Practical 4B

Introduction to Microcontrollers - Analogue to digital convertor and External interrupts

Before the practical ensure that you complete the following:

Read and revise:

Read the **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**.

Read the **Analog-to-digital convertor** chapter (pp. 204-240) in the **STM32F0x1/STM32F0x2/STM32F0x8 Reference Manual** [16].

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

4B.1 Introduction

In this practical you are going build a simple voltage data logger using the **Analogue to Digital Convertor**.

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:

Prac4B-STUDENTNUMBER

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

4B.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. Please see Appendix B.1 for more details.

- Open Eclipse on your computer
- Create a new project and name it as follows:

  Prac4B-STUDENTNUMBER
- Setup the project environment as described in the instructions in Appendix B.1

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

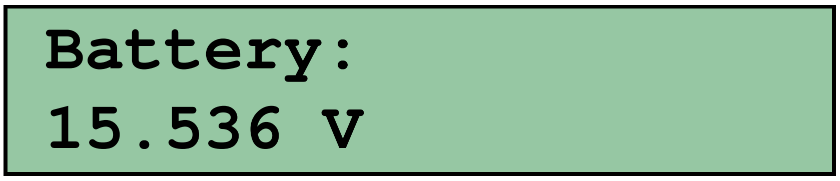
Connect your UCT development board to the PC using the USB cable. You are now ready to begin your task.

4B.3 Task

In this practical you will be simulate a basic battery monitoring circuit on your UCT STM32F0 Development Board. Data loggers are devices that read data in and store that data in a memory. The data can then be retrieved from the logger at a later stage. This logger will only have four memory slots. The incoming data will come from the potentiometer POT0. You will need to write the program in C to simulate this system and then run it on the UCT STM32 Development Board.

Battery Monitoring Circuit
A battery monitoring circuit gives feedback about the current state of the battery's voltage. It is important in battery powered devices to have some external signal to indicate that the battery's voltage has dropped below an executable value. In a microcontroller system we need to convert this analogue signal in to a digital value which we can read and use. To do this, we can use the built in Analogue to Digital convertor peripheral module.

Use the potentiometer **`POT0`** to simulate the battery voltage on this system. When the voltage on the ADC falls below the minimum value, the LEDs `D0 to D7` must be turned ON to indicate a low battery condition. Sample the battery voltage approximately once every second. When `SW3` is pressed the battery voltage must be displayed on the LCD as follows:



```
Battery:
15.536 V
```

Analogue to digital convertors
When we refer to analogue signals we are talking about signals, which can take on a range of values. Thus far we have concerned ourselves only with digital signals, which are always either TRUEo r FALSE. Analogue signals can take on any voltage in a specified range. As you are aware many physical phenomena are inherently analogue. Examples of this are voltage, temperature and position.

Clearly analogue signals cannot be ignored in a digital system and for this reason our microcontroller has a 12-bit analogue to digital converter built in to it. The function of the converter (also known as an A/D or an ADC) is to read in an analogue voltage and convert it to a binary number, which can be handled by our digital system.

When we refer to a 12-bit converter we are referring to the number of bits in the digital number that it produces. For an output number that is 12-bits wide there are 4 096 possible output combinations. The A/D takes in an input voltage in the range of 0 to 3 V and this range is split into 4 096 different levels, each level represented by a different output number. The A/D is linear in its conversion and thus each level represents 3 V/ 4 096 levels = 0.7 mV/level. (0 V gives a 12-bit digital value of 0_{10} and 3 V gives a value of 4095_{10}). The output digital value is given by:

$$\text{Output Value} = \frac{V_{input}(2^n - 1)}{V_{span}} \tag{4B.1}$$

Where V_{input} is the voltage input on the A/D channel, n is the resolution in bits of the A/D and V_{span} is the A/D voltage conversion range (AV_{SS} to AV_{DD}).

