# Design Construction & Test Individual report

## The University of York
## Department of Electronics Engineering

**Title: Fully functioning digital multimeter**

**Department: Department of electronics engineering**

**Date: April 15, 2019**

**Group number: Team B1**

**Members: Naijia Liu, Greg Guyll, David Denman, James Sweeney, Will Nightingale, Ben Coleclough**

**Name: Naijia Liu**

# 1. Introduction

The following report outlines the basic operations that our multimetre could do and main features as well as its specifications including technical overview. Meanwhile, any on-going problems and achievement against specification are to be discussed and potential improvements are to be suggested. Due to the group work of our design, work breakdown is going to be introduced and commented.

# 2. Summary of the group's project

## 2.1 Description of entire system

The approach of our entire system was to initially investigate what we intended to measure and how we may need in designing the circuits. To design and implement a fully functioning digital multimeter, DC voltage, DC current and resistance meter were basically designed by several Op-Amps and resistors, which were arranged firstly by hand-writing roughly and then chosen as certain values by being simulated in pSpice. Once satisfying results from simulation gotten, circuits are built on breadboard for testing against the simulation results. To make all circuits together, our hardware team also soldered them on the stripboard and had a PCB designed at the end.

Design of the software started from testing STM32F04VG discovery board by using LEDs as well as buttons on the mother board and programming LCD screen, designing user interface by importing "stm32f4xx.h", "stdio.h" library and "PB_LCD_Drivers.h" library. As software team acquired enough knowledge of how to program the ARM board through mini-USB, according to demanding from hardware team and project requirement. For most of the measuring modes, the initial software implementation was based on a function which performed the inverse of the relevant circuits, converting STM32's ADC reading back to the unknown voltage, current or resistance through the given circuits. Thus, our software team worked out these equations and tested them by giving input voltage below 2.5V separated by hardware. Meanwhile, the frequency meter and ADC initialization was implemented by "stm32f4xx_discovery.h" library. Peripherals, timer and USART are initialized to enable counters and clocks. As tested by sending a square wave or pulse with amplitude between 1V and 3V from the function generator, our frequency metre can measure frequency between 30Hz to 2MHz.

Data storage system was implemented to keep the final value converted by ADC in different mode and frequency counted by timer in an array and SW8 can view data stored in ARM. Meanwhile, auto-ranging is designed both in hardware and software. In software, displaying on LCD screen and conversion algorithms changes according to ADC to show the value more precisely. In hardware, our group used switches such as DG408 and DG409 in order to switch gain in order to get enlarged output voltage. Unfortunately, it hasn't been fully tested by combining hardware with software.

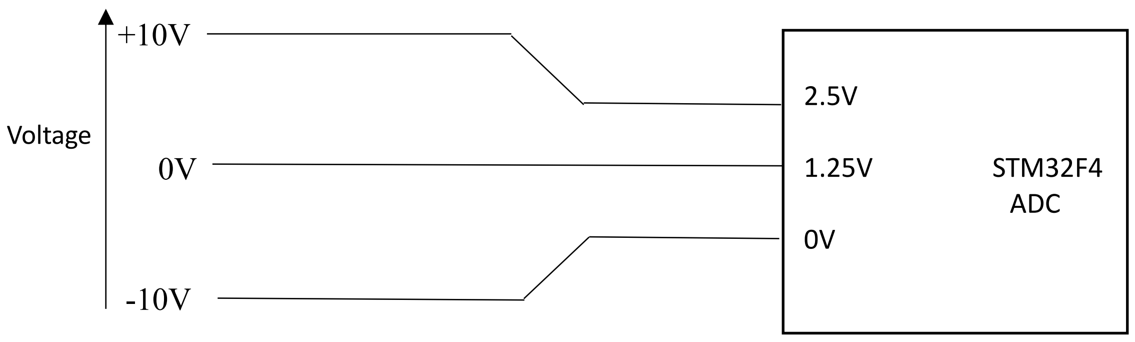
To protect the whole circuits, the power management circuits is designed to judge power consumption, which intended to provide an alarm in case of large input voltage into STM32 mother board. However, only three functions were written in software which can wake a buzzer up without the analogue buzz circuits.

