Practical 2B

Introduction to Microcontrollers - GPIO outputs

Before the practical ensure that you complete the following:

Read and revise:

Chapter 6 - C for microcontrollers (pp. 134 to 144) and **Chapter 7 - General purpose input output** (pp. 145-161) and in the **EEE2046F/EEE2050F course notes**. Please note that the practicals for this course contain a large self-study component and therefore be prepared before the start of the practical.

Bring the following to the lab session :

- 1 x UCT STM32F0 development board
- A PC/laptop running Eclipse with the ARM toolchain setup. All lab PC's in the departmental Red Lab have this preinstalled and configured.
- A USB-A to USB-B cable

2B.1 Introduction

The aim of this practical is to introduce you to basic C programming for microcontrollers. In this practical you are introduced to the Eclipse Integrated Development Environment (IDE), which you will use to program and debug your UCT STM32 development board throughout the semester. You will then investigate the general purpose output (GPIO) capability of your STMicroelectronics STM32F0 Microcontroller. You will also learn how to read the microcontroller reference manuals to identify which registers are needed to control the system's operation.

This practical contains both coding and a report submission and may take longer to complete than the 2 hour practical session. Please ensure you copy and paste your code from each question into a document and that it is well formatted so that it can be easily marked after the practical. Submit both your document and **.c** file on VULA by the due date. You are also welcome to complete the coding during free sessions in the Red Laboratory.

Please name your practical report as follows:

Prac2B-STUDENTNUMBER1

Call a teaching assistant or tutor if you need assistance at any stage during the practical session.

2B.2 Eclipse Integrated Development Environment (IDE)

Eclipse is an open source *Integrated Development Environment* (IDE) used for editing, compiling, debugging and programming. It can be used to produce code for a wide variety of platforms and can work with a number of programming languages. We will be using it to produce C code to program our STM32F051 microcontroller.

In this course, we will use Eclipse with an Embedded ARM toolchain to develop Embedded C applications for our STM32F0 microcontroller. All PC's in the main EBE and Electrical Engineering Computer Labs have a preconfigured Eclipse (Neon3) and ARM toolchain installed on them. If you would like to configure your home PC/laptop, follow one of the guides on VULA.

In this practical we will describe how to create a project, import both a header and source file, and then how to compile, debug and run the program on our target microcontroller.

1. Before opening Eclipse, download the following files from **VULA** → **EEE2046F** → **Resources** → **Practicals** → **Code**:

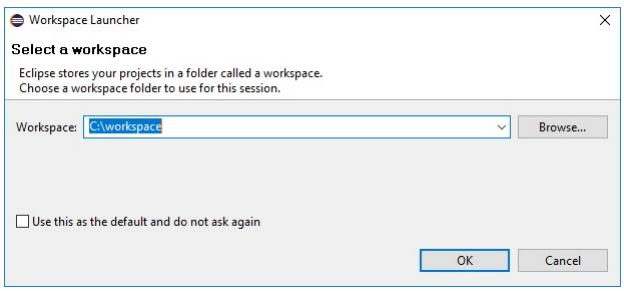
   - **lcd_stm32f0.c**
   - **lcd_stm32f0.h**
   - **main.c**

   and save them to the Desktop.

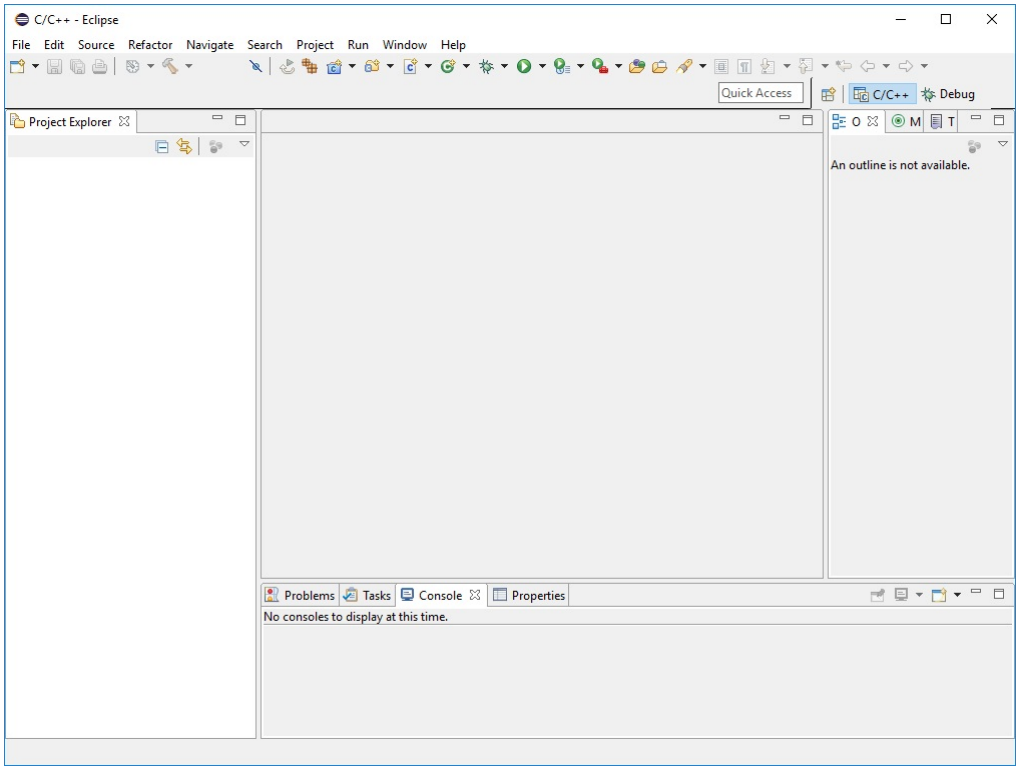
2. Connect your UCT development board to the PC using the USB cable.