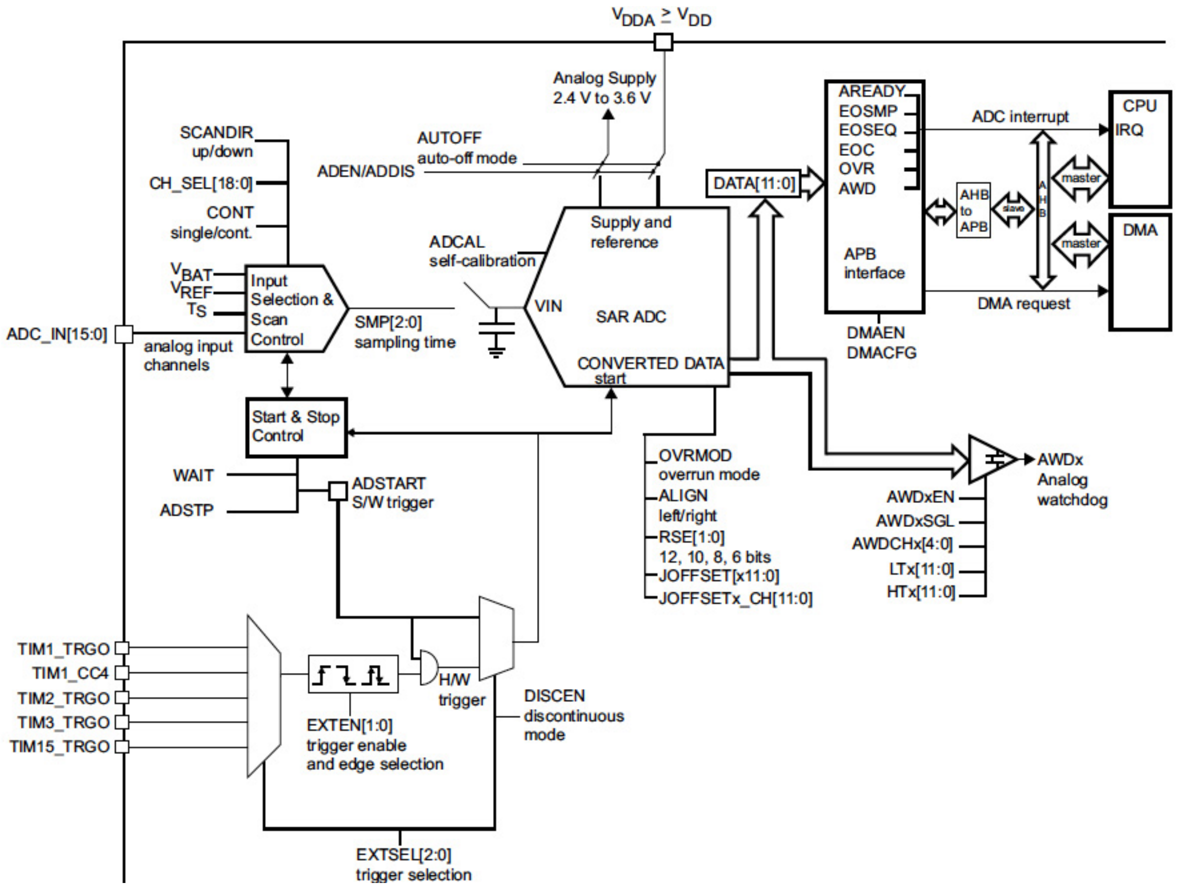


Figure 4B.1: Block diagram of the ADC in the STM32F0. (Taken from [16])

To configure an the A/D:

1. Enable the ADC clock `ADCEN` in the `RCC->APB2ENR` register.
2. Set up the ADC configuration using the ADC configuration register `ADC->CFG1`
   (a) Select single or continuous conversion mode
   (b) Select data alignment (right or left justified)
   (c) Select ADC resolution (12, 10, 8 or 6 bits)
   (d) Select scan sequence direction (up or down)
3. Select which channels to sample `ADC->CHSELR`
4. Enable/disable ADC interrupts
5. Once the ADC is configured, start it as follows:
   (a) Set `ADEN` in the `ADC->CR` register.
   (b) Wait until `ADRDY` is set in the `ADC->ISR` register. This bit is set after the ADC startup time is complete.
6. To take a sample, set `ADSTART` bit in the `ADC->CR` register.
7. Wait until the conversion has ended. Check the `EOC` bit in the `ADC->ISR` register, when it is set the conversion is complete.
8. The converted result is now available in the `ADC1->DR` data register. A read of this register clears the `EOC` bit and will mean that another conversion can be initiated.