## 2.2 Product and project planning

### 2.2.1 Choice of features

#### 2.2.1.1 DC voltage

As requested in the project outlines, our multimeter should be able to measure input voltage between +10V and -10V. Since STM32 board can use ADC to read voltage directly, our analogue group used a digital switches to change gain in case of missing tiny signals. In case of destroying the STM32 board, our group assumes that the maximum voltage that the ADC can take is 2.5V and our analogue team designs the circuits providing voltage below 2.5V linearly. By the lecture slides [1], the details of ADC range is given below:



According to the graph drawn above, our analogue group could transfer unknown voltage to 0 to 2.5V and our software group was to transfer 0 to 2.5V back to voltage between -10V and 10V to display on LCD.

#### 2.2.1.2 DC current

From the project requirement, our multimeter should be able to measure input current between 1A and -1A. It is quite similar with the voltage programming for software group to find the linear equations transferring ADC readings back. Unluckily, Dave didn't provide coefficients and tell how the current circuits works, this part was just assumed and left as one linear equation without reasonable auto-ranging functions, which was tested.

#### 2.2.1.3 Resistance

As for resistance, our multimeter should be able to measure resistance up to 1Mohms. Since the STM32F4 board cannot directly measure the resistance through the circuit, our group designed the circuits was to measure the potential deference across some reference resistance when a constant current flows. Besides, Ohm's law and potential divider are applied.

#### 2.2.1.4 Frequency metre

Because our group haven't designed an AC to DC converter and circuits to convert sine wave to square wave, the frequency metre can only reads the frequency of a square wave with an amplitude between 1V and 3V. The measuring range is between 30Hz and 2MHz. The resolution is 3% at the worst case. It is realized by setting 16MHz TIM3, with clock frequency 2MHz with prescalar from 1 to 65536, to record rising edge and falling edge in order to calculate the period. Due to the period, we can easily acquire frequency by the equation $= \frac{1}{T}$.

#### 2.2.1.5 Data Storage

Data storage is applied in the fourth case of function *Mode()*. This case should display the final value that ADC reads in different mode and the final frequency tested stored into an array when different mode was running. As to switch to view stored values in different modes, SW8 is used. In this mode, ADC still can read input voltage from the pin and LEDs on STM32 mother board show whether there is an input voltage to the ADC.

#### 2.2.1.6 Auto-ranging

The auto-ranging hasn't been fully tested by combining hardware with software. This is programmed in function *Mode()* and equations of converting ADC readings are implemented in three *Readxx()* functions. In hardware, a digital switch DG408 or DG409 is used. It can receive an enable from the software or be switched manually. In measuring resistance circuits, it changes different reference resistance. In measuring voltage circuits, it changes different gain through a non-inverting amplifier. When displaying tested value, the software changes units and converts values.

### 2.2.1. User interface and button used

The interface is designed user-friendly. By pressing the blue button on stm32 mother board, the LCD is switched as an order of DC voltage, DC current, resistance, frequency and data storage. In case of screen flashing, by generating an interrupt, ADC reading and screen updating happen 1/16 clock circle of system, which is the initial value of *SystemCoreClock*.

The STM32 discovery board is used in an LCD display and switching mode of measurements. The black button at right is for reset and the blue button is to switch mode. The interface on 2x16 LCD shows the name of current mode at the first row and the second row shows measured or stored values.

### 2.2.2 Target users

Because we only have implemented four functions which are DC voltage, DC current, frequency and resistance measurement, our product can be designed for students maybe in foundation or A-level year students who are concentrating mostly on learning knowledge in DC environment. This product using STM32F407 discovery board as well as improving correctness by way of auto-ranging is easy for students to view measured values and corresponds to analogue values' ranges that users are generally using.

