genoQs Machines

Development Environment Setup

Setting up a development environment for genoQs Nemo and Octopus using the legacy heart board

Gabriel Seher, Wilson Stockman
2025.05.12

Contents

Hardware overview

This document describes the installation and configuration of a development environment for the genoQs Nemo MIDI sequencer. The described procedure is easily transferrable to Octopus as well, since Octopus and Nemo share the same computing hardware.

Before proceeding with the development environment setup, it may be worth reviewing a few key bits of information about the hardware that we are developing for.

The genoQs heart-board (the red printed circuit board inside the machine – Nemo and Octopus equally) is an in-house development. The design is based on the ARM Evaluator 7T board and as such features many traits of that board.

Below is an overview of the relevant components. Further details can be found in the respective manufacturer data sheets which are still available on the internet.

Component	**Model**	**Board Features**
CPU	Samsung KS32C50100 / Samsung S3C4510 Note: In the early 2000's Samsung renamed / renumbered their ARM Processor models. Both these model numbers refer to the same device.	- 16/32-bit RISC architecture - ARM7TDMI core - JTAG-based debug solution - Two UARTs - 18 programmable I/O ports - 21 interrupt sources, including 4 external - 10MHz clock (the processor uses this to generate a 50MHz clock)
SRAM	Samsung K6R4016V1D	- 2 MB RAM
Flash Storage	AMD Am29LV400	- 4 MB Flash

Overview of the development environment

The development environment for the genoQs Nemo consists of the following building blocks:

1) **Base operating system**: the base OS for the host development machine, here Ubuntu Linux 22.04.3 LTS running in a VM. Source: ubuntu.org.

2) **Make and 32-bit support**: components that need to be installed on a basic 64-bit Ubuntu to run the build process of our source code. 32-bit support is required by the applications in the cross-toolchain and 'make' is required to execute the build process per se.

3) **eCos and the arm-eabi cross-development toolchain**: this is used to compile and inspect (as in debug) the code on the host machine (our Ubuntu VM) in such a way that it runs on the target machine (the genoQs heart board). Source: eCos Sourceware.

4) **genoQs source code**: the source code of the Nemo and Octopus Firmware provided in the version of the latest public release. Source: www.genoqs.net

5) **Customized eCos runtime:** this is the eCos OS library tailored and compiled specifically for the genoQs heart board. It provides key operating system functionality used by the genoQs source code (e.g. threads, mailboxes, semaphores, etc.) that make up the real-time OS character of the genoQs firmware. Source: www.genoqs.net

6) **VSCode:** we use VSCode as an integrated development environment (IDE). At the time of writing free and based on open source. Source: Ubuntu software repositories.

7) **OpenOCD:** used as an on-chip debugging server. This block is technically optional but by all means it is the most essential piece for any meaningful development effort. Here we use OpenOCD as a software (free and open source, from openocd.org) in tandem with the Olimex ARM-USB-TINY dongle (which needs to be purchased separately).

The next sections of the document will describe each building block in detail.

1. Installing the base operating system

Download the OS image

For this environment we chose a free and easy to install Linux distribution: Ubuntu. The environment described here will be installed into a **VM in VMWare** on Windows 11. Note: it may also be possible to install a similar environment on hardware or on an Intel-Mac under Parallels.

We need to download the OS installation image file from Ubuntu:

```
http://www.ubuntu.com/
```

Select the options for **Ubuntu 22.04 LTS Desktop** (or latest LTS version) and download the image file. When the download completes you should have a file with a name like "ubuntu-22.04.3-desktop-amd64.iso" or similar, according to the version of your Ubuntu.

Create a VM and install the OS

Create a VM following the relevant VMWare documentation and **install Ubuntu** from the downloaded image. You may choose any **username** you like, for the creation of this document we used 'genoqs'.

The **minimal configuration** offered by the Ubuntu installer is sufficient for our purposes. After the installation is complete, the machine will be rebooted.

Once the machine has rebooted, log in as the user you created during installation. Make sure that the fresh VM has **access to the internet** as we will require several items to be downloaded.

Should there be any issue with internet access, make sure you have the right entries for **nameservers**. Check the /etc/resolv.conf file:

```
$ sudo nano /etc/resolv.conf
```

Next, add Google's public DNS servers to the file, with the nameserver keyword followed by the IP address of the DNS server, as follows:

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Let's make sure to start with a fresh system. Now run:

```
sudo apt update && apt upgrade
```

Confirm any prompts asking for permission to continue.

2. Installing make and 32-bit support

Open a terminal window.

Installing make

We need make to be installed on our system. If it is not, you will not be able to run the makefile controlling the build process. To install make, issue the following command:

```
sudo apt install make
```

Installing 32-bit libraries

Our cross-toolchain (arm-eabi) consists of a set of 32-bit applications, while the standard architecture of our platform is 64-bit (at the time of writing). To run our 32-bit executables on a 64-bit Ubuntu system, we have to add the i386 architecture and then install several library packages.

Issue the commands below to install the **32-bit libraries** that are required:

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libncurses5:i386 libstdc++5:i386 \
libstdc++6:i386 libc6:i386 zlib1g:i386
```

If you are prompted to confirm the installation, please do so.

Note: We received reports that `libncurses5` may no longer be distributed in the Ubuntu repositories. Should you encounter this problem, you will have to download and install the deb file manually.

Some possibly helpful references:

- https://stackoverflow.com/questions/73464111/libncurses-so-5-library-missing-in-ubuntu-22-04-installation-for-arm64-rapsberr
- https://packages.ubuntu.com/focal/i386/libncurses5/download
- `wget http://security.ubuntu.com/ubuntu/pool/universe/n/ncurses/libn\curses5_6.2-0ubuntu2.1_i386.deb`
- `sudo dpkg -i libncurses5_6.2-0ubuntu2.1_i386.deb`

eCos installation

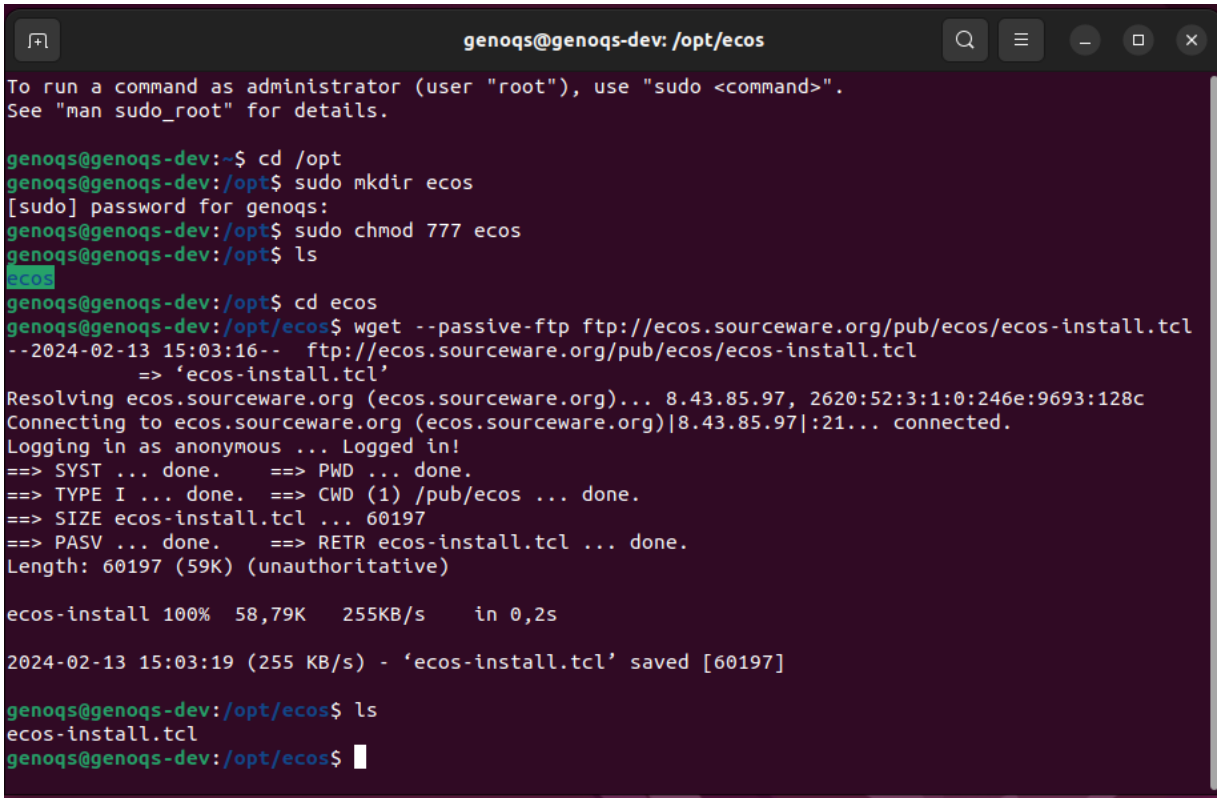
We will install the eCos environment into a **new directory**, in this case **/opt/ecos**. This directory needs to be created first and given proper **permissions** before we can proceed with downloading eCos.

Enter the commands below in the terminal window. You may be asked for your password, which allows you to run certain commands as an admin user. Use the password of your login user.

```
$ cd /opt
$ sudo mkdir ecos
$ sudo chmod 777 ecos
$ cd ecos
```

The **eCos installation** is done with the help of an **installation utility**, which we download with the following command:

```
$ sudo wget --passive-ftp ftp://ecos.sourceware.org/pub/ecos/ecos-install.tcl
```



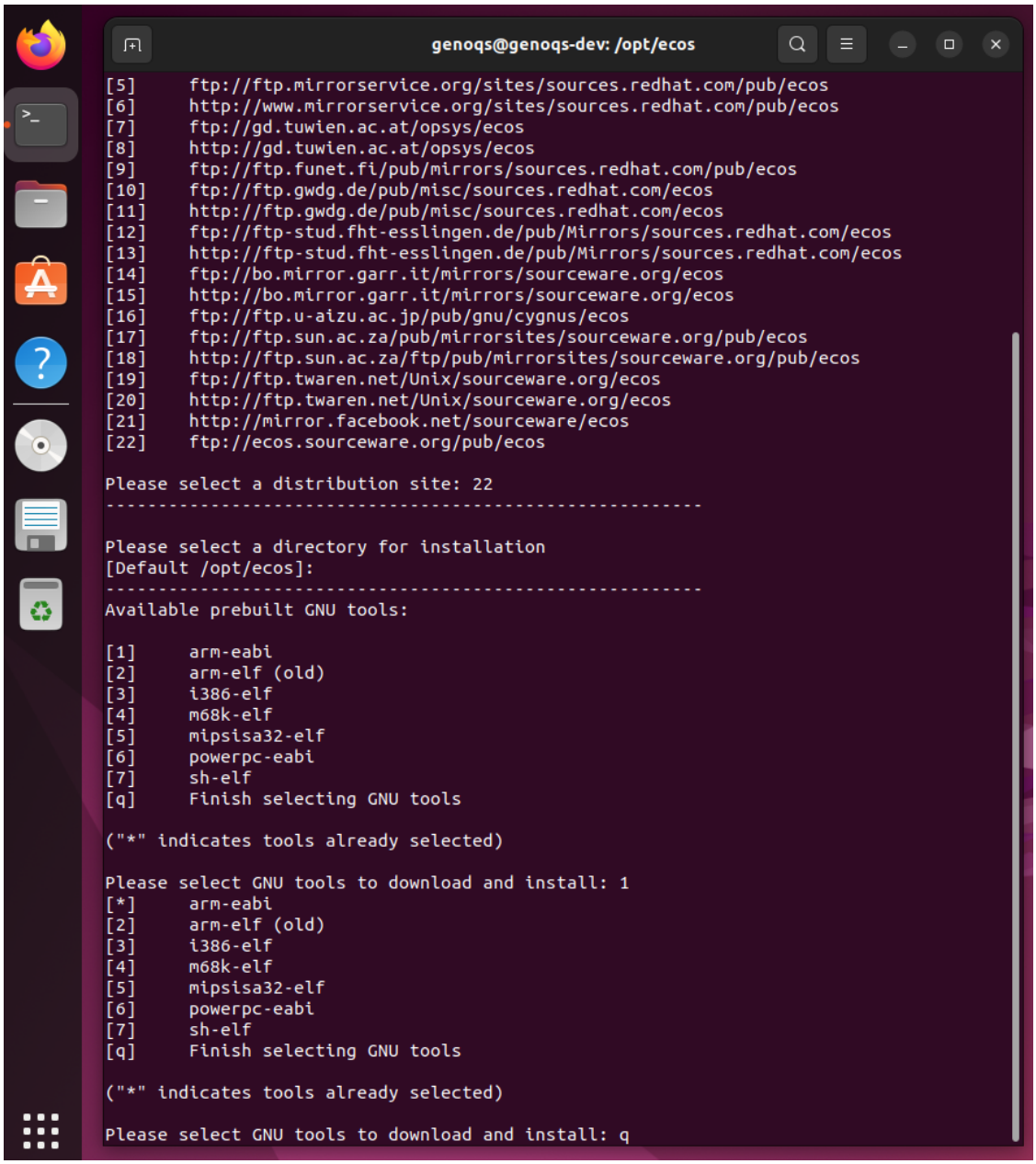
Once the **installation utility** is downloaded, **execute** it issuing the following command:

```
$ sudo sh ecos-install.tcl
```

You should see the ecos Installer start.

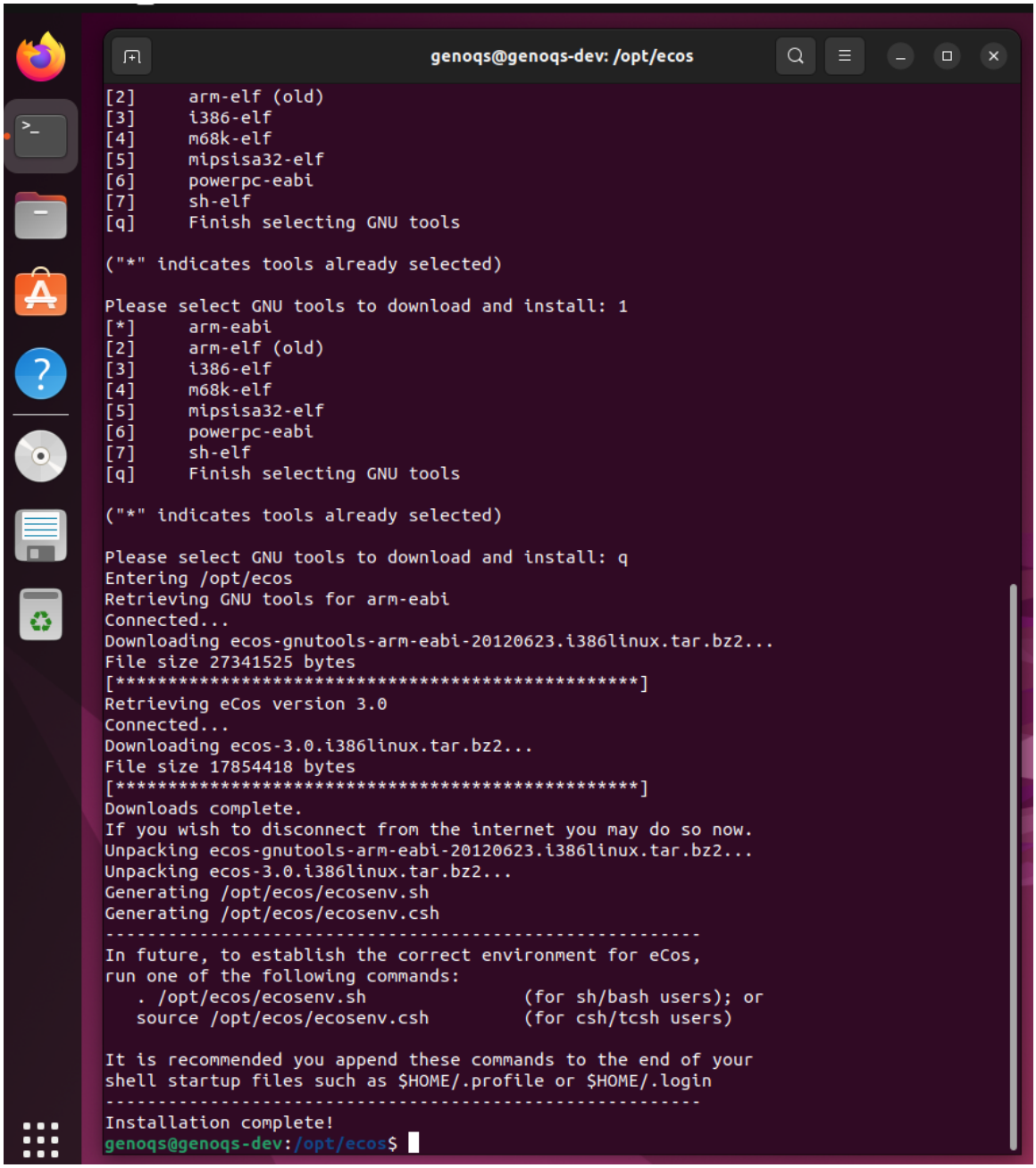
Select a **download site** and then accept the default download location (which should be /opt/ecos/).

Choose to install the "**arm-eabi**" toolchain, and then select **q** to finish the selection. Your screen should look like below:



Once you press Return the files should download and install automatically.

The installation of eCos should conclude with the message "**Installation complete**!".



IMPORTANT: to have the eCos environment working properly we now need to set the **PATH** correctly. Once ecos installation is complete, **invoke from the ecos install directory** (/opt/ecos/):

```
$ source ./ecosenv.sh
```

3. Downloading the genoQs source code

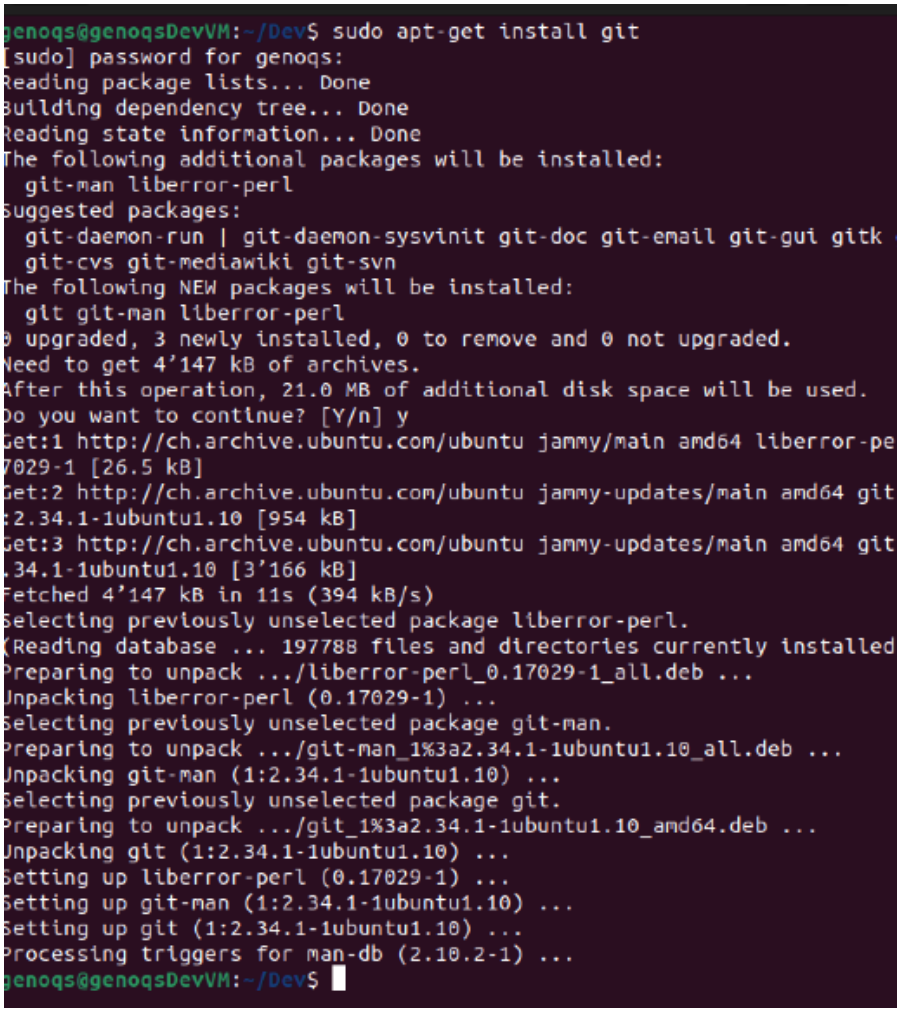
The source code files of the genoQs machines are available from the genoQs Github page. Using Github as the main code repository ensures that the developer has a direct latest copy of the code and is ready to work/collaborate (pushing code, branching etc.).