3. Open Eclipse on your computer.

4. Make sure the workspace is set to **C:\ workspace** and click **OK**.



5. The C/C++ Eclipse IDE should now open and look as follows:

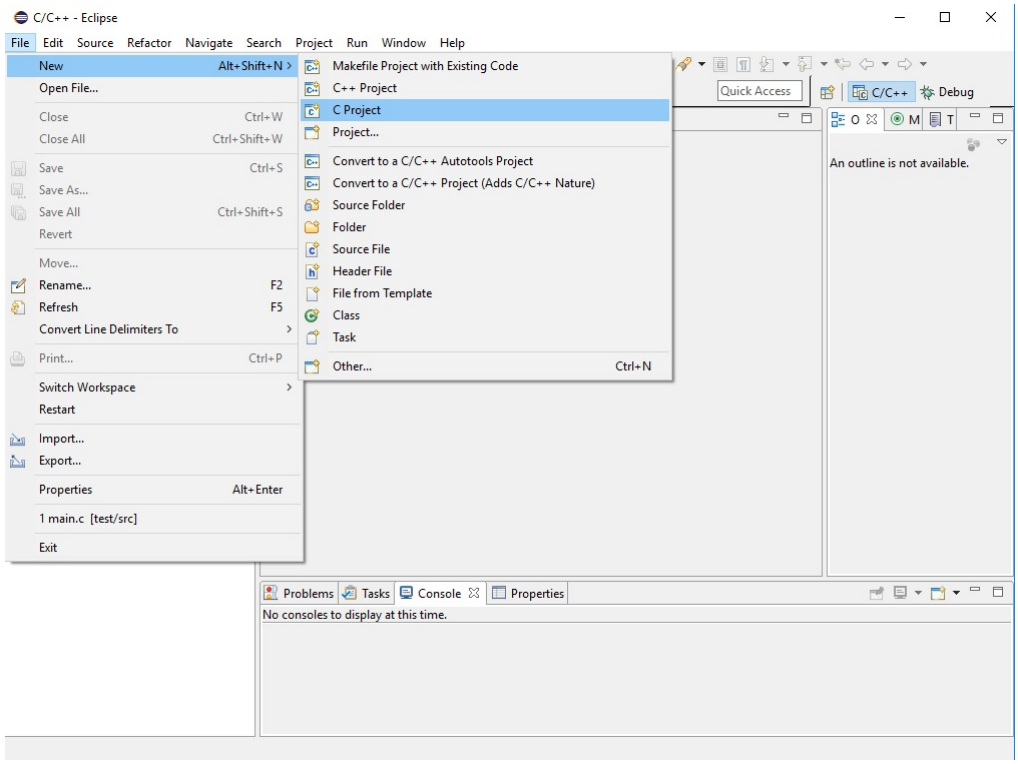


6. Create a new project

File → New → C Project (Alt+Shift+N).

name it as follows:

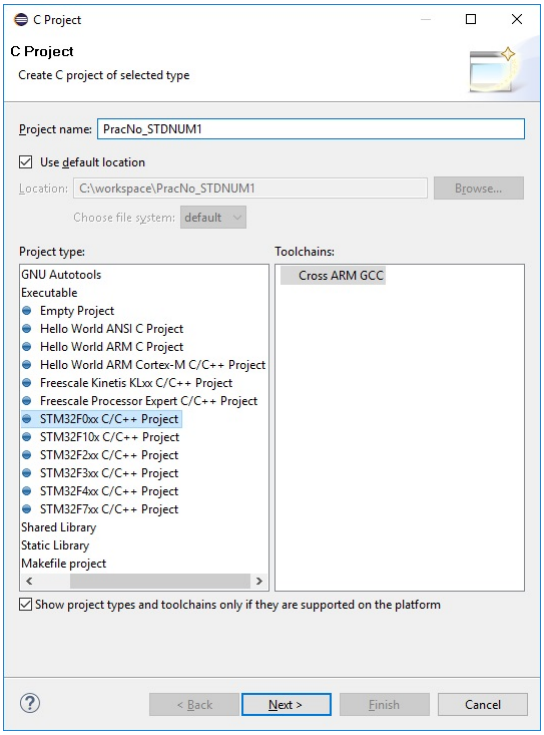
Prac2B-STUDENTNUMBER1



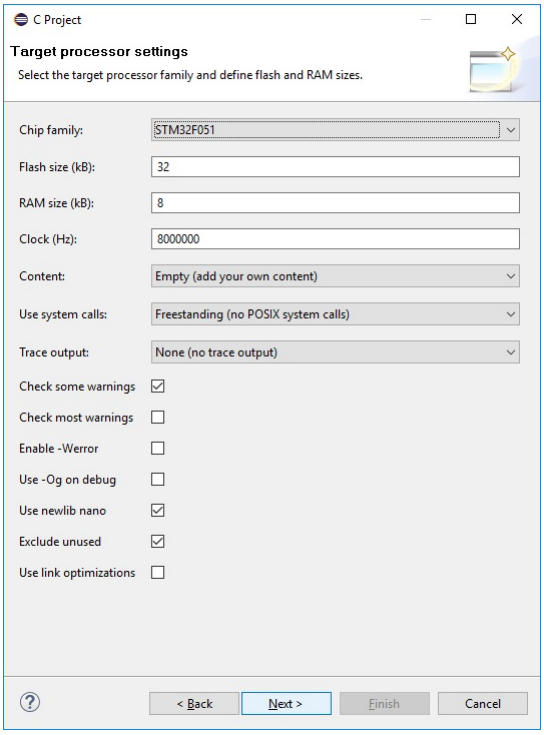
7. Name the project:

Prac2B-STUDENTNUMBER1

and select the **STM32F0xx C/C++ Project** under the Project type window. Ensure that the **Cross ARM GCC** toolchain is available[1].