## 2.3 Analysis of effectiveness

In terms of our achievement, our group made circuits to measure voltage, current in DC, resistances and frequency. Meanwhile, data storage is designed to store the final values tested in four different modes. To achieve the audio feedback, a buzz was intended to be implemented but only designed by three functions in software.

There are many areas where we can improve our multimeter, predominantly by improving the accuracy of our measuring circuits as well as the interpretation of the voltage values into STM32 discovery board. We would also seek of increasing the range and decreasing the resolution that the multimetre could measure each of our further testing. This could be done in further design iterations as well as using components with better accuracy.

In designing and building circuits in practical, most of the resistors are requested to use 0.1% tolerance resistors. As shown later in the sheet of section 3.2.5, there is a gap which is

300ohm resolution for testing large resistor, caused by programming in software and the worst case influencing measuring resistors between 100ohm to 1Mohm has 1% error theoretically and between 10 to 100ohm, it has 3% the worst, which is caused in analogue circuits designing.

Furthermore, in software testing, we found surrounding noise and unstable current influencing reading from ADC a lot, which causes inaccurate and nonstable measurements jumping between ranges. Changing values slightly in equations ensures reading a bit more precise but the error cannot be eliminated completely.

Software-specific improvements may include expanded data-analysis functionality such as being able to transfer data to PC and also providing an implementation for inductance, diodes as well as capacitance measurements. The frequency measurements should also accept sine wave and other waves with higher amplitude and frequency.

# 3. Individual contribution

## 3.1 Individual role in group

Since the project consists of hardware and software building, our members are distributed missions according to their interests and abilities. Hence, I am in charge of the software implementation, including interface design, ADC using, frequency metre and writing code of switching reference resistance etc. Meanwhile, I worked out linear relationships to transfer readings back to actual inputs with James Sweeney together during the labs. However, most of codes writing and testing was contributed by myself by connecting with the power supply and the function generator. Moreover, the implementation of frequency was completely done by myself but due to time limit, the frequency measuring circuits hasn't been designed.

## 3.2 Technical detail of individual contributions

### 3.2.1 ADC reading

From the instruments of STM32 board, it is discovered that the ADC uses the 3V supply for the whole board as the 3V reference, which is generated from the 5V supply using a BiCMOS voltage regulator and a small signal schottky diode. Therefore, this voltage is quite temperature variable and the measurement accuracy is quite limited. Therefore, an external ADC is designed to change any voltage through the analogue circuits between 0 and 2.5V. [2]

As required, one of ADCs in the STM32 mother board needs to be used to read output voltage from analogue circuits so that I choose ADC12_IN14. In coding, a function created as *initIO()* includes initialization of ADC1 as an analogue input mode relating to pin(PC4), channel 14. We also wanted ADC to convert discontinuously and DMA should be disabled in case of external triggers. Initialization of ADC together with peripherals, USART2 and TIM3 initialization are called in *HwInit()*, which is used at the beginning of the main function. [2]

For readings converted values from ADC, a function named *read_ADC1()* can read 16-bit values gotten from ADC converter and return them until end-of-conversion bit goes high. Values returned by this function is going to be used in algorithms programmed in ARM microcontroller.

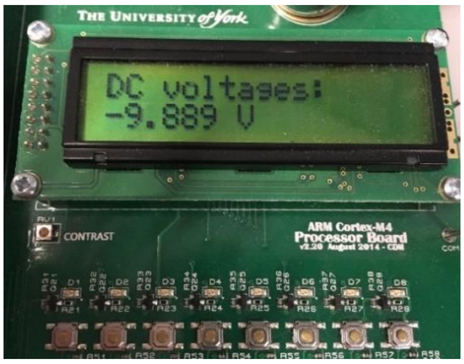
Since the value representing the voltage read by ADC is 12 bit right aligned so that we shift to isolate bits 15-8. After that, by equation [3]

$$ADC\ reading\ voltage(V) = \frac{12\_bit\ ADC\ reading}{ADC\ resolution\ (57600)} * Reference\ voltage\ (2.5V)$$