4B.4 System Design

Before starting the practical:

- Before opening Eclipse, download the following files from:

  VULA → EEE2046F → Resources → Practicals → Code:

  - **lcd_stm32f0.c**
  - **lcd_stm32f0.h**
  - **template.c**

  and save them to the Desktop.
- Open Eclipse on your computer.
- Create a new project and name it as follows:

  Prac4B_STUDENTNUMBER1
- Setup the project environment as described in the instructions in Appendix B.1.
- Include **lcd_stm32f0.h** in the **includes** folder and **lcd_stm32f0.c** in the **src** folder. These files provide you with default functions to write to the LCD screen. Follow the instructions in Appendix B.1 on how to import files into a project.
- Open the existing **main.c** file in the Project Explorer. Delete the contents of the **main.c** file and copy and paste the contents of the **template.c** file into the **main.c** file.

- Plug in your **UCT STM32F0 Development Board** using a USB type A to B cable.
- **Build** your project. Click on **Project** → **Build All (Ctrl-B)**.
- If your project compiled with no errors, debug it and program our target processor.
- Initialise the debugger and run the code.
- You should now see the following message appear on the LCD of your development board.

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

You are now ready to begin your task.

4B.5 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) Write C initialisation functions in your **main.c** file for the GPIO ports and ADC[1] :

- `init_GPIO()` (initialises the `GPIO` ports for pushbutton inputs `(SW0, SW1, SW2, SW3)`, analogue input `(POT0)` and LED outputs `(D0 to D7).)` (3)
- `init_ADC()` (8-bit, right-aligned, continuous mode, no-interrupts) (4)

Copy these functions and paste them in your report. **[7]**

(b) What value will be output by the STM32F0 ADC (8-bit mode), if we want the battery low voltage condition to be 14 V. Assume that the maximum battery voltage is 24 V (simulated with the potentiometer `POT0`) and the STM32F0's ADC voltage reference range is 0 to 3 V. **[3]**

[1]Note: You may reuse code from previous practicals where needed/appropriate in this practical.

(c) Write a C function called `check_battery(void)`, which updates a global variable called `battery_voltage` with the current "battery voltage" value read from `POT0`, displays the battery voltage on the LCD and turns the LEDs `D9` ON when the voltage drops below the low voltage condition. Copy this function and paste it in your report. **[10]**

(d) Write a function called `display(void)`, which displays your values on the LCD when push buttons `SW0` (Welcome message), `SW1` (Battery monitor message), or `SW2` (display battery voltage) are pressed. **Message after `SW0` is pressed: Welcome Message**

```
EEE2046F Prac4B
Your_Name
```

Message after `SW1` is pressed:

```
Battery Monitor
Press SW2
```

Message after `SW2` is pressed:

```
Battery:
15.536 V
```

Copy this function and paste it in your report. **[10]**

(e) Complete the code and test it. 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**.

4B.6 Practical Submission

Submit your completed practical on VULA as a **.pdf**[2] under the correct assignment and submit your completed and working **main.c** file. Show all calculations and include your code segments under the correct sections in your report. Do NOT zip your project folder, you will receive 0 for the code section. The **main.c** file can be found in a subfolder called **src**.

Your document and code must be named as follows so that they are easily identifiable:

Prac4B-STUDENTNUMBER.pdf
Prac4B_STUDENTNUMBER.c

Do **NOT** zip the files together but upload them individually to VULA. Ensure that your name and student number is clearly written on your practical report.

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

4B.7 Marks Breakdown

Marks	
Questions	
(a)	7
(b)	3
(c)	10
(d)	10
Code	10
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.