```
https://github.com/genoqs-community/source
```

You can download(clone) the code publicly, however if you would like remote write permissions please get in touch or raise a pull request.

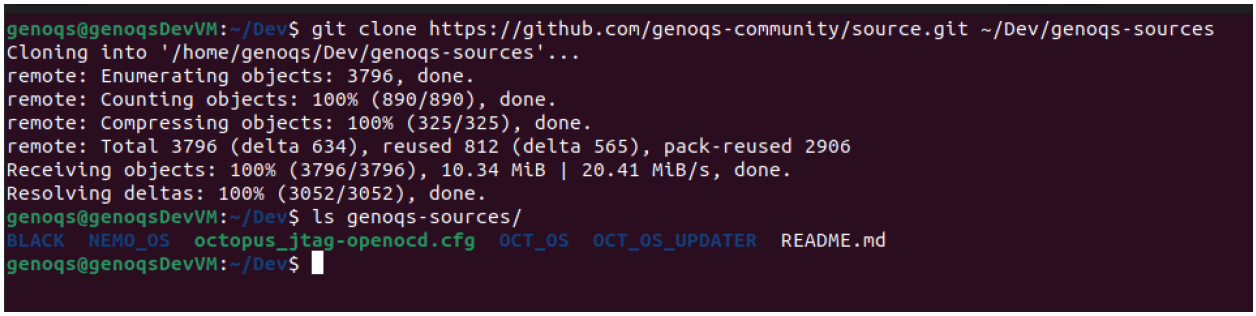
Install the git client tool

```
$ sudo apt-get install git
```



Create a working directory **~/Dev** and clone the latest code into it:

```
$ mkdir ~/Dev
$ cd ~/Dev
$ git clone https://github.com/genoqs-community/source.git \
~/Dev/genoqs-sources
```



3. Optional - only needed for remote pushing etc.: setup a Github user at www.github.com.

If you would like to push your changes of the code into the master branch get in touch with the repository owners via Github.

4. Downloading the eCos runtime

The **eCos runtime library** is required for the compilation of the source code we downloaded in step 4. The runtime library customized for our hardware can be downloaded from the genoQs site. The provided library is configured to support the genoQs heart board (which Nemo and Octopus share) and compiled with the toolchain provided with ecos-3.0, which we have installed in step 3.

Note: building the eCos library with another (newer) toolchain may be possible and feasible. This may be required at some point in the future where incompatibilities between the toolchain and the used IDE may appear. Instructions on how to build the eCos runtime library are provided as an appendix.

Download

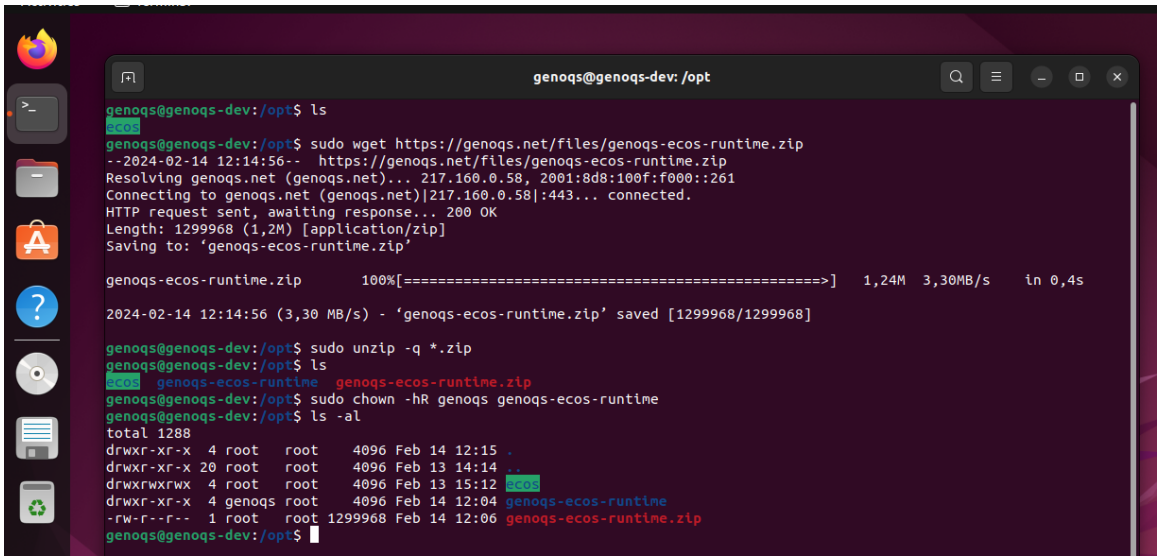
Now we need to **download** the runtime library and eCos base files supplied by genoQs. Issue the following commands to download the required files to the /opt directory:

```
$ cd /opt
$ sudo wget https://genoqs.net/files/genoqs-ecos-runtime.zip
```

Unzip the file and then **change ownership** of the files to your login user (here assuming it is genoqs) by issuing the following command:

```
$ sudo unzip -q *.zip
$ sudo chown -hR genoqs genoqs-ecos-runtime
```

Your screen may look similar to the below:

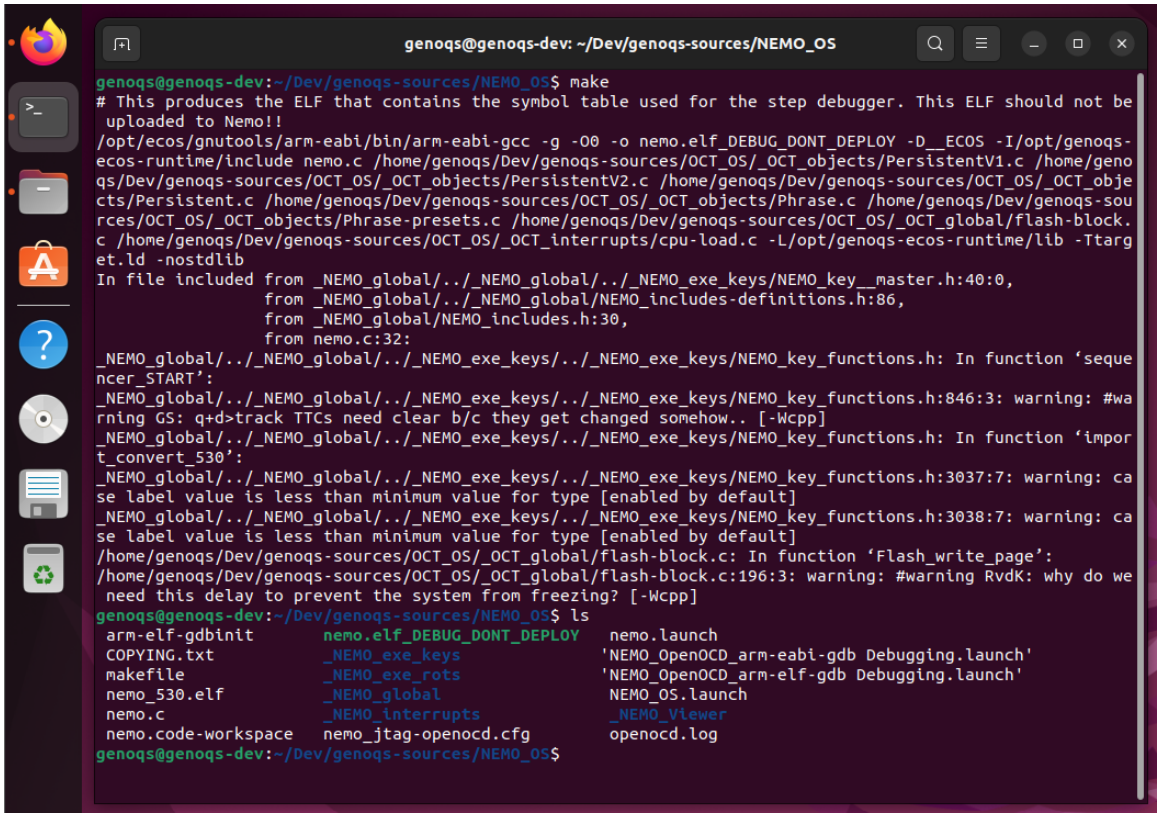


Verification

At this point we should be able to **compile the code**. Let's verify it:

```
$ cd ~/Dev/genoqs-sources/NEMO_OS/
$ make
```

This should produce a screen like the below. Note that some **warnings** are manually triggered and may appear, but they do not prevent the code from compiling. Once the build process is complete you should see the **output file** located in the directory; this is your **newly compiled OS**!



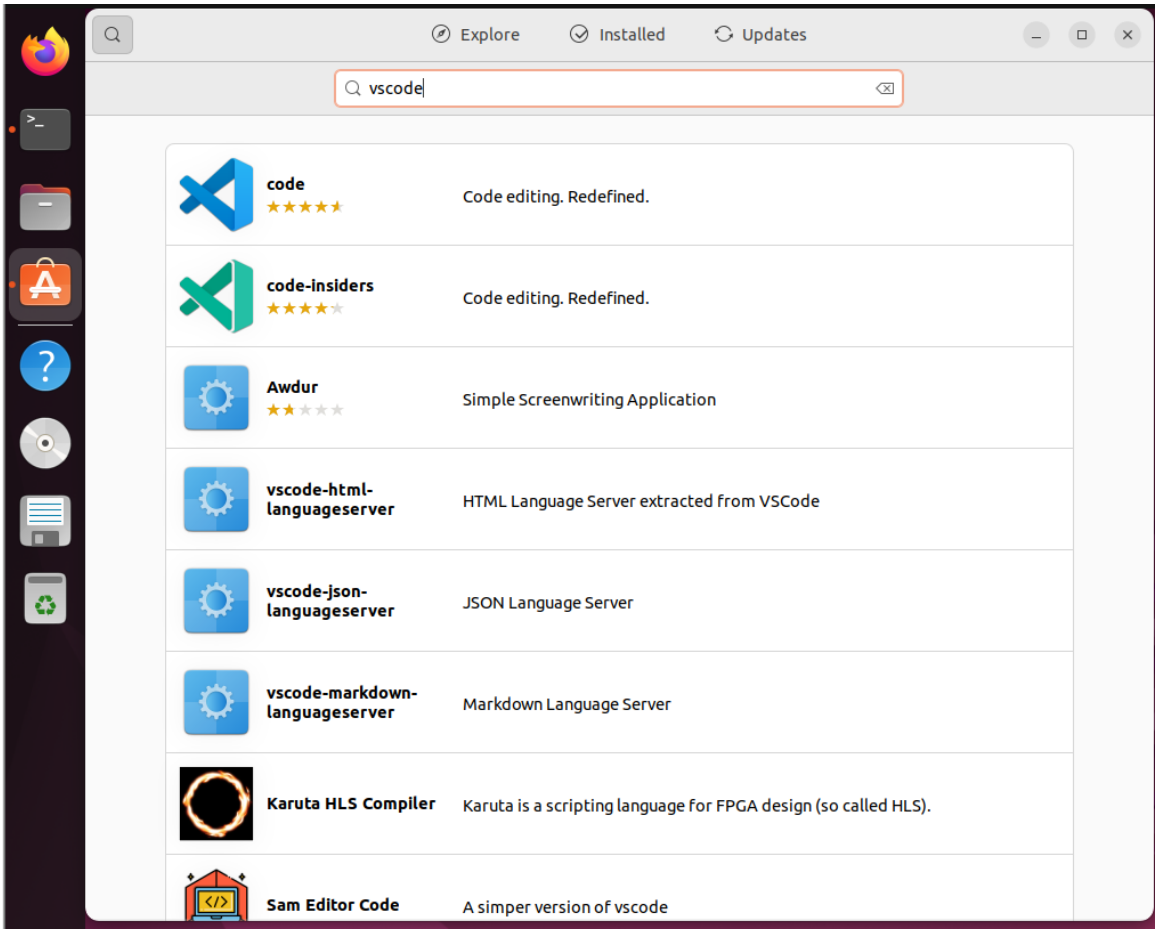
In the example above the output file is nemo.elf_DEBUG_DONT_DEPLOY.

Note: this particular file is created for development purposes and contains debugging symbol tables. It can be run in the development environment but **should under no circumstances be flashed to Nemo!** Please see the makefile for details.

5. Installing and configuring VSCode

Installation

The easiest way to install VSCode as our IDE (integrated development environment) is from the Ubuntu Software Manager. Open the Software Manager and in the search bar (you may need to click the top left icon) type **'vscode'**.



Click on **'code'** and then on the green 'Install' button. Once the installation is complete, open a new Terminal window and **start VSCode** by simply typing in:

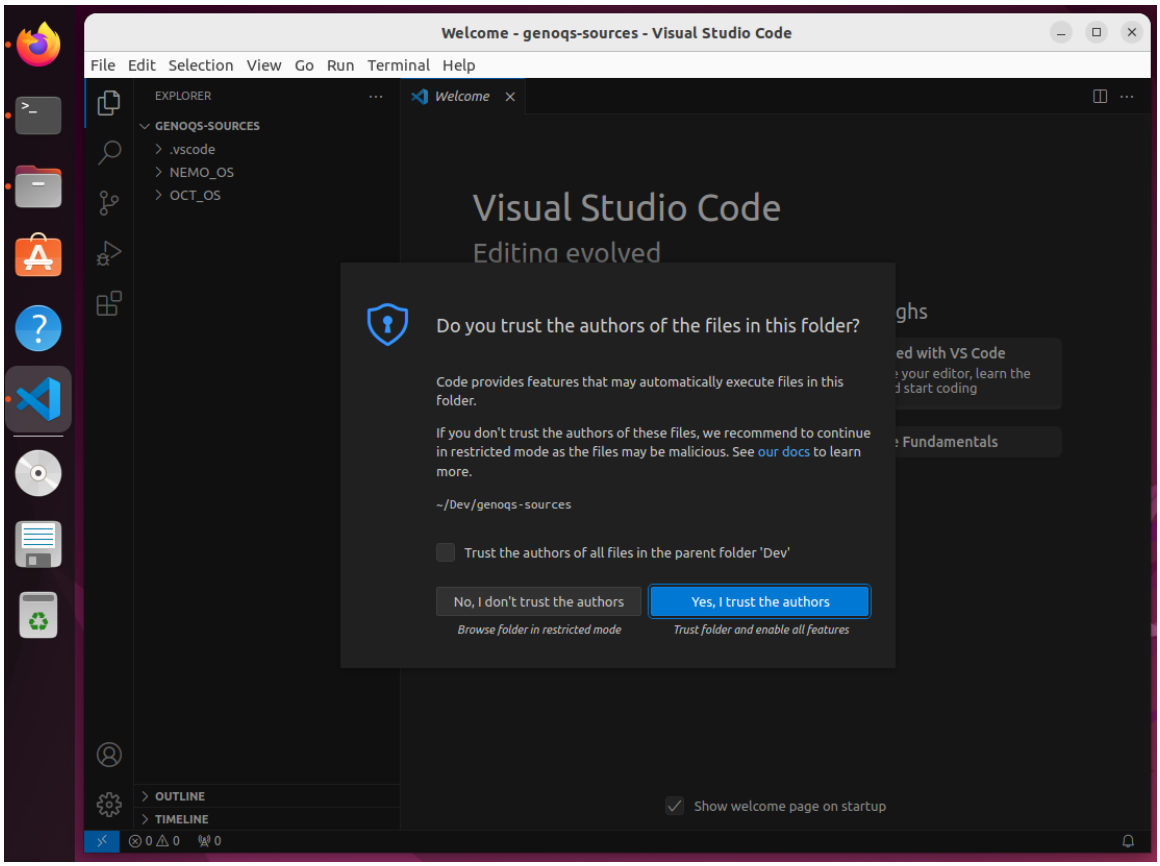
```
$ code
```

Along with the startup window you should see the VSCode icon in the Dock. We recommend making it stick to the Dock (Right-Click on the icon -> **Add to Favorites**).

Setting up the Build configuration

In a first step we will link the IDE graphical user interface (GUI) to the build environment. The objective here is to have a nice **view of the source code** via the IDE editor and at the same time be able to **build the executable** from the GUI elements of the IDE.

In a first step we need to **link our source code to a VSCode project**. For this, on the appearing VSCode Welcome page, select 'Open Folder'. In the appearing window select **Dev/genoqs-sources** -> Open. You may get another prompt asking if you trust the authors of this folder, you can confirm: Trust folder and enable all features. Your screen may look like the below:



Install the **Makefile Tools** extension: since our code uses make and a makefile for compilation but VSCode does not support makefiles from the get-go, we need to install the Microsoft-provided extension to enable VSCode to work with makefiles. For this, do the following:

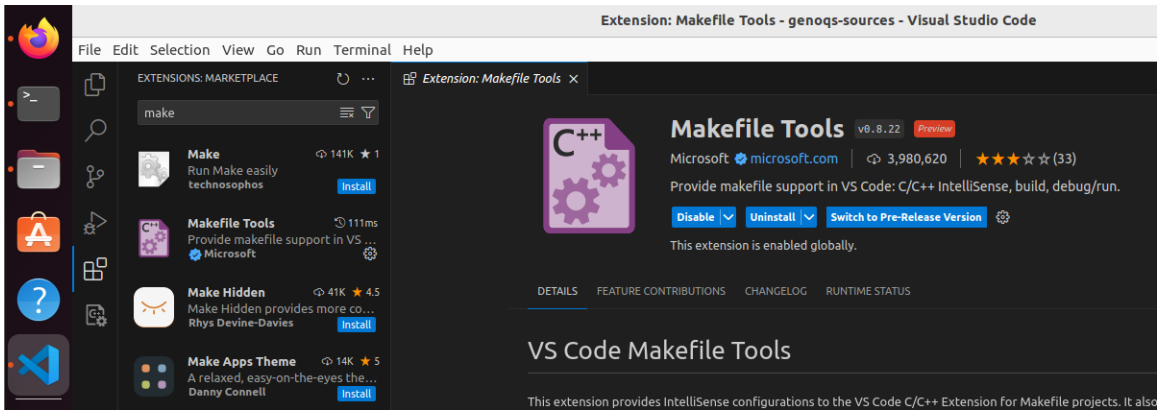
Click on the **Extensions** icon in the VSCode window (the one with the square building blocks)

In the search field type **'make'**.

Under the search results you should see one named **'Makefile Tools'** provided by Microsoft.

Click on **Install**.

