ECDBoost Embedded Development

Isuru Gunasekara

Department of Systems and Computer Engineering  
Carleton University  
1125 Colonel By Drive  
Ottawa, ON. K1S-5B6 Canada

aguna100@uottawa.ca

**ABSTRACT**

In this paper, the procedure to develop ECDBoost based applications on embedded systems is presented with an example of a line following robot.

# INTRODUCTION

Real-time (RT) and embedded systems can be found in many applications ranging from telecommunications, customer electronics, medical equipment and automated systems. Most embedded systems run on simple microcontrollers with single threaded operations and comprise of very cheap hardware components. But the software that runs on embedded systems are costly to develop and usually involve development and testing on real world systems that tend to skyrocket the cost of development. The reliability of these systems also tend to be affected if extensive testing is not done on these systems. Therefore formal platforms to do simulations of real world scenarios are required for running hundreds and thousands of test scenarios before deploying a system.

# DEMES

Discrete Event Modelling of Embedded Systems (DEMES) is an approach that offers a practical method that allows models to be used throughout the development cycle. DEMES is a modelling and simulation based development methodology based on Discrete Event systems specification (DEVS), which is a discrete event simulation formalism for modelling and simulating dynamic systems. The process to develop a DEMES based system is illustrated in Figure 1. A designer starts by modelling the system with formal specifications, these models are then converted into a DEVS representation, transformed into timed automata and verified using model checking tools. The model components can be tested individually and the physical environment can also be simulated together with the model. The decentralized nature of the model allows individual components to be tested and verified before deployment.

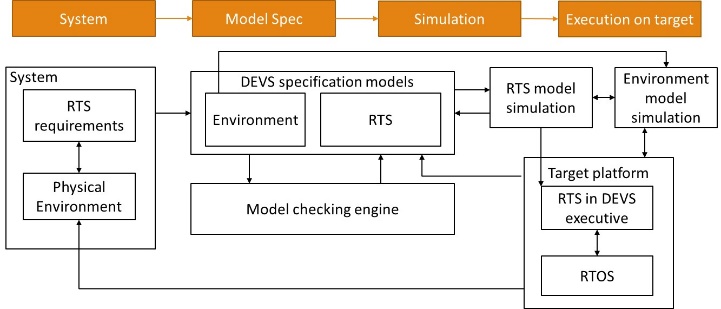


Figure : Development process of a DEMES based system

# Hardware setup

Currently two robots has been tested with ECDBoost and they are, STM32F429 Discovery board [1] by STMicroelectronics coupled with the Parallax robot shield kit [2] and SeedShield robot [3] coupled with the STM32F11RE [4] Nucleo board.

## mbed

The mbed [5] platform is used as the base for controlling the hardware components of the robot. It can be accessed from <https://developer.mbed.org/> and a free account is required to start developing. The first step in order to start developing applications is to create a free account and compile a freely available example application and run it on a target microcontroller.

The mbed platform supports many microcontrollers and development boards. Currently two robots have been tested with mbed and ECDBoost and they are, STM32F429 Discovery board [1] and the STM32F11RE [4] Nucleo board.

The target platform should be selected on the mbed compiler by clicking the selection button in the top right corner as seen in Figure 2. In the dialog box that appears, select the relevant board in order to start developing applications.

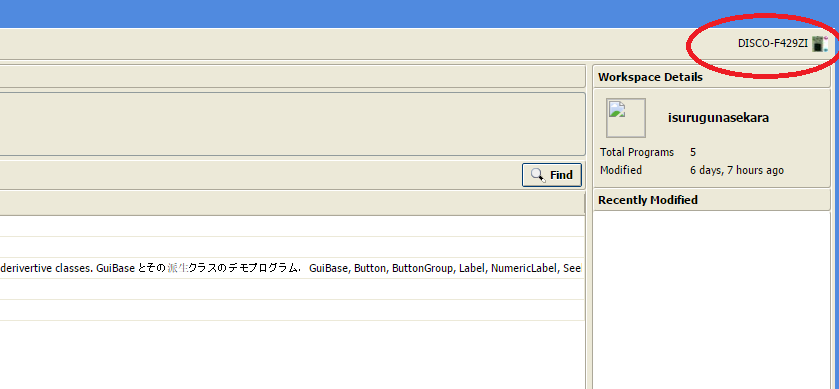


Figure : Selecting target platform in mbed compiler

Import an available example project (such as a “blinky” project) and select, compile to download a hex or bin file that can be programmed into the target microcontroller. The program can also be exported to an offline compiler and IDE such as Keil MDK [6], Eclipse, etc. so that development can be done offline. Right click on the project and select export and select the required IDE to export a zip file. The hex or bin file downloaded from this step will be used in the proceeding steps to program into a microcontroller. Further details about the mbed compiler and using it with hardware devices can be found at [7].

## STM32 driver setup.

The STM32 microcontroller boards used in this setup include an integrated ST-LINK on chip debugger and programmer with a SWD connecter. In addition to the drivers, STMicroelectronics provides a software utility to program microcontrollers. The ST-Link Utility [8] and the USB driver [9] can be downloaded from the STMicroelectronics web pages listed under references. Run the installers in the downloaded files and open up the ST-Link Utility to program the microcontroller board.

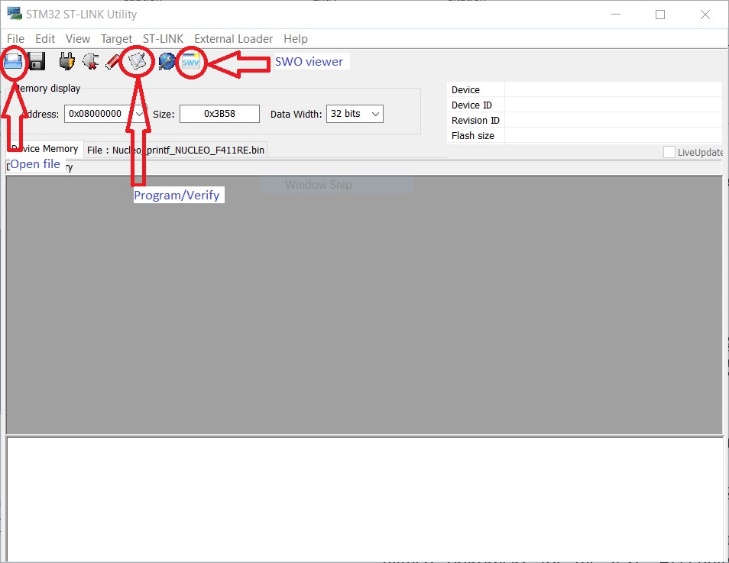


Figure : STM32 ST-Link Utility

Connect the target microcontroller board to the computer using a USB cable. The STM32F4 Nucleo and the STM32F429 Discovery boards are connected using mini USB cables to the computer. The locations of the programming ports can be seen in Figure 4.