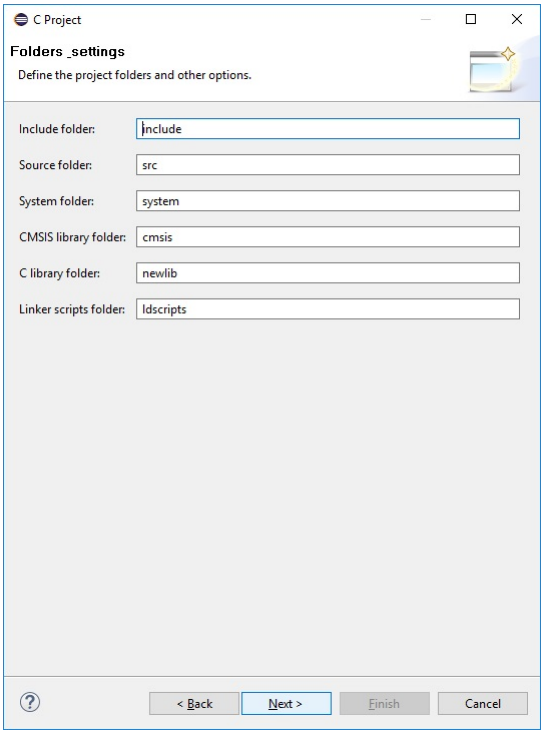


8. Ensure that the **Target processor settings** are configured as below. Click **Next**.

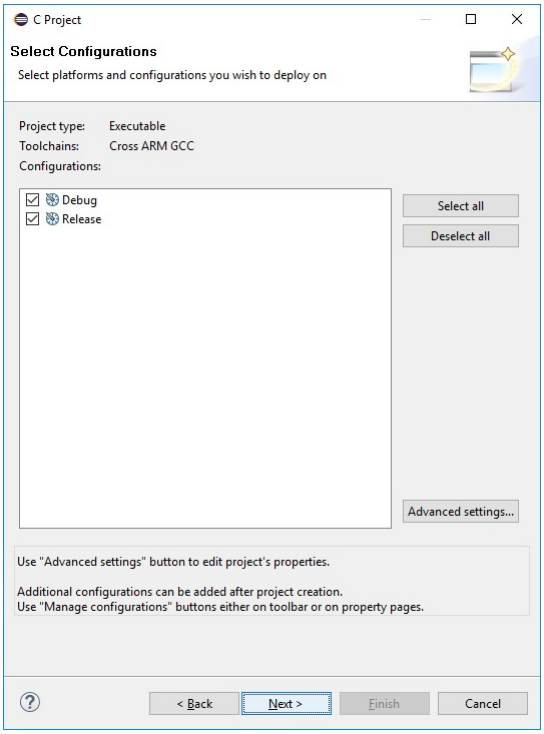


9. Ensure that the **Folder settings** are configured as below. Click **Next**.

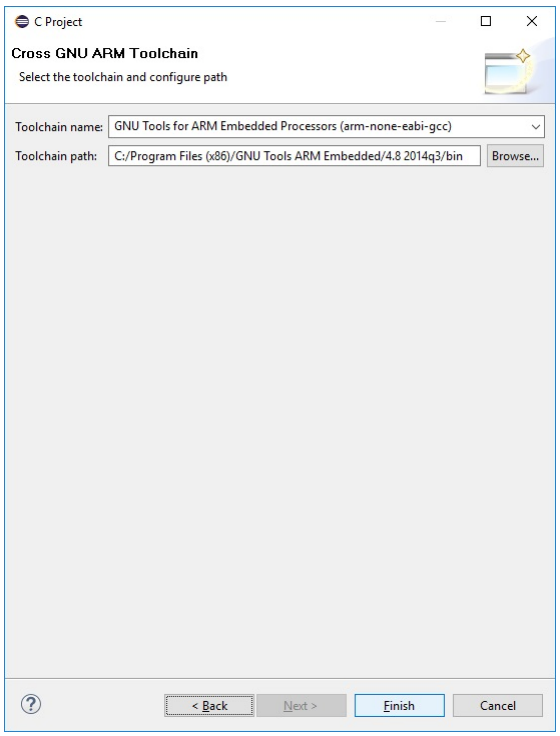
[1]Note that you need to install and configure the toolchain if running Eclipse on your own machine



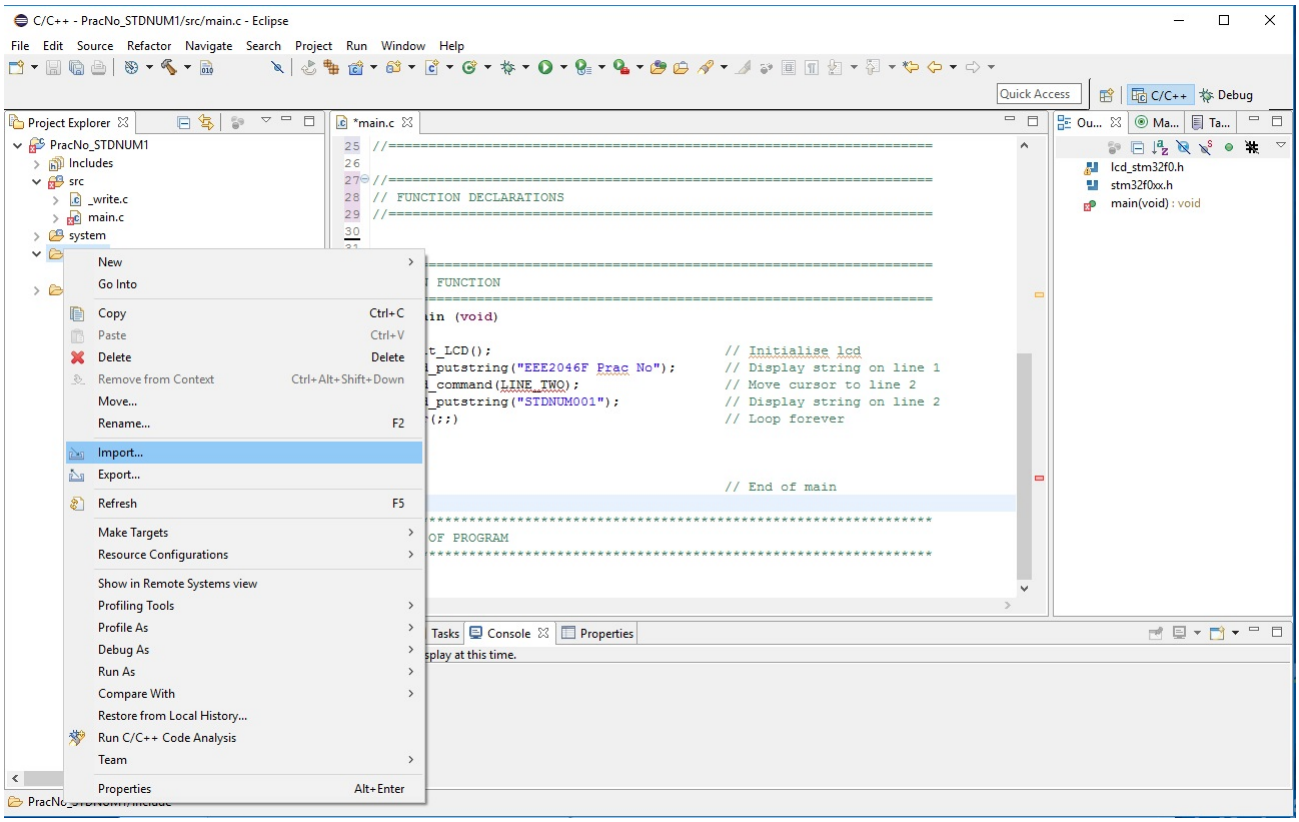
10. Ensure that the **Select Configurations** are configured as below. Click **Next**.



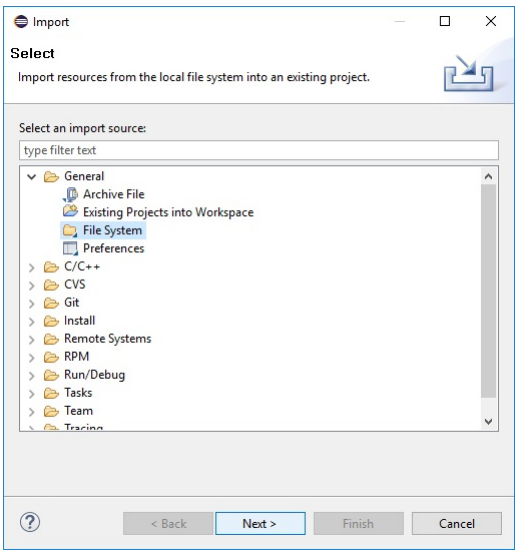
11. Ensure that the Cross GNU ARM Toolchain is configured as below. Click **Finish**.



