.local make install
```

3.5 Testing the setup

To test the setup we will compile, upload and debug a short program that blinks the yellow LED present on the VESNA core board and sends a message to debugger's serial port.

First, download the source of the program:

```
$ git clone https://github.com/avian2/vesna-hello-world.git
```

Then compile it:

```
$ cd vesna-hello-world
$ make
```

And load it into microcontroller's flash. For this to work, you need to have an Olimex ARM-USB-OCD programmer connected to VESNA through the SND debug board. Make sure the power and JTAG connectors are securely attached.

```
$ make hello-world.u
```

The yellow LED on the VESNA board should start blinking after programming is complete. To see messages sent to the serial port, you need to attach the serial connector on the Olimex ARM-USB-OCD to the VESNA's serial port.

You then need to read the correct device file. If you don't have any other USB-to-serial adapters connected to the machine, this will usually be `/dev/ttyUSB0`. The program sets the baud rate to 19200 bps, and we have to match that setting on the Linux side.

```
$ stty -F /dev/ttyUSB0 19200
$ cat /dev/ttyUSB0
```

You should now see a stream of **Hello, world!** messages appear in the terminal. Of course, you can also use your serial terminal emulator of choice

Note that on most setups your user account needs to be in the `dialout` group in order to have access to serial adapters.

3.6 Debugging setup

OpenOCD allows you to attach a GNU debugger instance to VESNA. This allows you to conveniently debug the code running on the microcontroller, for instance by manually inspecting memory contents or single stepping through code.

We begin by starting OpenOCD in server mode.

```
$ openocd -f interface/olimex-arm-usb-ocd.cfg -f target/stm32f1x.cfg
```

Then we can start GNU debugger and attach it to OpenOCD.

```
$ arm-none-eabi-gdb
(gdb) target remote localhost:3333
```

At this point we can start debugging the code that is already running on the microcontroller using the usual GDB commands. To reset the microcontroller, use:

```
(gdb) monitor reset halt
(gdb) continue
```

You can also program the microcontroller's flash directly from the debugger. For instance, to load the hello-world program, do:

```
(gdb) monitor reset halt
(gdb) file hello-world.elf
(gdb) load
(gdb) monitor reset halt
```

Note that to see the source of the libopencm3 functions correctly, you might need to adjust the source search path. Adjust the path in the command below to point to the location where you compiled libopencm3.

```
(gdb) directory ~/libopencm3/lib/stm32/f1
```

3.7 Troubleshooting

This section tries to give solutions to common problems. The list might be incomplete, so feel free to contact the author for including new entries in this section.

3.7.1 unable to open ftdi device: device not found

`openocd` will report this error if initialization failed, even if the USB device itself was actually found. Common reasons are access denied errors due to bad device file permissions or `ioctl` errors because OpenOCD expects a different type of hardware.

All instructions above assume you are using the Olimex ARM-USB-OCD programmer. Note that Olimex ARM-USB-OCD-H is a different device, requiring a different OpenOCD configuration.

If you are using a different programmer, you have to set the `OPENOCD_PARAMS` environment accordingly. For instance, for Olimex ARM-USB-OCD-H, the following line can be added to the top of the `.bashrc` file:

```
export OPENOCD_PARAMS="-f interface/olimex-arm-usb-ocd-h.cfg \
-f target/stm32f1x.cfg"
```

Also note that at the time of writing, `openocd` packages for Debian and Ubuntu do not include proper `udev` rules for ARM-USB-OCD-H hardware and hence do not set correct permissions for device files in that case. This can be corrected by adding the following configuration to a file named `/etc/udev/rules.d/40-openocd.rules` (unplug the programmer beforehand):

```
ACTION!="add|change", GOTO="openocd_rules_end"
SUBSYSTEM!="usb", GOTO="openocd_rules_end"
ENV{DEVTYPE}!="usb_device", GOTO="openocd_rules_end"

# Olimex ARM-USB-OCD-H
ATTRS{idVendor}=="15ba", ATTRS{idProduct}=="002b", MODE="664", \
GROUP="plugdev"

LABEL="openocd_rules_end"
```

References

- [1] Marko Mihelin, SensorLab VESNA open source development environment setup manual for Windows based development, 2011