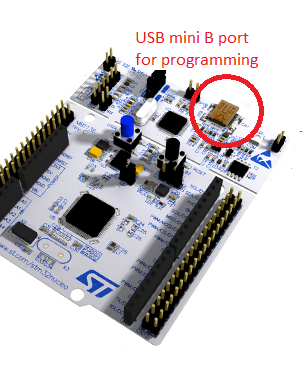
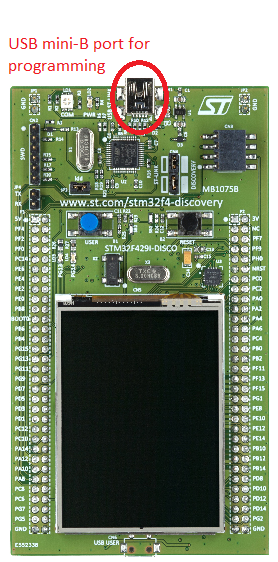


Figure : Programmer ports of the two microcontroller boards used

After connecting the microcontroller board, press the “Connect to the target” button in the STLink Utility and the software will automatically connect and identify the microcontroller board. Open the file downloaded in 3.1 and click the Program/verify button to program the board.

# Toolchain and IDE setup

Next step is to install the ARM toolchain and the Eclipse IDE for compiling, programming and debugging the hardware devices. The GNU ARM Eclipse tool was used for this purpose due to its free nature and unlimited code size. An online version of installation instructions can be found at [10]

## GCC ARM Embedded toolchain

The GCC ARM Embedded toolchain is used to cross compile code on a host computer so that the binary file running on a target microcontroller can be created on a host computer. The toolchain can be downloaded from [11] and select the relevant package according to the host computer setup. For Microsoft windows, download the latest .exe installer file (currently gcc-arm-none-eabi-5\_3-2016q1-20160330-win32.exe) and run it. Run the installer and go through the license page, etc. It is recommended not to change the default installation directory, “C:\Program Files (x86)\GNU Tools ARM Embedded\” since the eclipse plugin that will be installed later detects the toolchain automatically if it was installed in the default location. During the installation, make sure to disable adding the toolchain path to the environment variables by deselecting the options: “Launch gccvar.bat” and “Add path to environment variable”

## Windows build tools

This step is specific to Microsoft windows hosts and this installs two command line programs, “make” and “rm” on your computer. Download the packaged installer from [12] and follow the instructions in the installer.

## OpenOCD

OpenOCD is a free tool that can be used to program and debug microcontrollers straight through eclipse instead of using the STLINK Utility which doesn’t have the ability to perform on chip debugging. Download the latest version from [13] (64 bit or 32 bit depending on the host) and follow the instructions to install the latest version of the OpenOCD Tool. In order for OpenOCD to work, a hardware driver for the target platform should already be installed on the computer. This was done in 3.2.

## QEMU

QEMU is a powerful emulator capable of emulating microcontrollers on host machines so that they can be simulated without any hardware. QEMU can be installed by following the instructions at [14]

## Eclipse and Eclipse plugins

Eclipse can be installed simply by downloading the latest version of Eclipse IDE for C/C++ developers from <https://www.eclipse.org/downloads/> and copying it into any folder and running the eclipse.exe file contained inside. After opening up eclipse, go to “Help -> Install new software” and select the “Add” button and add <http://gnuarmeclipse.sourceforge.net/updates> under the URL section. Select all components as shown in Figure 5 and click next to install.

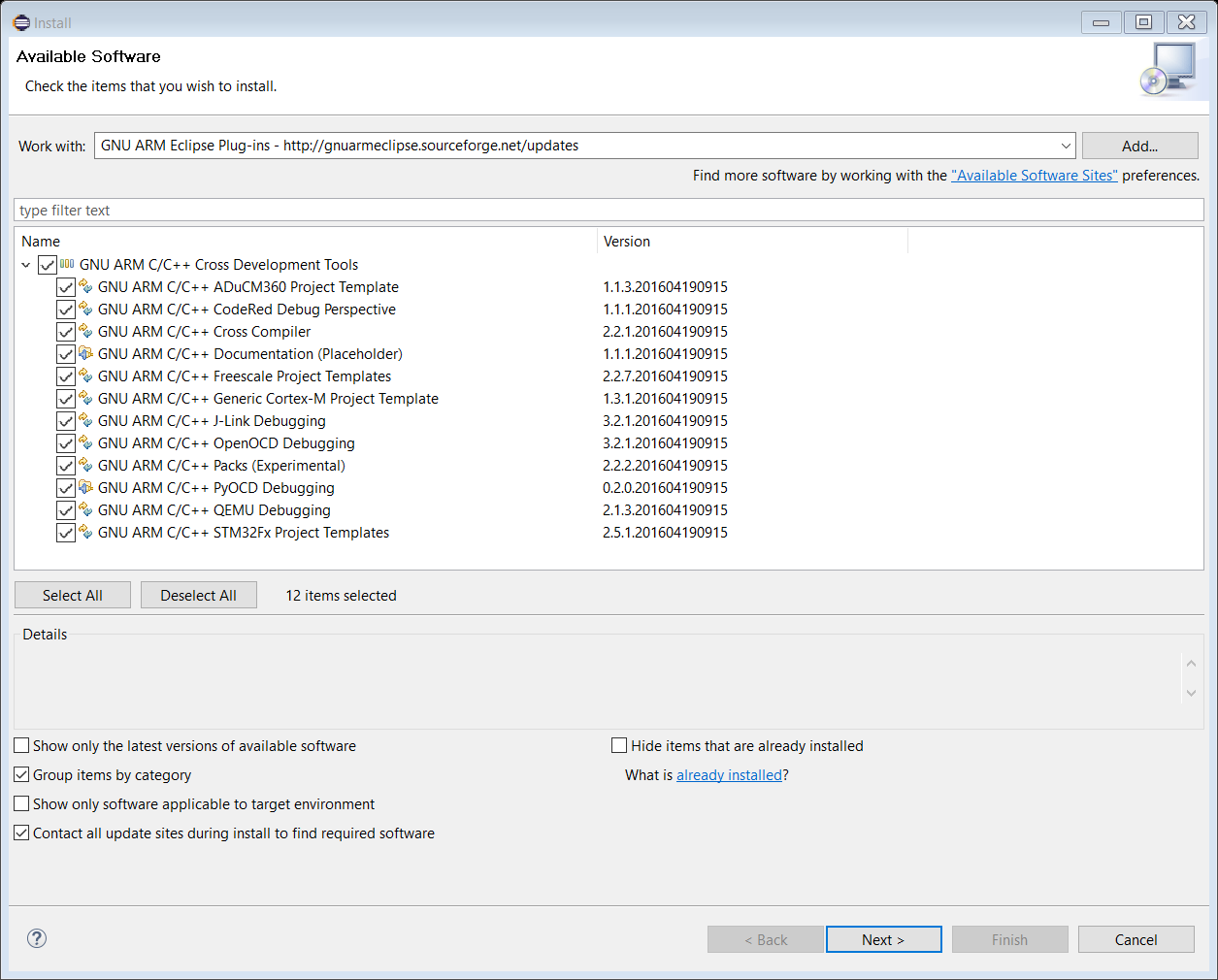


Figure : Eclipse ARM plugins

## Installing hardware pack descriptors

In the Eclipse IDE go to “Window -> Open Perspective -> other” as shown in Figure 6

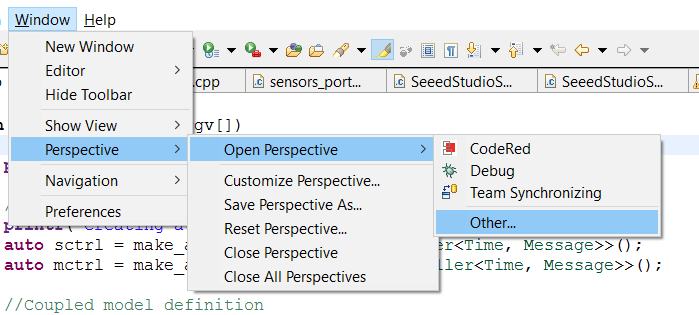


Figure : Changing Eclipse perspective

In the next window that appears select the “Packs” perspective as shown in Figure 7