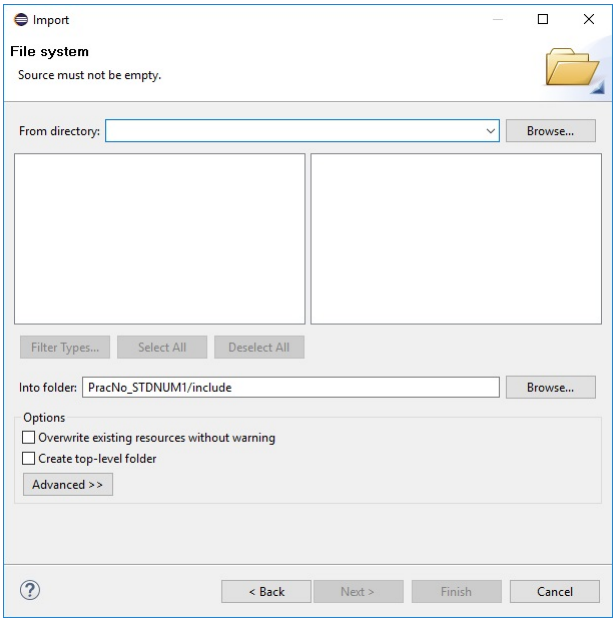
12. You have now created a new C/C++ project. A project can have a number of source files (src) and included header files (include). We are now going to include the files we downloaded earlier which provide us with default functions to write to the LCD screen. Right-click on the include folder. Click **Import...**.



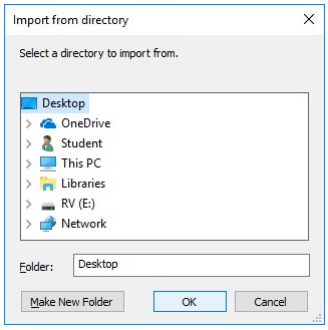
13. Click on **File System**.



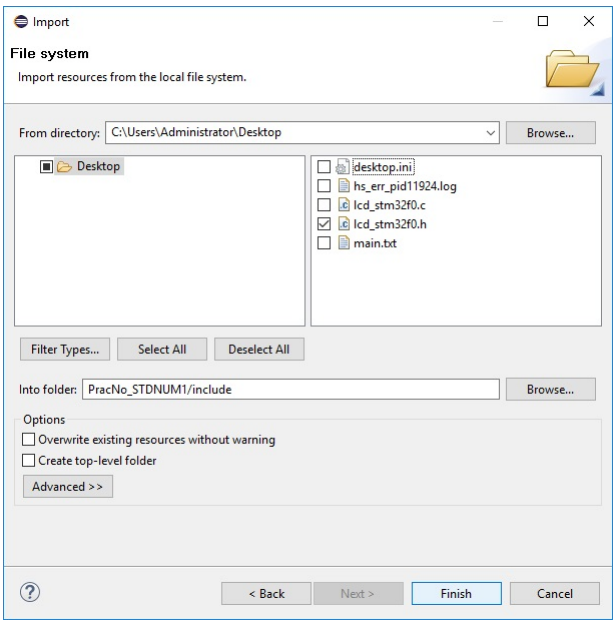
14. In the **From directory** box, click **Browse...**.



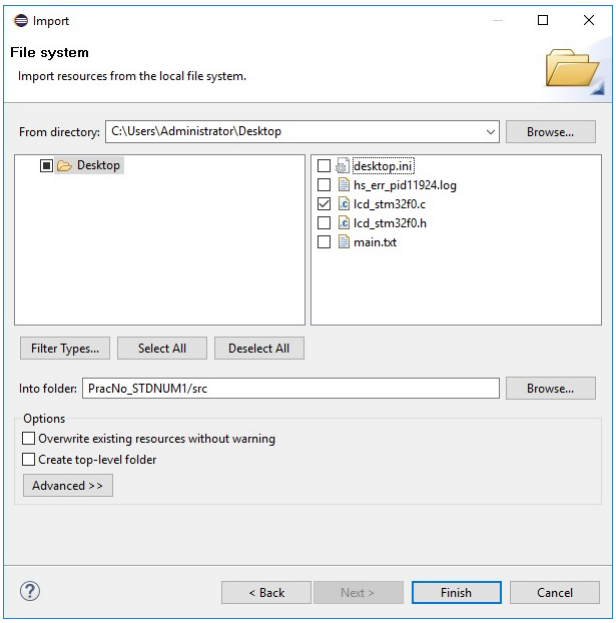
15. Select **Desktop**.



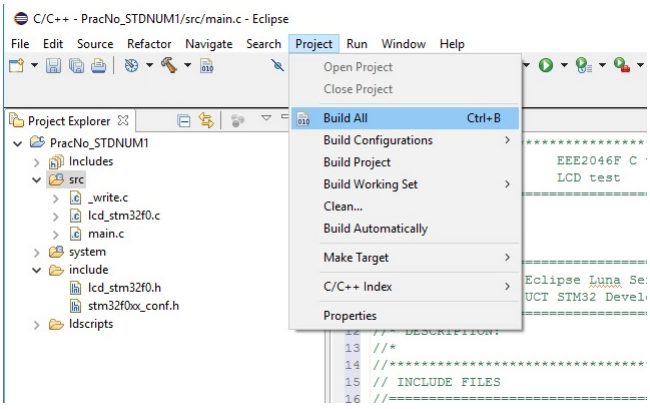
16. You should now see a list of the (**.c** and (**.h** files)). Select the **lcd_stm32f0.h** (see Appendix B.4 for details of the library). Click **Finish**.



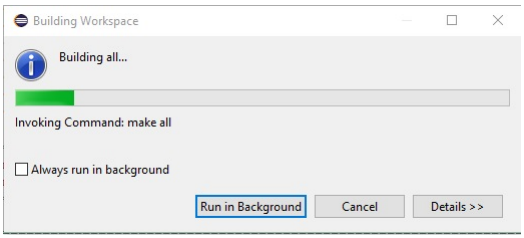
17. Go to your desktop. Go back to Eclipse and delete the **main.c** file under the src folder. Now right click on the **src** folder. Repeat steps 12 to 14. Select the **lcd_stm32f0.c and main.c** . Click **Finish**.



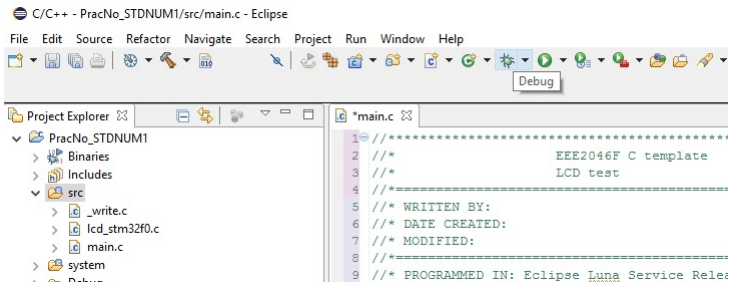
18. We are now going to **Build** our project. Click on **Project** → **Build All (Ctrl+B)**.