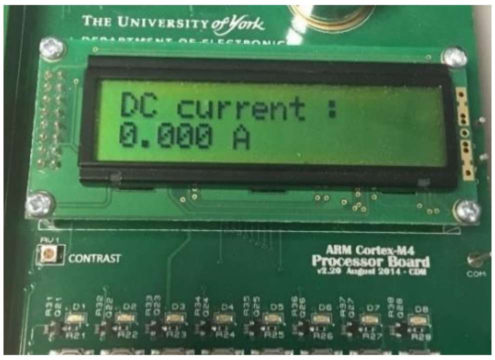
We can get the actual voltage in volts that our ADC reads from analogue circuits and we can use this for calculations in order to transfer back to unknown resistance, voltage or current.

### 3.2.2 Interface design

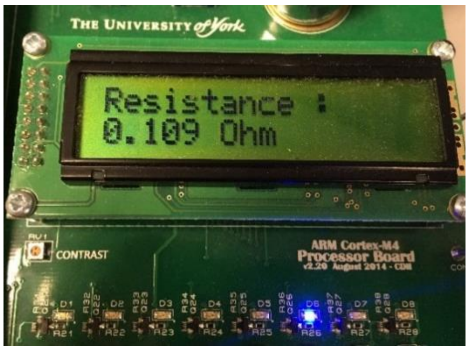
There are four functions consisting of what our multimeter can do, which transfers ADC reading back to tested DC voltage, DC current, frequency of square wave or resistance. They are called in *Mode()* function by means of five *switch case* statements including data storage. The modes can only be changed in the order of DC voltage, DC current, resistance, frequency, data storage and then back to DC voltage by pressing the blue button. The interface on LCD screen is designed as following:



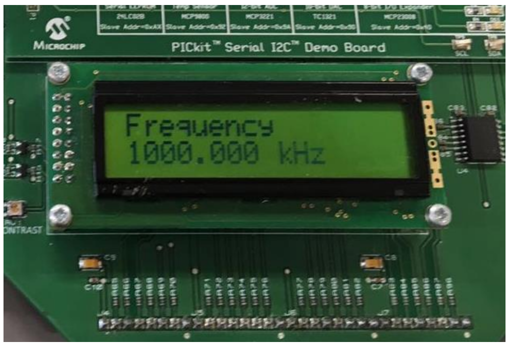
*LCD screen in DC voltage mode*



*LCD screen in DC current mode*



*LCD screen in Resistance mode*



*LCD screen in frequency mode*

In data storage mode, viewing values stored in different mode uses the button SW8. Unfortunately, the button debouncer wasn't implemented so that there is possibility to change values all the time when user keeps pressing this button.

The calculation transferring ADC reading back to measured items is implemented in different functions which are *ReadVoltage(), ReadCurrent()* and *ReadResistance()*. They are called in three *switch cases* to display values returned from these functions on LCD screen.

To make the board more amusable, lights on STM32 mother board can be lightened. This is realised by setting a light initialisation function *initialiseLEDs()*. In this function, PD12, PD13, PD14 and PD15 which are GPIOs as digital outputs and related clock registers are enabled. Rule of the lights follows the 16-bit ADC reading from analogue circuits because values is written in these LEDs.