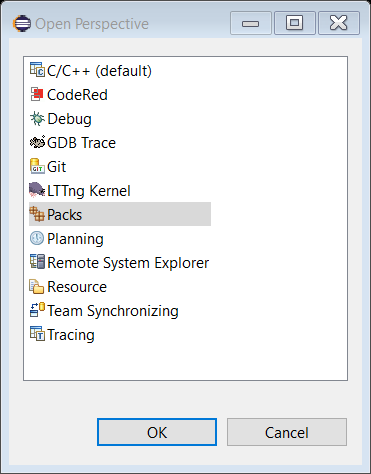


Figure : Packs perspective

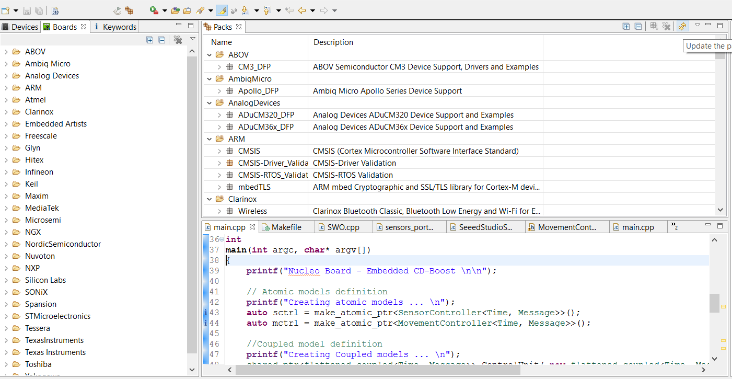


Figure : Eclipse Packs perspective

In the packs perspective, select the refresh button as shown in Figure 9

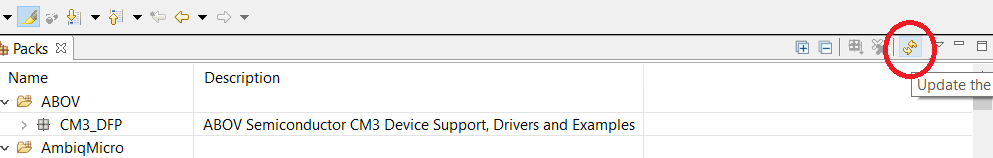


Figure : Refresh the packs list

After refreshing the list of available packs, select the boards tab (or the devices tab if the required board is not listed) and navigate to the STMicroelectronics category in the list and select the required board (STM32F429I Discovery in this case) and click on the STM32F4xx\_DFP as shown in Figure 10 and select Install a local copy of the selected package(s) as shown in Figure 11

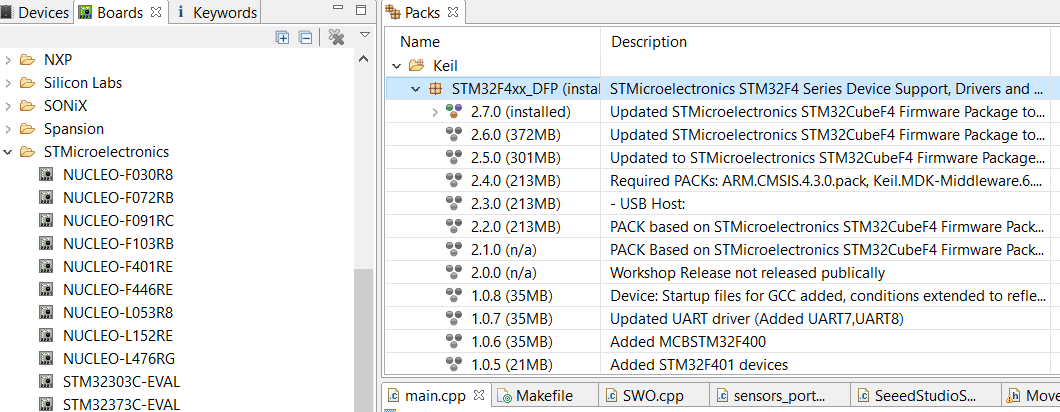


Figure : STM32F4xx\_DFP pack

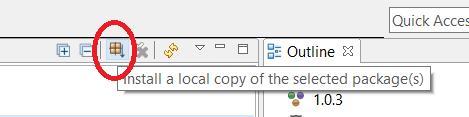


Figure : Installing a local copy of the required pack

The pack installations will require a working internet connection and some packs contain large files and therefore it is recommended to install the packs in a directory with sufficient space.

## Creating a simple test application

Open the Eclipse IDE and create a new C++ Project (Or a C project) from the file menu as shown in Figure 12.

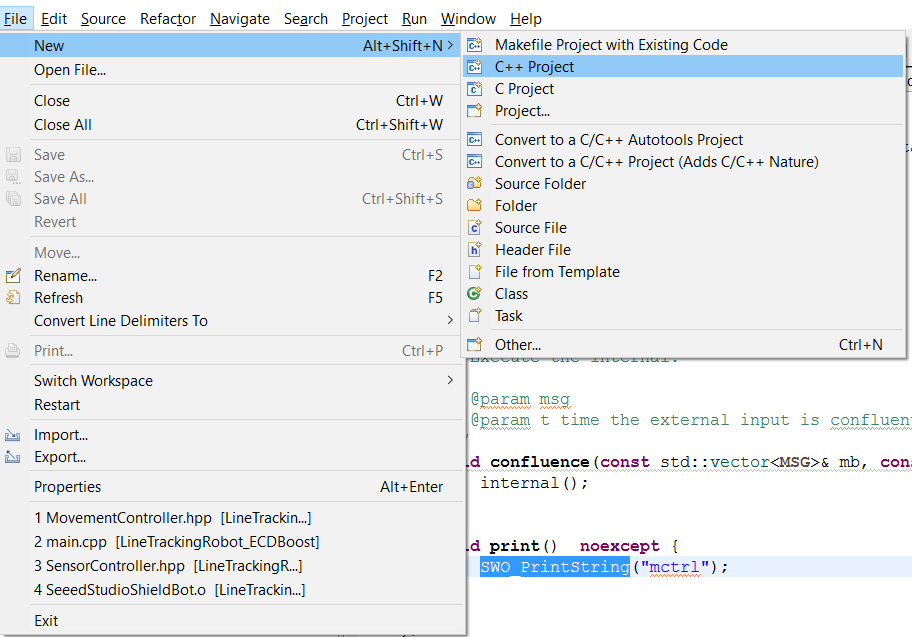


Figure : Creating a new project

On the next window select STM32F4xx C/C++ project under the list of available project types as shown in Figure 13