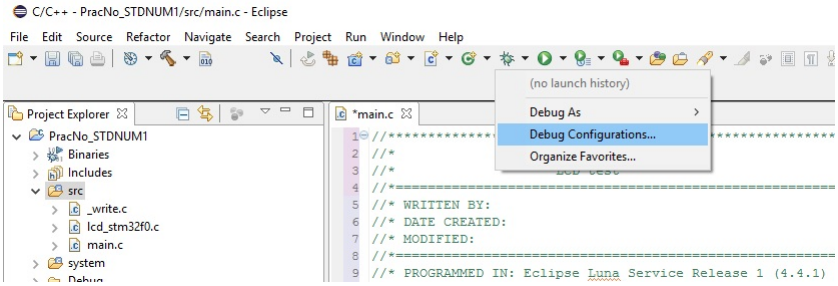
19. You should see a **Building all...** status bar box.



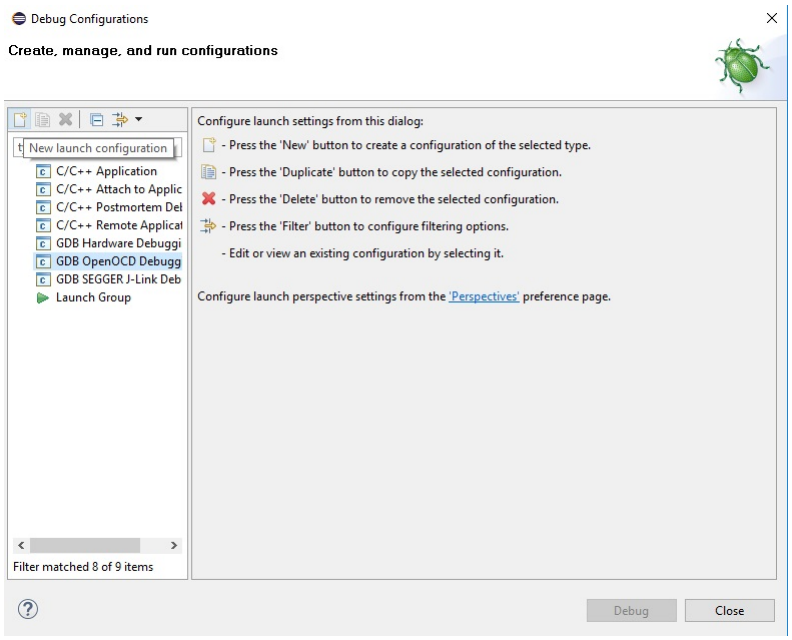
20. If your project compiled with no errors, then we now need to debug it and program our target processor.



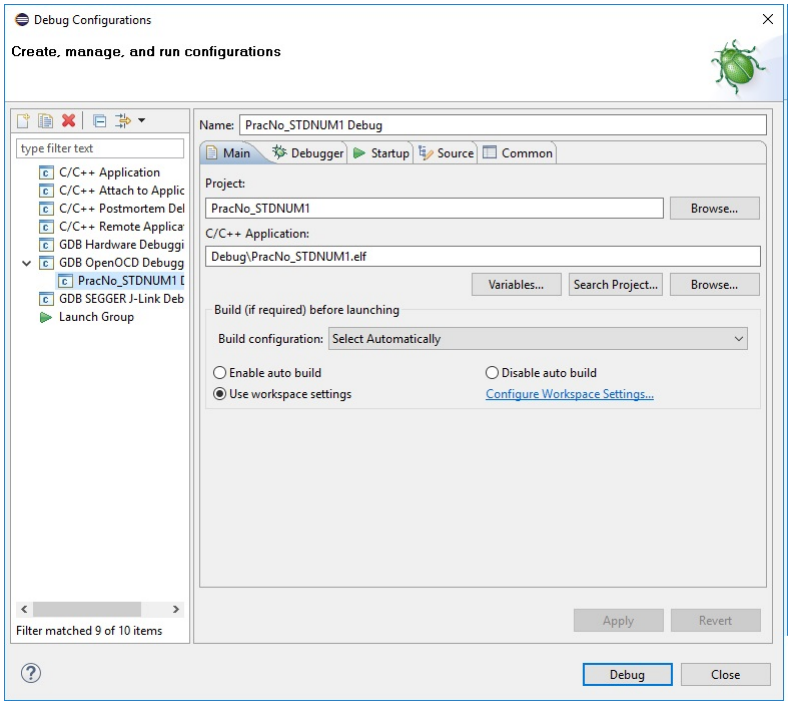
21. Plug in your **UCT STM32F0 Development Board** using a USB type A to B cable. You should see the power LED light up on your board. Right-click on **Debug**. Select **Debug Configurations...**



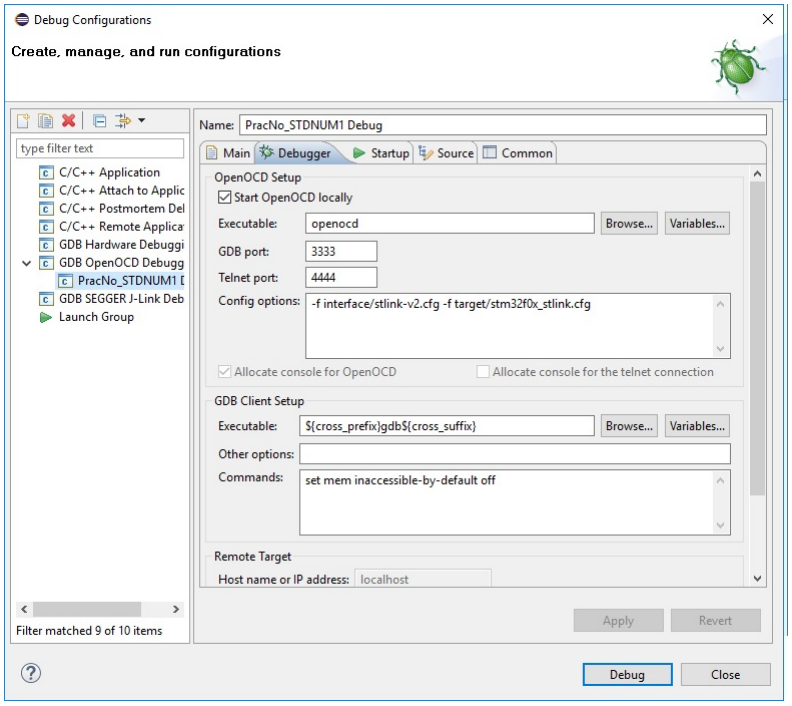
22. We now need to create a new Debug configuration. Select the **GDB OpenOCD Debugger**. Click on the **New** button.