Once you do so, you should see a screen similar to the below.



Now we need to **configure Makefile Tools** for our environment. Here we will configure it to build the Nemo firmware.

With the extension 'Makefiles Tools' installed we now have an **icon** for it in the dock of VSCode. Click on the extension icon which will bring up a few setting options. Make sure that two values are set: the **Make executable** to ‘make’ (resolving to /usr/bin/make) and the **Makefile location** to NEMO_OS (resolving to home/genoqs/Dev/genoqs-sources/NEMO_OS/makefile).

Invoking **Build** from the tool menu of the Makefile Tools extension should now trigger the build process just like we have done it by invoking make on the command line in step 5.

Completing the install

To complete the base setup of VSCode we need to install the **C/C++ extension,** as well as the **C/C++ extension pack.** These provide both some comfortable IDE features (Intellisense etc.) as well as some key functionality needed later to set up the debugging environment in VSCode. The installation can be done in a very similar fashion to that of the Makefile Tools.

6. Setting up the on-chip debugger

We will use **OpenOCD** as a debugging server, which in our setup will run on our Ubuntu VM and translate commands issued by **arm-eabi-gdb** (our debugger) to something that the genoQs heart board understands, and vice versa.

The JTAG dongle

Communication between OpenOCD and the heart board takes place with the help of a JTAG dongle. The dongle that our setup is based on and proved to be working (with the provided configuration file in the source package) is the **Olimex ARM-USB-TINY.**

`https://www.olimex.com/Products/ARM/JTAG/ARM-USB-TINY/`

Note that other JTAG dongles may work, however, we have found that even very subtle differences in the hardware could prevent the debugging environment from working properly. For example, the Olimex ARM-USB-TINY works, however, its improved, more performant, and highly praised sibling, the Olimex ARM-USB-TINY-H did not work properly in our case. Therefore, if you plan to purchase an Olimex ARM-USB-TINY dongle, make sure to precisely get that 'plain' version (and not the -H version).

For the installation and setup of communication with your computer you will likely need to install FTDI drivers, according to the Olimex specification. Details about this can be found on the product page for the dongle (link above) or on ftdichip.com.

Installing OpenOCD

To **install OpenOCD** we follow the instructions provided at openocd.org. In our case this means issuing the following command:

```
$ sudo apt-get install openocd
```

Once installed, OpenOCD can be started from the command line by simply invoking 'openocd'. You may **try it** although no, we are not ready for prime time yet; but invoking the command confirms whether the installation was successful or not. If the installation was successful you should see an output from OpenOCD stating that it can't find the openocd.cfg file. With the installation confirmed, we need to provide OpenOCD with the right configuration for our environment.

On the one hand it needs to know **which adapter** we are using and on the other, **which CPU target** it is talking to. Both pieces of information are captured in the configuration file **nemo_jtag-openocd.cfg** which is found in our source code folder NEMO_OS, which we get to in a minute.

7. Connecting the hardware via JTAG

We now turn to the hardware side of the setup. Make sure the **genoQs machine** (Nemo in this case) is **turned off and disconnected from the power source**.

Now **open the enclosure** and gain access to the heart board (the red PCB). The heart board features a JTAG connector. **Attach** the flat cable that comes with the Olimex adapter to this connector. There should be only one possible way to attach it due to the shape of the connector.

Once the Olimex adapter cable is connected to the heart board you may reconnect the power source and turn on the machine.

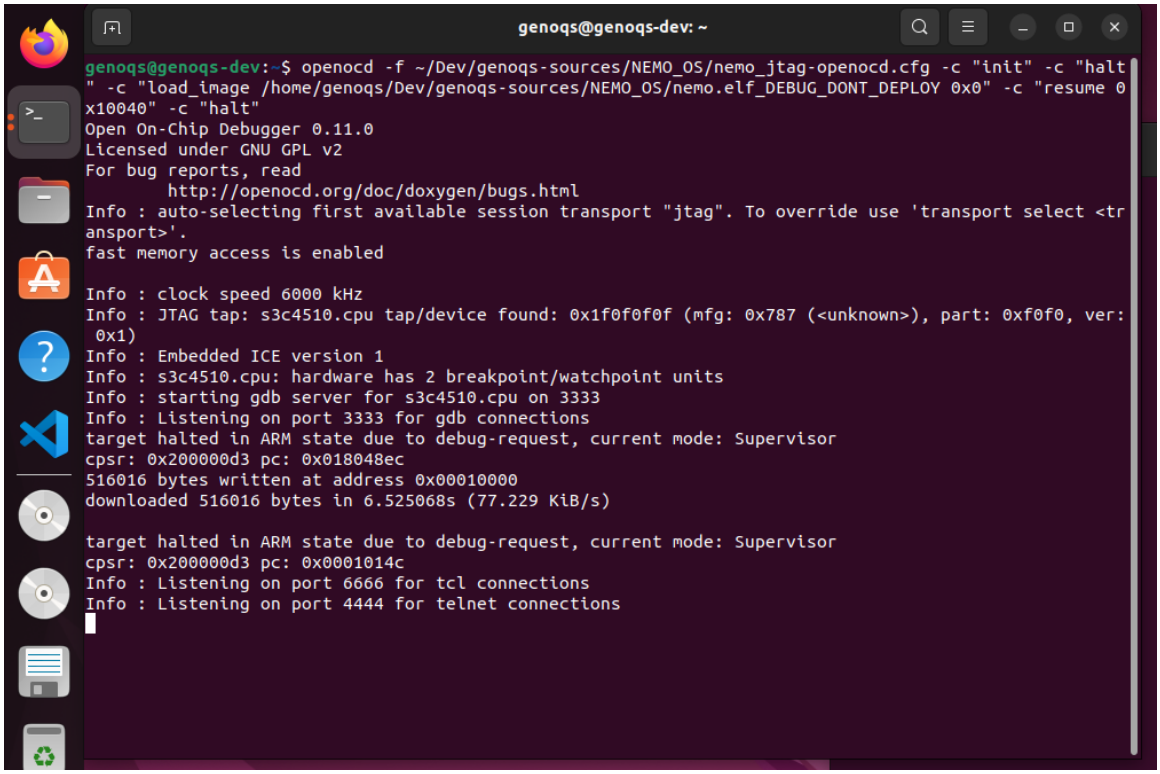
Back on the software side, and with OpenOCD installed, **connect the Olimex ARM-USB-TINY adapter to a USB port** on the computer that is hosting our VM and make sure that the respective USB device (the Olimex adapter) is also logically **connected to your VM** (and not the host).

Running a debugging session from the Terminal

1. Start an OpenOCD session: in a terminal window (which we shall call **Console1**) we will start the OpenOCD server and pass it a few commands that get the heart board ready with a loaded image at start address 0x0, resumed, halted and ready for gdb connection. Issue the command:

```
$ openocd -f ~/Dev/genoqs-sources/NEMO_OS/nemo_jtag-openocd.cfg -c "init" -c "halt" -c "load_image /home/genoqs/Dev/genoqs-sources/NEMO_OS/nemo.elf_DEBUG_DONT_DEPLOY 0x0" -c "resume 0x10040" -c "halt"
```

This command should produce an output similar to:



2. Load arm-eabi-gdb: open an additional Terminal window (which we shall call **Console2**) and issue the following command:

```
$ /opt/ecos/gnutools/arm-eabi/bin/arm-eabi-gdb ~/Dev/genoqs-sources/NEMO_OS/nemo.elf_DEBUG_DONT_DEPLOY
```

The last line of the output from gdb should read: "Reading symbols from /home/genoqs/Dev/genoqs-sources/NEMO_OS/nemo.elf_DEBUG_DONT_DEPLOY...done."

3. Connect to openOCD and set breakpoints: while still in **Console2** we issue the following commands in sequence on the gdb command line. **Important** to add the first break in cyg_user_start before adding the break NEMO_key__master.h:80 - otherwise we see an error.

```
$ target extended-remote localhost:3333
$ break cyg_user_start
$ continue
$ break NEMO_key__master.h:80
$ continue
```

After the last **continue** command we should see the **Nemo blink as usual**, and running the freshly uploaded firmware. However, there is a breakpoint set at a place in the code, where key presses are interpreted. This means, **once we hit a key,** we should see it break the execution.

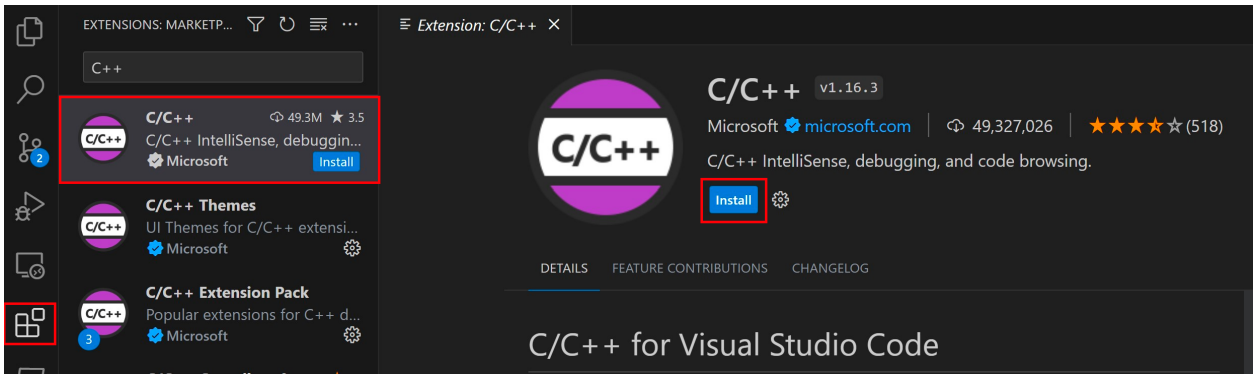
4. Start debugging: now hit any button on Nemo. The debugger should catch that button press and break accordingly. You may now use the gdb console interface to inspect registers, variables, etc.

At this point we are certain that the connection between the genoQs machine and our environment is working and that we have a working on-chip debugging environment!

The last step in our setup is to integrate the debugging environment and VSCode.

Integrate VSCode and GDB/OCD

Start VSCode and go to the Extensions panel. If not already installed (last part of Step 6) it is now time to **install the extension C/C++**. This extension includes the debugger adapter plugin vppdbg which we require, and which supports configurations of OpenOCD etc.

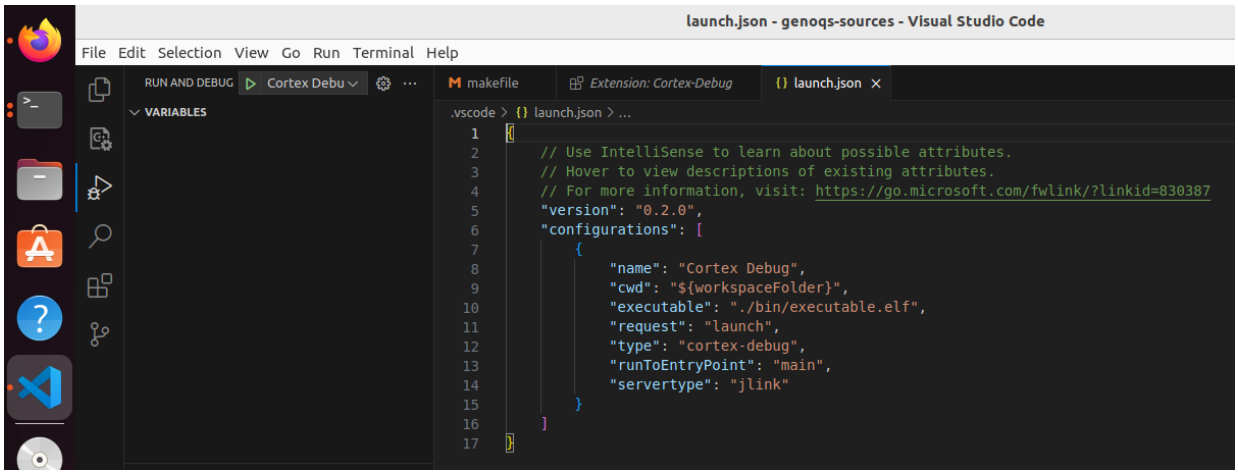


Go to the **Run and Debug** panel.

Click on the **'create a launch.json file'** highlighted text.

Then in the appearing menu select **C/C++: (gdb) Attach** (the first option from the top). This will create and present us a generic launch.json file in the editor.

The launch.json file stores configurations to be used and invoked by the Run and Debug functionality. Therefore, we need to adjust the launch.json file according to our needs.



The launch.json file resides in an invisible folder, in our case at the following location: `genoqs-sources/.vscode`

We will replace the freshly created generic `launch.json` file with one that is already prepared for our environment. We can obtain it from the genoqs site and put it into the right location with the following command sequence:

```
cd ~/Downloads
sudo wget https://genoqs.net/files/launch.json
cp launch.json ~/Dev/genoqs-sources/.vscode
```

Go now to the **Run and Debug** panel again - there should be a configuration called **NEMO OpenOCD** (in the top menu).

Select it.

Now **press the green triangle** button ('Play' or 'Run'..). This should start the make process followed by the upload to our machine.

Congratulations! From here you should be able to use the debugging facilities provided by VSCode.

Appendix A: Building the eCos runtime library

The procedure to build eCos is described on the ecos page here: https://ecos.sourceware.org/docs-3.0/user-guide/using-ecosconfig-on-linux.html. Even at the risk of duplication, below are the steps to configure and build the ecos runtime from our downloaded sources.

The eCos runtime library can be compiled with the eCos configuration (ecosconfig) tool included in the ecos installation that we have downloaded. To compile eCos from source for our purposes we must achieve two milestones:

1) Build the eCos tree as specified in the configuration file for our specific hardware (file available on the genoqs site) and

2) Inject the memory layout files specific for the genoQs heart board (files available on the genoqs site) into the build tree before we compile it.

Here are the steps to be carried out. It is essential that all steps are carried out exactly and in sequence.

1. Set the **environment variables** using the utility provided by ecos:

```
# Set the environment variables for ecos commands to work
$ cd /opt/ecos
$ source ./ecosenv.sh
```

2. Prepare the **work directory**, where we will build the ecos runtime

```
# Build our workspace where we will compile our stuff
$ sudo mkdir /opt/ecos-work
```

```
# Make sure that the directory can be used to build the ecos tree inside
$ sudo chmod 777 /opt/ecos-work
$ cd /opt/ecos-work
```

3. Create the **ecos build tree** needed for our hardware:

```
# Get the ecos configuration file from the genoqs site.
$ sudo wget https://genoqs.net/files/genoqs-ecos-config.ecc
```

```
# Set the file ownership to prevent permission issues with ecosconfig.
$ sudo chown -hR genoqs genoqs-ecos-config.ecc
```

```
# Check the configuration. Final output line should be 'No conflicts'.
$ ecosconfig --config=genoqs-ecos-config.ecc check

# Create the build tree for the genoqs heart board as target.
$ ecosconfig --config=genoqs-ecos-config.ecc tree
```

4. Inject the **memory configuration files** into the ecos build tree before compiling it:

```
# Move to the Downloads directory
$ cd ~/Downloads

# Download the genoqs heartboard memconfig package
$ sudo wget https://genoqs.net/files/genoqs-heartboard-memconfig.zip

# Inject the heartboard memconfig files into the build tree.
$ unzip genoqs-heartboard-memconfig.zip -d /opt/ecos-\
work/install/include/pkgconf
```

5. Build **eCos**:

```
# Builds the ecos environment. The final output line should read 'build finished'.
$ cd /opt/ecos-work
$ make
```

6. Closing remarks:

The directory /opt/ecos-work/install/ contains the runtime packages needed to compile the code.

`/opt/ecos-work/install` needs to be used as **PREFIX** in our makefile.

`/opt/ecos-work/install/include` needs to be used as **INCLUDE_PATH** in our makefile.

Document version history

Version	Date	Author	Scope and changes
1	2007	John Kimble	Set up an environment based on Ubuntu 7 and Eclipse 3 up to the point of ‘make’. Does not include on-chip debugging.
2	2024.02.21	Gabriel Seher, Wilson Stockman	Instructions to set up an environment based on Ubuntu 22 and VSCode to the point of ‘make’ from within VSCode. Instructions to set up for on-chip debugging via OpenOCD and arm-eabi-gdb from within VSCode. Instructions on how to compile the eCos library from scratch.
3	2025.05.12	Gabriel Seher	Minor corrections and typo fixes.