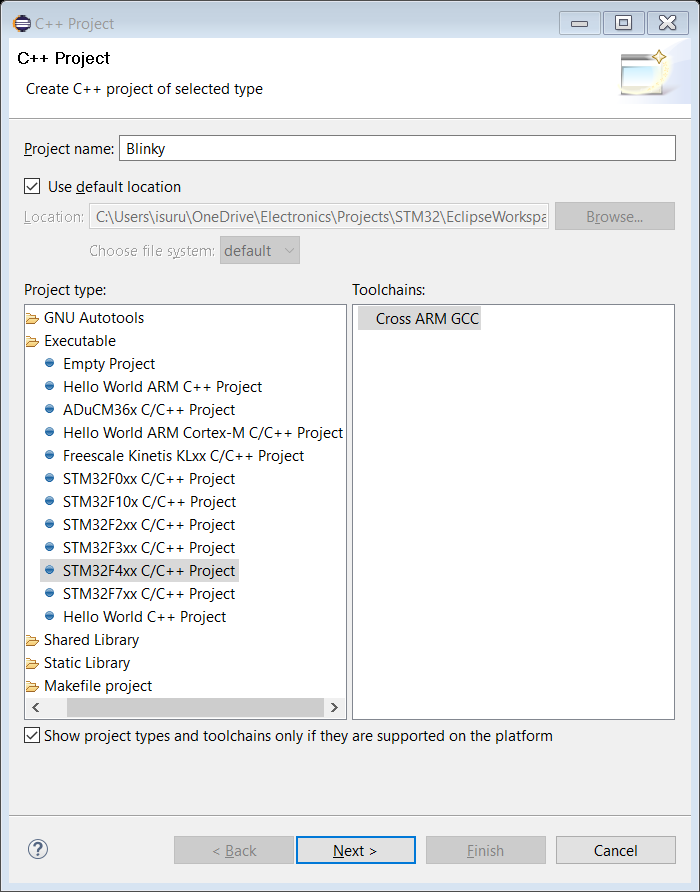


Figure : Creating a simple blinky project

On the next window select the correct microcontroller and fill in the correct flash size and clock speed. The flash size of the microcontroller can be found in the datasheet of the microcontroller used and the clock speed is the base clock frequency of the external clock source (or internal clock in some boards) installed in the development board. Refer to the development board user manual to find the correct values for these fields. The correct settings for the STM32F429 discovery board can be found in Figure 14

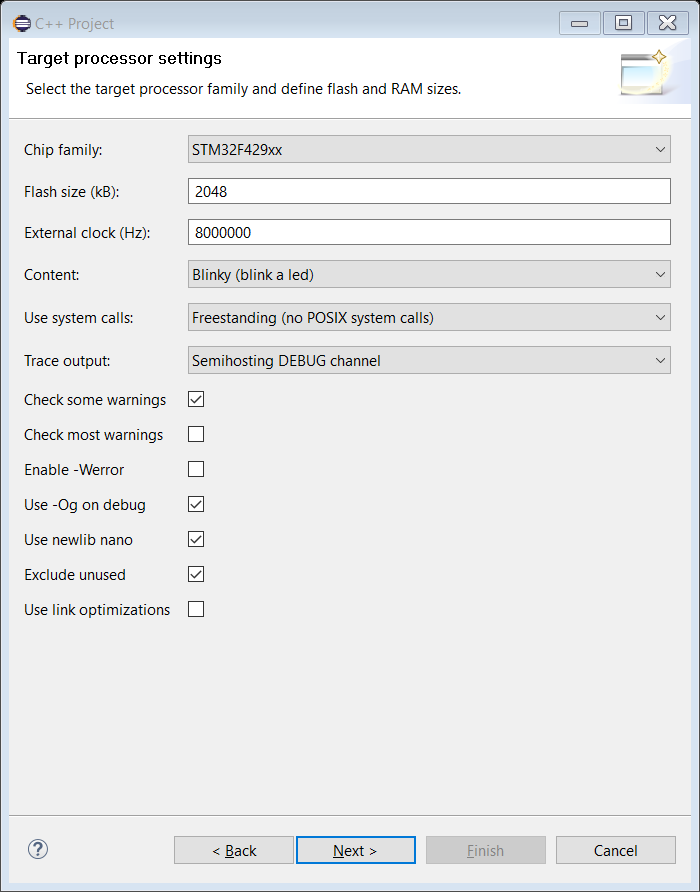


Figure : Project settings for the STM32F429 Discovery

In the proceeding pages, select the correct toolchain that was installed in 4.1 and click finish at the end to create a new project.

Right click on the newly created project in the project explorer and click Properties. In the new window that appears, under the category C/C++ Build, go to sub category “Settings” and in the devices tab select the correct microcontroller as shown in Figure 15

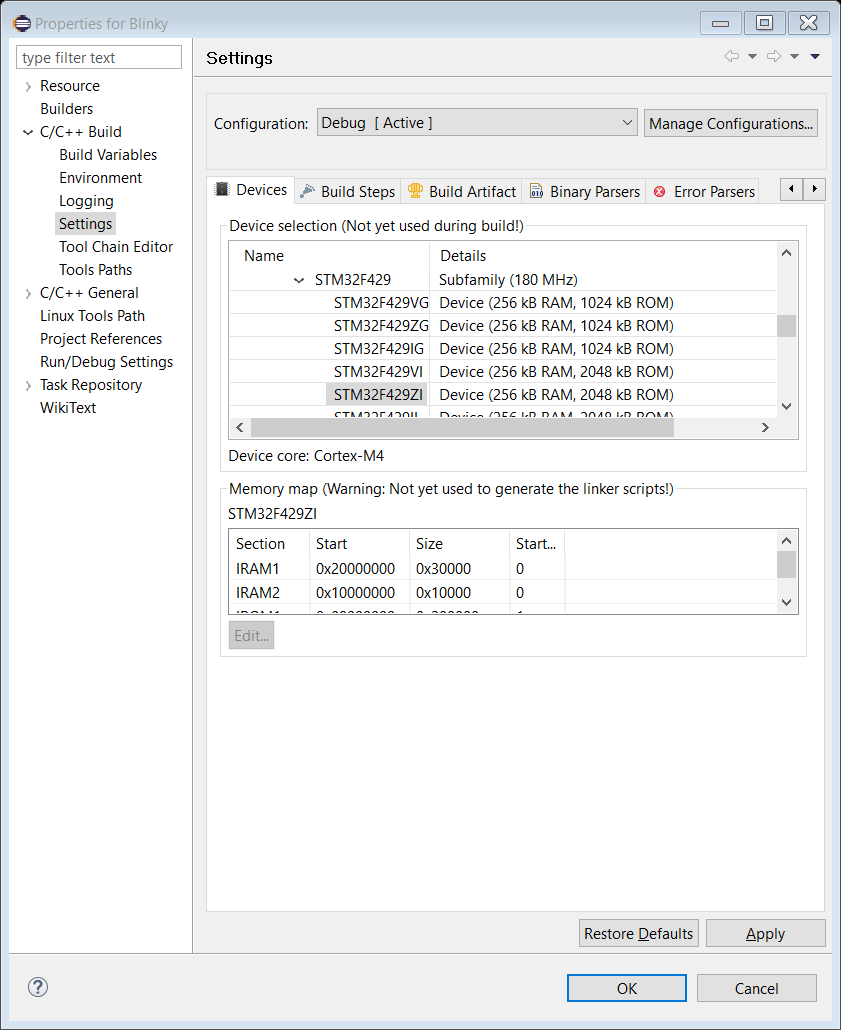


Figure : Selecting the microcontroller in the eclipse project properties window

After selecting the correct microcontroller, the project can be compiled by navigating to the menu “Project -> Build All”

## Emulating a simple application on QEMU

QEMU can be used to emulate the development board and run simple applications such as the blinky project created in section 4.7 to test if the toolchain and the compiler are installed and configured properly and to test simple application logic.