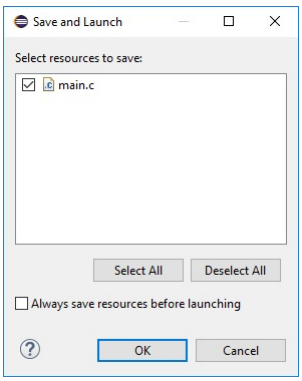
23. You should now see a new debug configuration as follows. Ensure that your **.elf** file is available under the C/C++ Application.



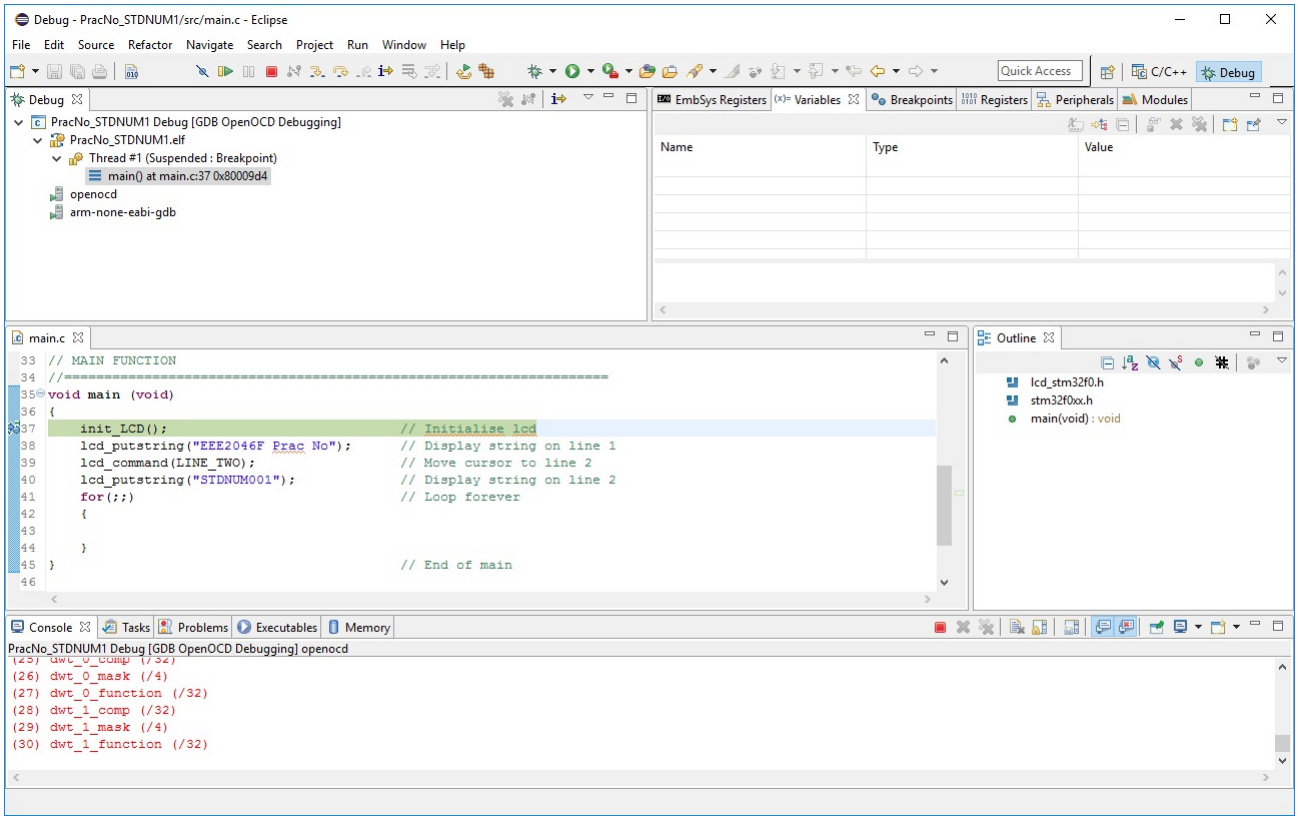
24. Click on the **Debugger** tab and ensure that is configured as below. Click **Debug**.



25. Click **OK**.



26. The Debug window should now open and the connection with the ST-LINKv2 should be enabled (if working properly). Press **F8**.



NOTE: Clear, concise and well commented coding is encouraged. Please ensure that you explain all steps in your code and follow good coding practice.

2B.3 GPIO

In this practical you will learn how to configure and use the *General Purpose Input/Output* ports (GPIO) of your STMicroelectronics STM32F051C6 microcontroller. Microcontrollers contain a number of *General Purpose Input Output* (GPIO) pins which act as a generic physical interface or connection to external devices. The behaviour and the functionality of these pins can be modified by the programmer in firmware and must be defined before use.

Input and *output* (I/O) signals can be either digital or analogue and this depends on the type of external device which is connected to the pin.

Microcontroller GPIO pins are grouped together in a parallel interface known as a *port*. These ports are normally labelled with a letter such as port A, port B etc. and often contain 8 to 16 pins per port.

All microcontrollers contain GPIO ports, however their physical implementation and functionality varies between manufacturers and families. The 48-pin STM32F051C6 microcontroller contains 39 GPIO pins, a number of which are multiplexed with internal peripheral modules built into the microcontroller chip. The pin names and numbers can be seen in the figure 2B.1.

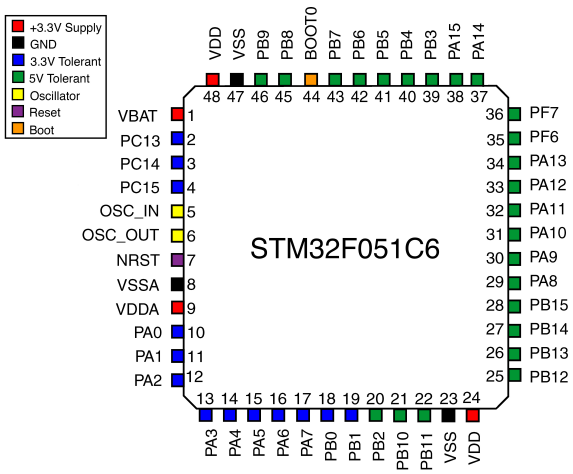


Figure 2B.1: The pinouts of the STM32F051C6 microcontroller. The colour of the pin represents the pin type and I/O structure.

Each GPIO port's operation is controlled by a number of GPIO peripheral registers as well as a *Reset and Clock Control* peripheral register (**`AHBENR`**) which define the following:

- The AHB bus clock connections to the GPIO port.
- The register's mode (Digital input, digital output, alternative function such as SPI, I^2C etc. or analogue mode)
- GPIO port's output type (Output enabled as push/pull or enabled as open-drain)
- GPIO port's output switching speed (low, medium or high)
- GPIO port's internal pull up or pull down resistor enable
- Used to read GPIO port input data register
- GPIO port output data register
- GPIO port set or reset the output data register
- Registers to select an assigned alternative function for a port pin

The internal structure of a single pin in a port can be seen in figure 2B.2.