To create a run configuration for QEMU, right click on the project and select Run as -> Run configurations as shown in Figure 16

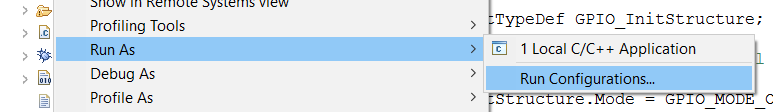


Figure : Run configuration bring-up

In the Run configurations window double click on the GDB QEMU Debugging section as shown in Figure 17 to create a new run configuration.

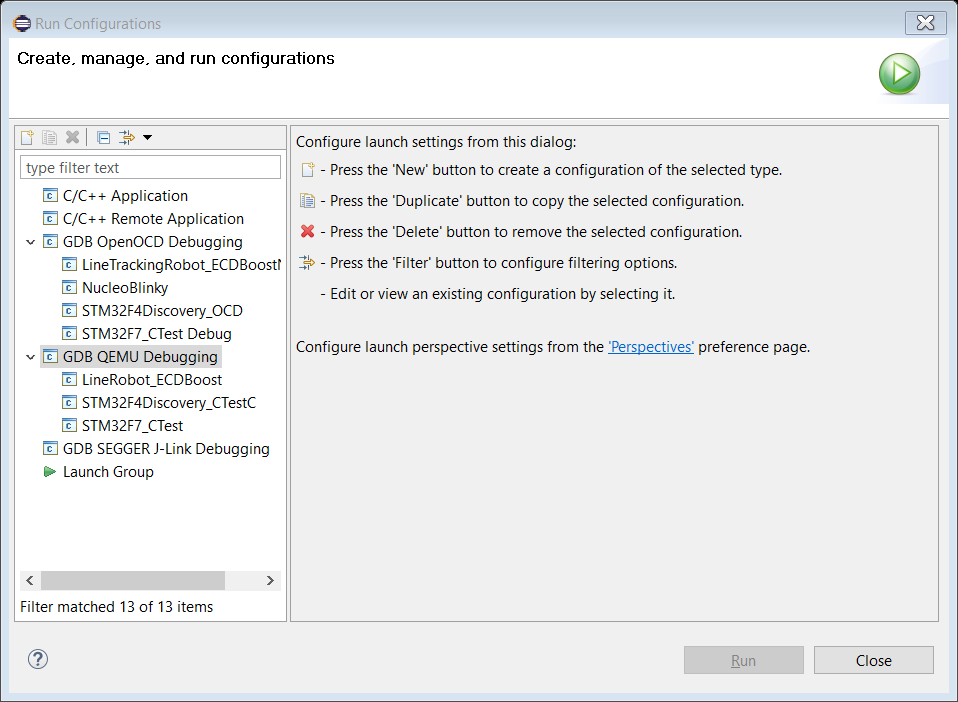


Figure : Creating a new run configuration

Click on the newly created run configuration and go to the Debugger tab. If the device was properly selected in section 4.7 , the settings for QEMU debugger will be automatically filled out. If they are not filled out, fill them according to Figure 18. The settings that needs to be checked are as follows:

Board name: STM32F429I-Discovery

Device name: STM32F429ZI

After selecting these, click run and a new window will appear and a STM32F429 Discovery board will be rendered along with its LED’s blinking according to the code.

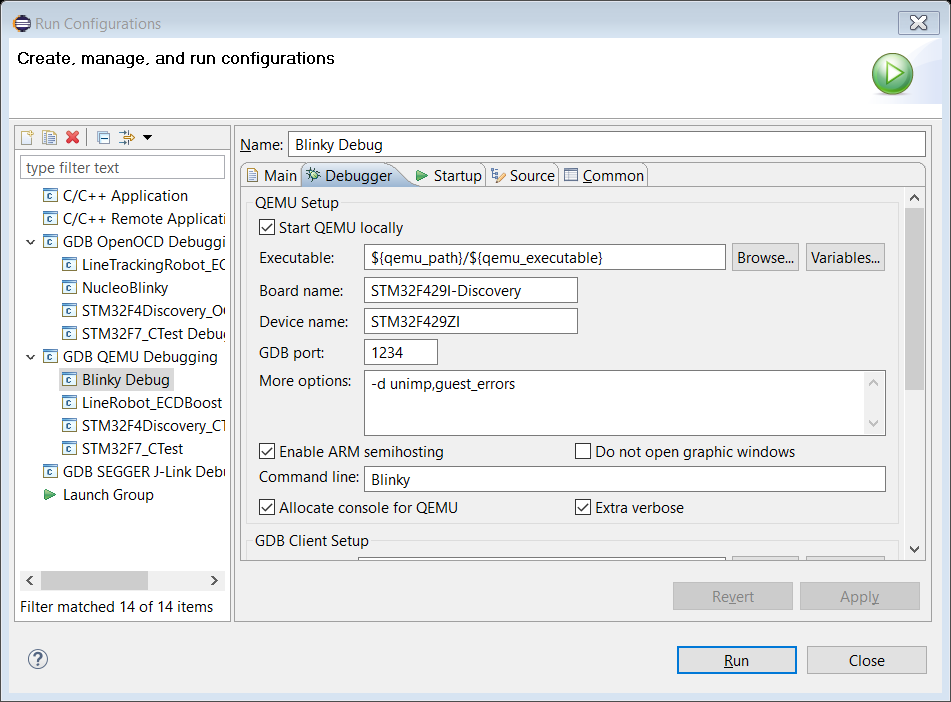


Figure : QEMU Debugger settings for the STM32F429 Discovery

## On Chip Debugging using OpenOCD

In the previous section, the compiled binary was run on an emulated environment with the QEMU debugger. But in real world applications, the testing will need to happen in the target environment itself and for this a chip programmer and an on chip debugger is required. OpenOCD is a free on chip debugger which supports the STMicroelectronics STLink programmer. To start debugging, right click on the project in the project explorer, and click Debug as -> Debug configurations as shown in Figure 19

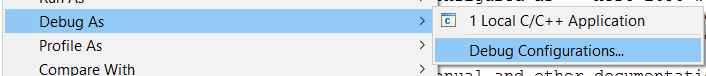


Figure : Debug configuration bring-up

In the Debug configurations window, double click on the GDB OpenOCD Debugging section as shown in Figure 20

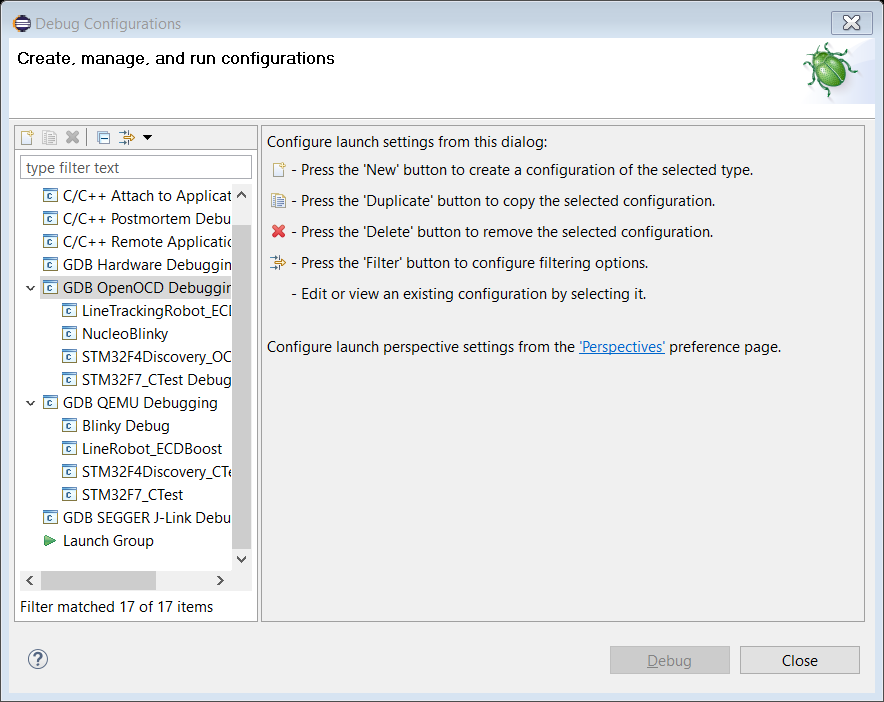


Figure : Creating a new debug configuration

In the new configuration created, go to the Debugger tab and check if configuration options are set up properly. In the Config options section, type “-f stm32f429discovery.cfg” to select the stm32f429 discovery board as shown in