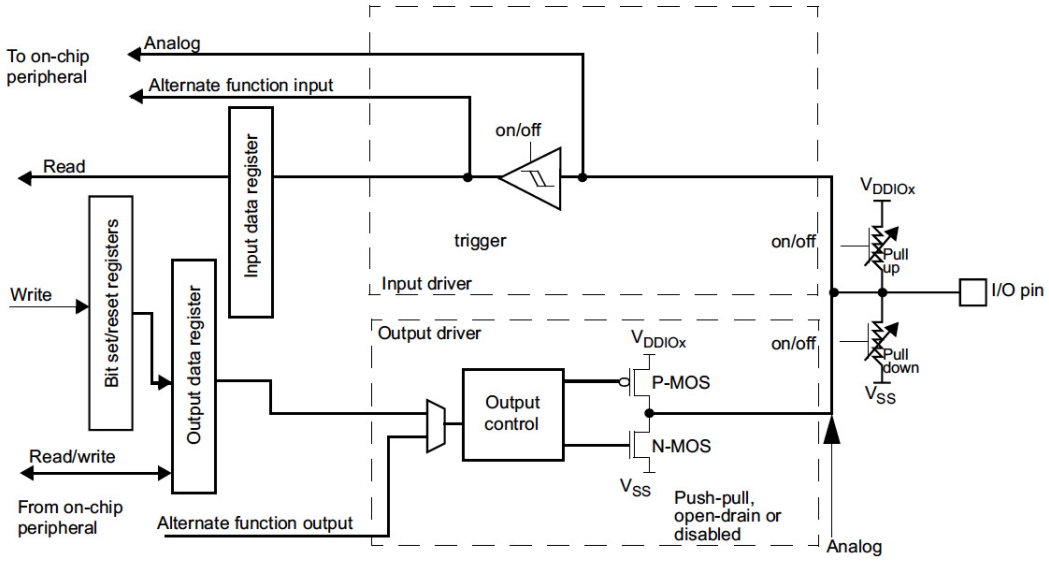


Figure 2B.2: The internal structure of a GPIO pin in the STM32F0 microcontroller. (Taken from [16])

The layout of the devices on the UCT development board can be seen in figure 2B.3.

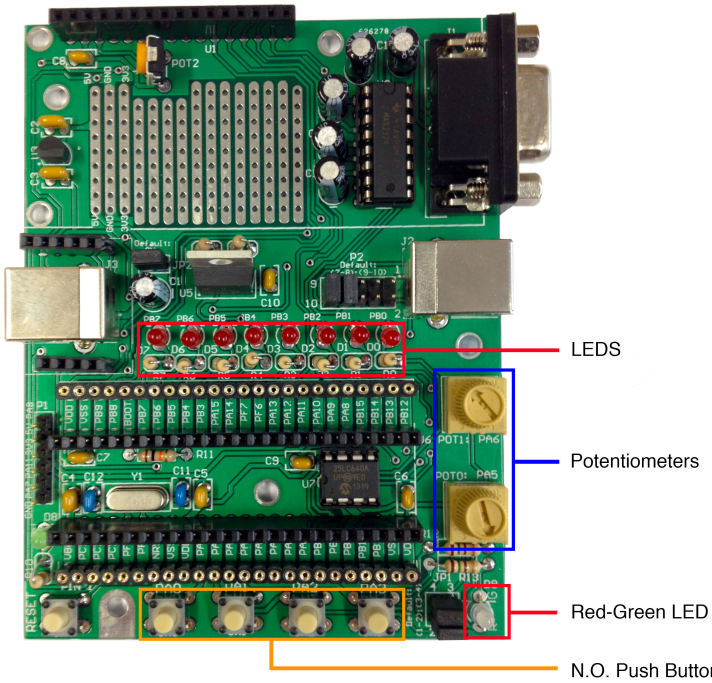


Figure 2B.3: The simple input/output devices on the UCT STM32F0 development board The basic I/O devices on the development board have been labelled as follows: **red** - digital outputs which have been connected to LEDS, **orange** - digital inputs which have been connected to normally open push buttons and **blue** - analogue inputs connected to two potentiometers.

2B.4 Questions and coding

Please answer the following questions in your practical report and where necessary modify or write code into your **main.c** file. Build and test your code after each code modification step.

(a) Using the schematic for the UCT STM32F051 Dev. Board at the end of the practical sheet (figure 2B.4), identify which General Purpose Input/Output port(s) and pins that the LEDS are connected to on the UCT STM32F0 development board. **[1]**

(b) What does the code example, which you added to the Eclipse project above, produce on the UCT STM32F0 development board? **[1]**

(c) Explain what this means **GPIOB->MODER.** **[2]**

(d) Explain what each of the following lines of code do in the **main.c** example code, using the Reference sheet at the end of this manual for guidance. **[6]**

(i) **RCC->AHBENR |= 1<<18;**

(ii) **GPIOB->MODER|= 0x00505555;**

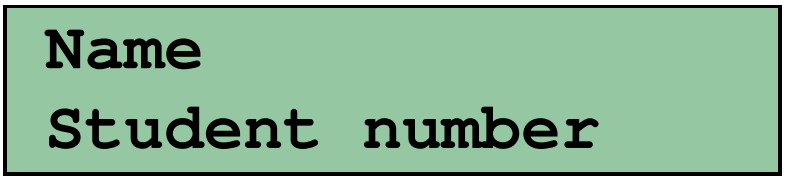
(iii) **GPIOB->ODR = 0b0000010000001111;**

(e) Write **RCC->AHBENR |= 1<<18;** in an alternative way that will produce the same effect on the system. **[2]**

(f) Explain what the following block of code does in the **main.c** example code. **[2]**

```
GPIOB->ODR = ~GPIOB->ODR;
for (int i = 0 ;i<=100;i++)
    for (int j = 0;j<=5000;j++);
```

(g) Modify the example code in your **main.c** file to display your details in the following message on the LCD.



Copy this code and paste it in your report. **[2]**

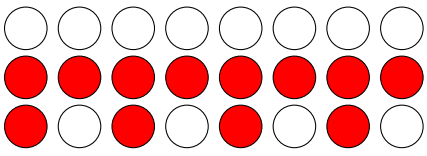
(h) Create a function called `Delay()` which uses two nested `for` loops to create an approximate 1 second delay. Create two symbolic constant values `DELAY1` and `DELAY2` which are initialised at the start of the program. These will be used as the values for your `for` loops to count up to. You can therefore change the length of your delay easily. Copy this function into your report. **[3]**

(i) Create three variables called **bitpattern1** , **bitpattern2** and **bitpattern3** which hold the values which will produce the following patterns on the LEDS. **[3]**

(i) **bitpattern1**

(ii) **bitpattern2**

(iii) **bitpattern3**



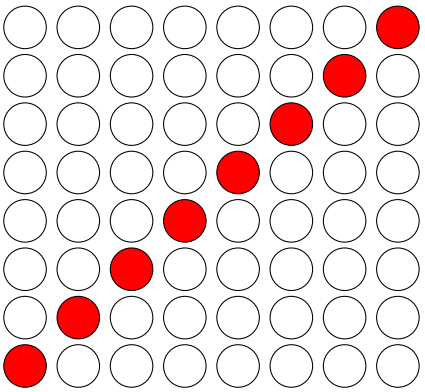
Copy these code line(s) and paste them in your report.