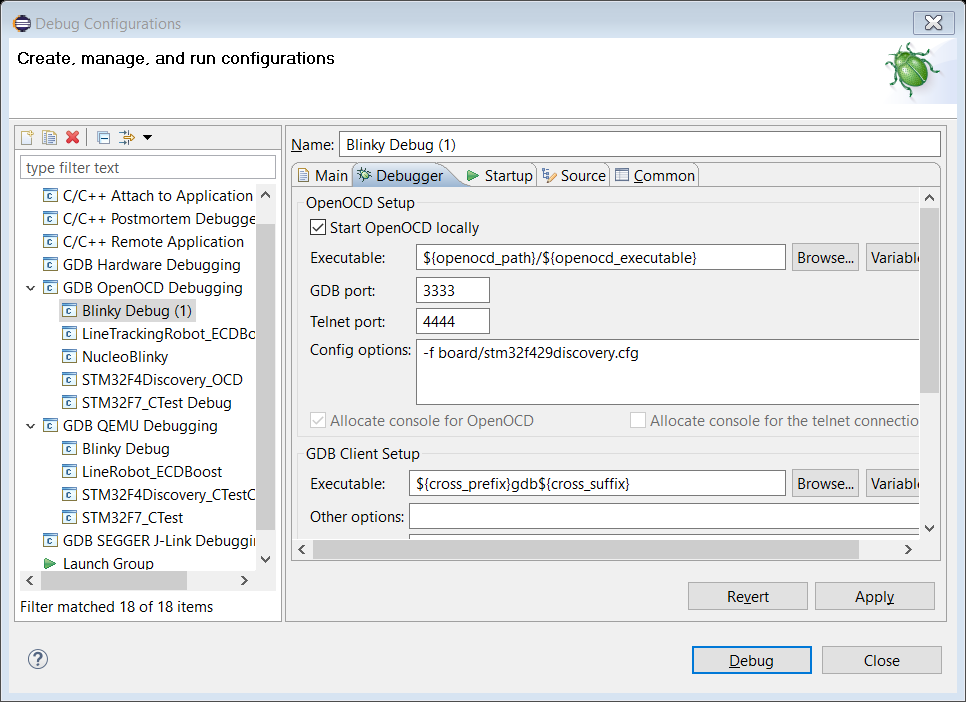


Figure : OpenOCD Debug settings for the STM32F429 Discovery

Configuration files for common microcontroller development boards are found in the OpenOCD installation folder which would look something like “C:\Program Files\GNU ARM Eclipse\OpenOCD\0.10.0-201601101000-dev\scripts\board” replace the .cfg file selected under Config options in Figure 21 for different boards. For example the setting for the STM32F4 Nucleo board is: “-f board/st\_nucleo\_f4.cfg”

After filling out the correct settings, the project is now ready to be debugged. Plug in the USB port of the development board and click Debug to start debugging the application. Open Debug perspective in Eclipse to better debug the application and the Debug session will run similar to a normal C++ debugging and options such as breakpoints, pause, stepping through code are available in the Eclipse debug perspective.

# Line tracking robot example ECDBoost application

## Importing the project into eclipse

The example application is a makefile project and thus can be imported as a makefile project. However, this approach doesn’t work in most systems and therefore the recommended method to import the project is described in this section.

Create a new project by navigating to File -> New -> C++ project. In the window that appears select “Empty project” under the category “Makefile project” as seen in Figure 22. Under toolchains, select Cross ARM GCC.

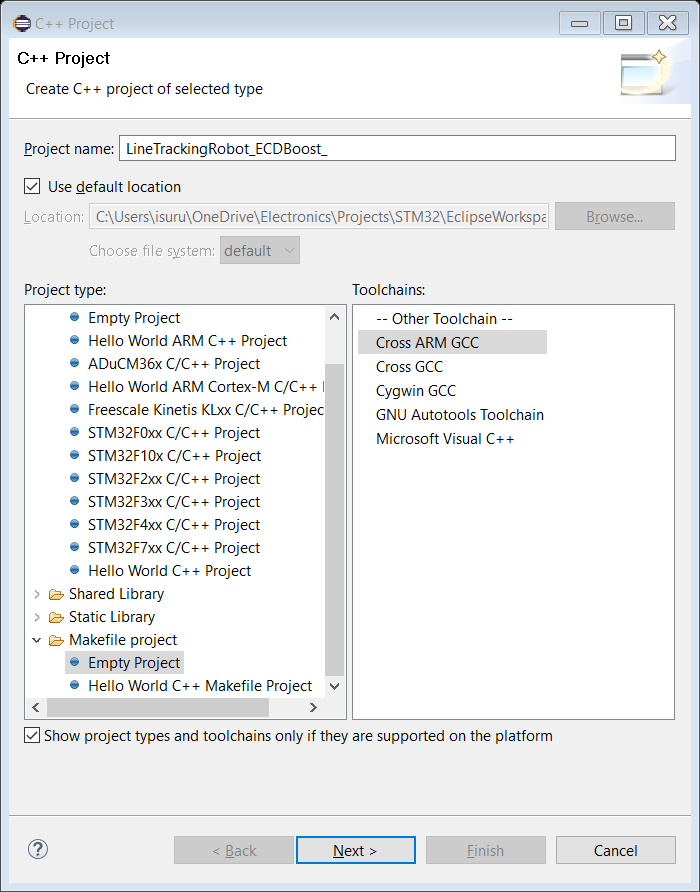


Figure : Settings for creating an empty makefile project

Follow through the next windows and click finish to create a project. After creating the project, navigate to the project location in the file explorer and replace everything inside the project folder with the contents from the provided “LineTrackingRobot\_ECDBoost” folder.

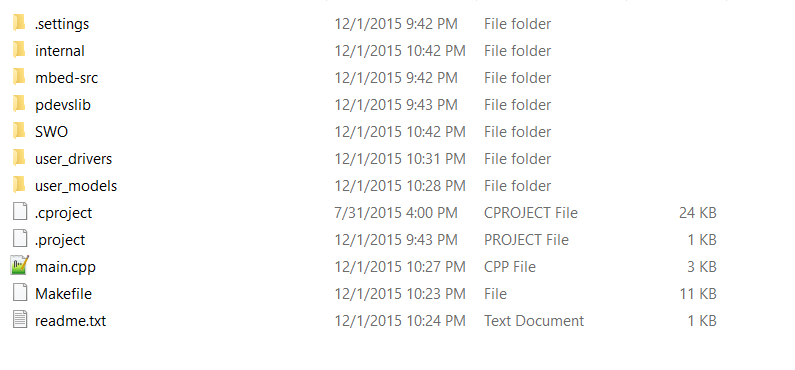


Figure : Folder structure of the LineTrackingRobot\_ECDBoost project

The project is not yet ready for compiling. The next steps will show how to compile the project.

The ECDBoost library uses the C++ Boost library and therefore needs to know where the Boost library is located. This can be done by editing the Makefile inside the LineTrackingRobot\_ECDBoost folder. Open the makefile using eclipse or any other text editor and under the section “INCLUDE\_PATHS” look for an absolute path which references to a location of a boost\_X\_XX\_X (X being the version number). This should look something like: "C:\Users\Daniella\Desktop\ARS\boost\_1\_55\_0". Unzip the provided boost\_1\_57\_0.rar (or newer version) to a location in your computer and replace the text mentioned above in the makefile with the location of the boost library in your computer. This would reference the boost library properly in order to compile the project.

Now the project is ready for compiling. Build the project and if it’s compiling, the project is ready for deployment. On some computers due to an ambiguity in the sleep function in the mbed source and the generic unix sleep function, the following compiler error might occur:

Figure : Atomic model definitions

// Atomic models definition

**printf**("Creating atomic models ... \n");

**auto** sctrl = make\_atomic\_ptr<SensorController<Time, Message>>();

**auto** mctrl = make\_atomic\_ptr<MovementController<Time, Message>>();

Conflicting declaration of C function 'void sleep()' sleep\_api.h /LineTrackingRobot\_ECDBoost/mbed-src/hal line 41 C/C++ Problem

If this error appears the easiest solution to it is to comment out the sleep function in the sleep\_api.h and sleep\_api.c files inside the mbed-src/hal folder. Even though this is not a recommended procedure, since the mbed sleep function is not used in the current robot code, this would not affect the operation of the robot.

Figure : Coupled model definition

**printf**("Creating Coupled models ... \n");

shared\_ptr<flattened\_coupled<Time, Message>> ControlUnit( **new** flattened\_coupled<Time, Message>{{sctrl,mctrl}, {sctrl}, {{sctrl,mctrl}}, {mctrl}});

## Project structure

Figure : Model hierarchy diagram of the line tracking robot

Figure : Simplified model hierarchy diagram of the line tracking robot

### Model definitions

As shown in Figure 27 and Figure 26 the line tracking robot model consists of one coupled model “ControlUnit” and two atomic models: “SensorController” and “MovementController” these are defined in the “main.cpp” file as shown in Figure 24 and Figure 25

The I/O ports of the models which are “START\_IN” “LIGHT\_IN”, “MOVEL\_OUT” and “MOVER\_OUT” are defined as follows inside the main.c file:

**printf**("Defining top I/O ports ... \n");

// Input ports

**auto** start = make\_port\_ptr<START\_IN<Time, Message>>();

**auto** light = make\_port\_ptr<LIGHT\_IN<Time, Message>>();

// Output ports

**auto** motorleft = make\_port\_ptr<MOVEL\_OUT<Time, Message>>();

**auto** motorright = make\_port\_ptr<MOVER\_OUT<Time, Message>>();

Next a “runner” is created to coordinate the models and this is created as:

**printf**("Preparing runner \n");

Time initial\_time{Time::*currentTime*()};

erunner<Time, Message> root{ControlUnit, {{start,sctrl},{light,sctrl}} , {{motorleft,mctrl},{motorright,mctrl}} };

Time end\_time{Time(0,30,0,0)};

Finally, the simulation is started by calling: end\_time = root.runUntil(end\_time); with the end\_time specified to stop the simulation.

The internal, external and confluence functions for the models are defined in the MovementController.hpp and SensorController.hpp files inside the folder /user\_models.

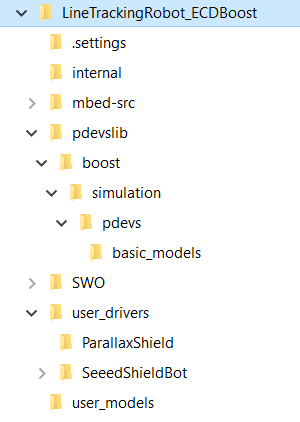


Figure : Expanded folder structure of the LineTrackingRobot\_ECDBoost application

### Hardware drivers and changing between hardware.

The hardware drivers are found inside the /user\_drivers folder and currently it contains drivers for the Parralax robot with the STM32F429 Discovery board and the SeedShieldBot with the STM32F411 Nucleo board. Switching between hardware is achieved by changing an include file in the main.cpp file. To use the Parallax robot with the STM32F429 microcontroller, change the following include:

#include "linerobot\_driver.hpp"

To,

#include "linerobot\_driver\_Parallax.hpp"

And vice versa. Intuitively, the linerobot\_driver.hpp file corresponds to the SeedShieldBot [3] and the linerobot\_driver\_Parallax.hpp file corresponds to the parallax robot. The SeeedShieldBot includes 5 line sensors and two motors as shown in Figure 29

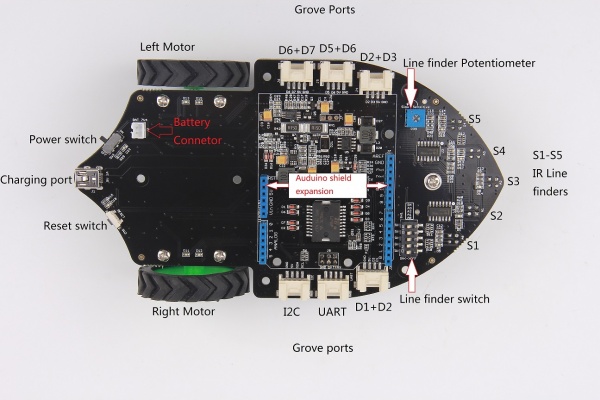


Figure : Seeed Shield Robot components

The drivers for the robot hardware can be found in the /user\_drivers folder and it contains separate folders for the parallax robot and the Seeed Shield Bot as shown in Figure 28.

The pdevslib folder contains the .hpp files referenced by the files defining the models and their functions and examining these files along with the example code allows to understand the syntax used to define new models.

## Debugging the robot

After compiling the project, the steps in section 4.9 can be followed to debug the robot with OpenOCD or by programming the .elf file found inside the /binaries folder using the STMicroelectronics STLink utility as shown in section 3.2. Text output from the robot can be viewed using two methods.

1. Using a serial port terminal
2. Using the SWO output built in to the STLink programmer.

### Using a serial port terminal to view output.

In the Line tracking robot example code, the printf() function is redirected to the serial port and therefore any string printed using the printf() function can be viewed through the serial port.

Any serial port terminal can be used for viewing text output from the robot and the recommended method is to use Tera Term [15] in Microsoft Windows based host machines. Tera term can be downloaded from [15].