(j) Modify the example code in your **main.c** file to display **bitpattern1** , **bitpattern2** and **bitpattern3** on the LEDS with a 1 second delay in between each pattern display. Copy this code and paste it in your report. **[3]**

(k) Modify the example code to produce an 8-bit binary counter, which will display the current count value on the LEDS. The counter must start counting up from **0** and should keep counting up in increments of 1 until the maximum value is displayed on the LEDS. The counter should be incremented approximately every second. Copy this code and paste it in your report. **[5]**

(l) Modify the example code in your **main.c** file to display the following pattern on the LEDS with a 1 second delay in between each pattern. Repeat the pattern once the end of the pattern sequence is reached.

Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
Step 7
Step 8



Copy this code and paste it in your report. **[5]**

(m) Ensure that your **main.c** file is well commented and **copy the entire contents of the main.c file and paste it onto a new page in your report**. **[5]**

2B.5 Practical Submission

Submit your completed practical report on VULA as a **.pdf**[2] under the correct assignment. Show all your calculations.

Your document must be named as follows so that it is easily identifiable:

Prac2B-STUDENTNUMBER1.pdf

Additionally you must submit your **main.c** as a separate file on VULA. It must be renamed as follows so that it is easily identifiable:

Prac2B-STUDENTNUMBER1.c

Do **NOT** zip the files together but upload them individually to VULA. Ensure that your name and student number is clearly written in your practical report. Practical reports must be completed and submitted on VULA by 23h55 by the due date on the VULA calendar.

[2] **Note** if it is submitted as a **.doc/.docx** or equivalent format file, it will **NOT** be marked.

2B.6 Marks Breakdown

Marks	
Questions	
(a)	1
(b)	1
(c)	2
(d)	6
(e)	2
(f)	2
(g)	2
(h)	3
(i)	3
(j)	3
(k)	5
(l)	5
Code	5
Total	**40 Marks**

Up to 5 marks will be deducted for untidy reports or uncommented code. **5% will be deducted per day for late hand in's for up to one week, after which you will receive 0 and the practical report will no longer be accepted.** Please ensure that you submit the files in the correct place and read the instructions given on VULA with regards to file naming. **NO EMAILED** pracs will be accepted.

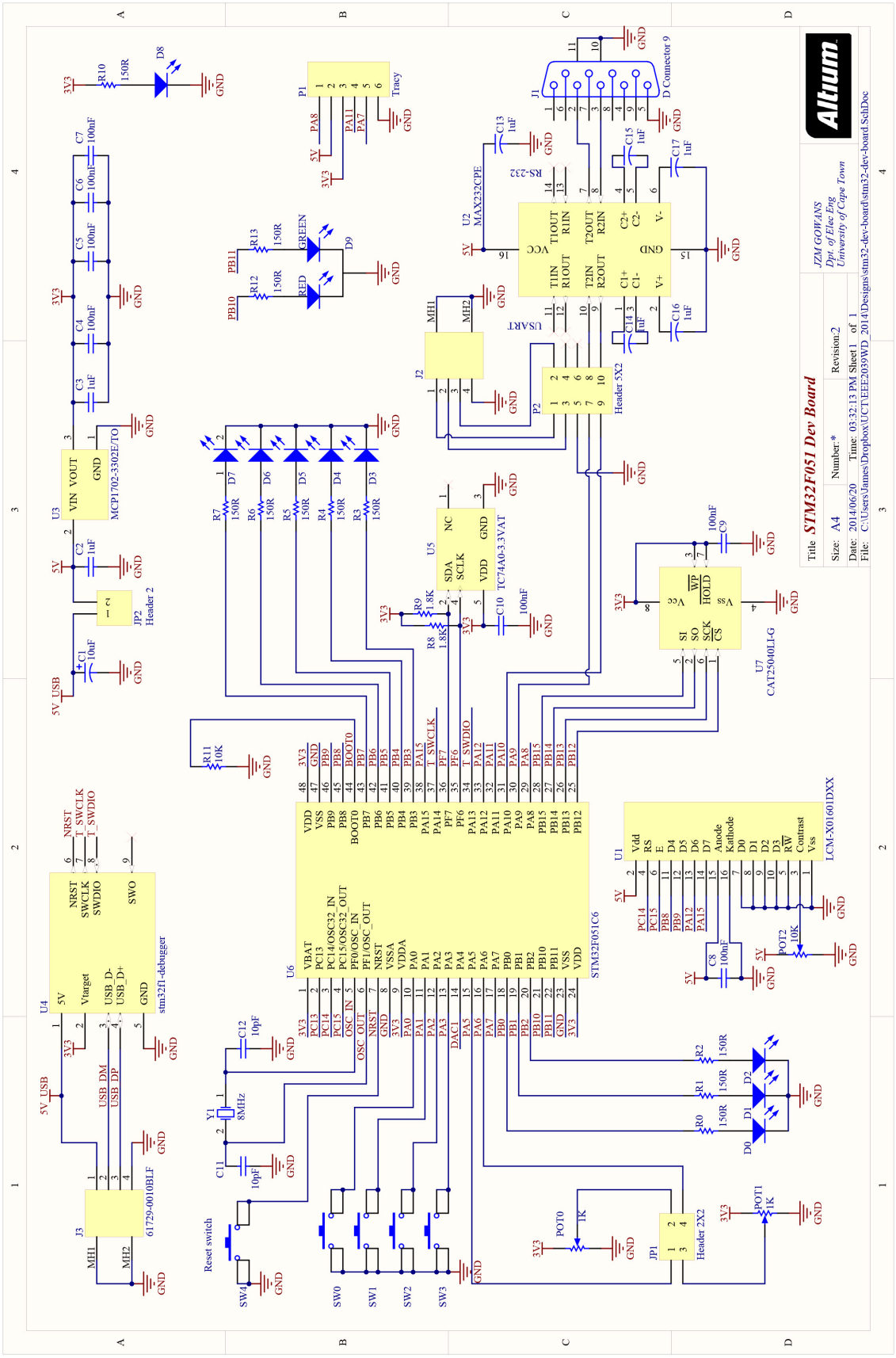


Figure 2B.4: Circuit diagram for the UCT STM32 Development Board. (Taken from [6])