After installing Tera Term, open the application and select the serial port that corresponds to the hardware board that is being used as shown in Figure 30

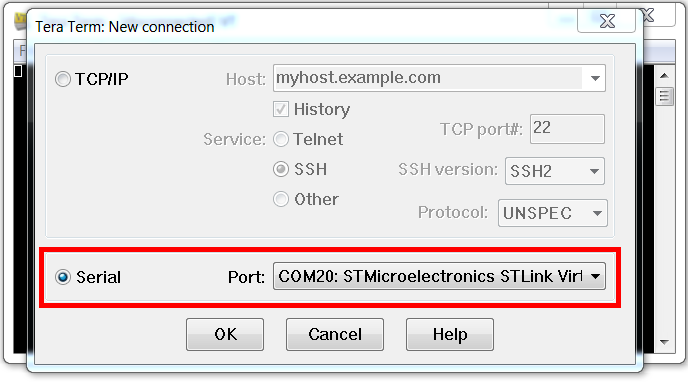


Figure : Tera Term Serial port selection

After selecting the serial port, navigate to “Setup->Terminal” and enter the settings shown in Figure 31

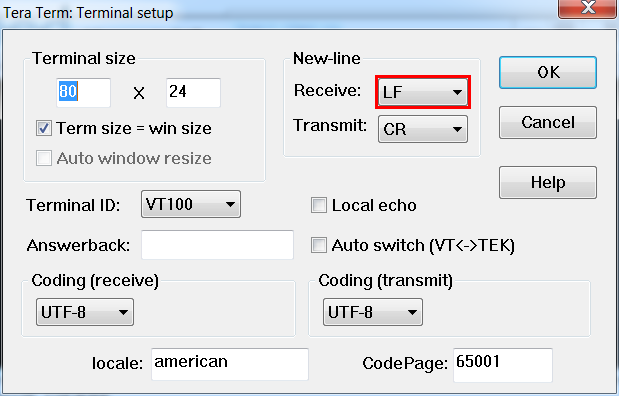


Figure : Tera Term terminal setup

Next navigate to “Setup->serial port” and enter the following settings as shown in Figure 32

Baud rate: 9600

Data: 8 bit

Parity: none

Stop: 1 bit

Flow control: none

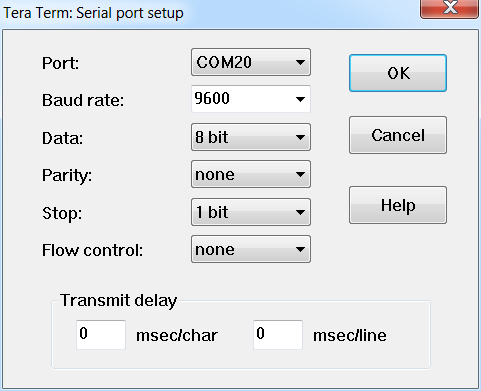


Figure : Serial port setup

Now the output from the terminal will appear on the serial port.

### SWO output viewer.

In the Line tracking robot example, the output from the function “SWO\_PrintString();” can be viewed through the STLink Utility’s SWO viewer. To bring up the SWO viewer open the STLink Utility and click on the SWO Viewer button as shown in Figure 3.

In the proceeding window enter the system clock frequency of the microcontroller as shown in Figure 33

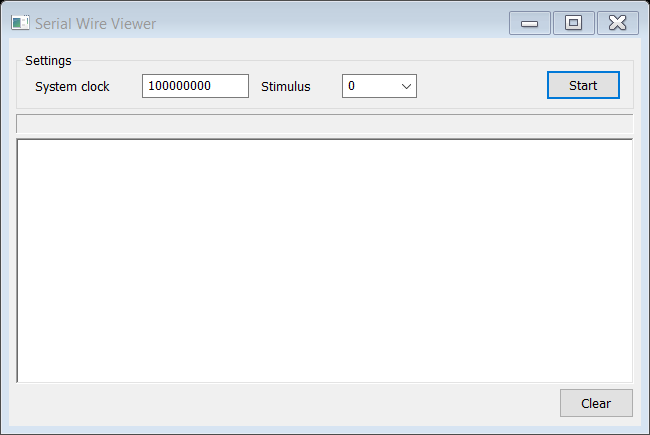


Figure : SWO Viewer settings

The System clock setting in this window should be the clock frequency of the microcontroller after the PLL Multiplexer stage and not the frequency of the crystal oscillator found in the development board. The setting for the STM32F411 Nucleo board is 100000000 as the microcontroller is running at 100MHz. The correct setting for this can be found by browsing the relevant Data sheet of the microcontroller and, by default the microcontrollers are set to run at the maximum frequency by initialization code. Therefore, if the clock frequency of the microcontroller was not changed by changing the code, trying out with the maximum possible clock speed of the particular microcontroller will most definitely work. The clock speed of the microcontroller with the STM32F429 Discovery board is 180MHz and therefore the SWO clock speed setting should be 180000000 for the STM32F429 Discovery board.

# ACKNOWLEDGMENTS

I would like to thank Professor Gabriel Wainer of the Department of Systems and Computer Engineering at Carleton University and Daniella Niyonkuru for their contributions in developing this guideline for setting up ECDBoost projects.

# References

|  |  |
| --- | --- |
| [1] | STMicroelectronics, "32F429IDISCOVERY Discovery kit with STM32F429ZI MCU," STMicroelectronics, [Online]. Available: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF259090. [Accessed 16 04 2016]. |
| [2] | Parallax, "Parallax robot shield kit," Parallax, [Online]. Available: https://www.parallax.com/product/130-35000. [Accessed 16 04 2016]. |
| [3] | Seeedstudio, "Shield Bot," Seeed Studio, [Online]. Available: http://www.seeedstudio.com/wiki/Shield\_Bot\_V1.1. [Accessed 16 04 2016]. |
| [4] | STMicroelectronics, "NUCLEO-F411RE," STMicroelectronics, [Online]. Available: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1847/PF260320. [Accessed 16 04 2016]. |
| [5] | ARM, "mbed online compiler," ARM mbed, [Online]. Available: https://developer.mbed.org/. [Accessed 16 04 2016]. |
| [6] | ARM, "KEIL MDK," ARMKeil, [Online]. Available: http://www.keil.com/arm/mdk.asp. [Accessed 17 04 2016]. |
| [7] | ARMmbed, "STM32F429 discovery with mbed," ARMmbed, [Online]. Available: https://developer.mbed.org/platforms/ST-Discovery-F429ZI/. [Accessed 10 04 2016]. |
| [8] | "STSW-LINK004 STM32 ST-LINK utility," STMicroelectronics, [Online]. Available: http://www.st.com/web/en/catalog/tools/PF258168. [Accessed 10 04 2016]. |
| [9] | "STSW-LINK009 ST-Link, ST-Link/V2, ST-Link/V2-1 USB driver signed for XP, Windows7, Windows8," [Online]. Available: http://www.st.com/web/en/catalog/tools/PF260219. [Accessed 10 04 2016]. |
| [10] | "GNU ARM Eclipse Installation instructions," [Online]. Available: http://gnuarmeclipse.github.io/install/. [Accessed 12 04 2016]. |
| [11] | "GCC ARM Embedded toolchain download," [Online]. Available: https://launchpad.net/gcc-arm-embedded/+download. [Accessed 12 04 2016]. |
| [12] | "Windows build tools download," [Online]. Available: https://sourceforge.net/projects/gnuarmeclipse/files/Build%20Tools/. [Accessed 12 04 2016]. |
| [13] | "OpenOCD Download," [Online]. Available: https://github.com/gnuarmeclipse/openocd/releases. [Accessed 13 04 2016]. |
| [14] | "QEMU Installation," [Online]. Available: http://gnuarmeclipse.github.io/qemu/install/. [Accessed 12 04 2016]. |
| [15] | "Tera term download," [Online]. Available: https://ttssh2.osdn.jp/index.html.en. |