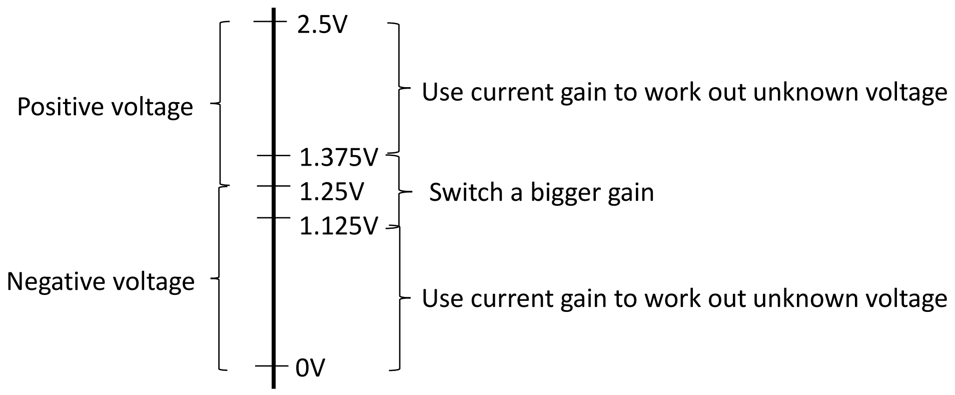
### 3.2.3 Button implementation

In testing the board, I found there was no internal debouncer of the blue button on the STM32 mother board, which our group mainly used to switch modes. To solve this problem, beside setting a function to check pushing button, there is also a function called *CheckReleasebutton(),* which is used in checking releasing button. In main function where after *CheckPushbutton()*, this function is called in a *while* loop to ensure mode switched once the button is released.

Global reset button which is the blue one on the STM32 mother board resets the whole program, which causes it back to reading voltage whatever mode is at. SW8 is only used in switching to view different stored values.

### 3.2.4 DC voltage linear equations

Equations that the software group works out is based on analogue circuits. As the ADC can read DC voltage directly, we used external ADC to convert -10V to 10V voltage to 0 to 2.5V and let ADC read this. Meanwhile, according to the auto-ranging circuits with various gain from our hardware group, I implemented three *if* statements to check whether the input voltage is small enough for switching the range from the smallest gain to the largest one. The condition we checked the small signal in *if* statement is giving below:


2.5V
Positive voltage
Use current gain to work out unknown voltage
1.375V
1.25V
Switch a bigger gain
1.125V
Negative voltage
Use current gain to work out unknown voltage
0V

Every time the range is switched to the next, we intend the ADC to reread the voltage from the analogue circuits because we also change the gain by means of giving a digital enable to the digital switch DG409. To keep small signals readable when using a large gain, 5 decimal numbers are retained. Gain we use for each range and the linear equations which transfers ADC readings back to the input voltage is given as following:

| **Unknown voltage range** | **Gain** | **Linear equations** |
|---|---|---|
| Positive: 1V – 10V<br>Negative: -1V – -10V | 0.123 | $V_{unkonwn} = 8 * V_{ADC} - 10$ |
| Positive: 0.125V – 1.25V<br>Negative: -0.125V – -1.25V | 1 | $V_{unkonwn} = V_{ADC} - 1.25$ |
| Positive: 0.0125V – 0.125V<br>Negative: -0.0125V – -0.125V | 10 | $V_{unkonwn} = 0.1 * V_{ADC} - 0.125$ |
| Positive: 0.00125V – 0.0125V<br>Negative: -0.00125V – -0.0125V | 100 | $V_{unkonwn} = 0.01 * V_{ADC} - 0.0125$ |

### 3.2.5 Resistance equations

The analogue circuits measuring resistance provides constant current when selecting different reference resistance from the largest to the smallest. Our group applies 2.5V reference voltage on the reference resistance and it gives a constant current which can be calculated by $\frac{2.5V}{reference\ resistance}$. ADC reading is the voltage of the tested resistance. By the potential divider, we can summarize the equation for calculating the unknown resistance as follows:

$$R_{unknown} = \frac{R_{reference} * V_{ADC\ reading}}{2.5}$$

Starting from the largest resistance which is 1Mohm, providing the smallest constant current through the circuits, to protect the circuits won't be burnt. From the equation, we know that tested resistance should be big when we use a large reference resistance.

However, different from voltage measurements, shifting to smaller resistance is in accordance with the ratio $\frac{to\ shift\ resistance}{current\ used\ resistance}$. If the voltage read by ADC is below or equal 1.25 or 0.5 in different cases, the tested resistance is small and we need to switch a smaller resistance in case of systematic error. Similarly with the voltage circuits, once we switch the resistance, ADC should reread the voltage. Equations applied in different cases are shown below:

| **Resistance range** | **Reference resistance** | **Ratio** | **Equations** |
|---|---|---|---|
| 1Mohm - 500kohm | 1Mohm | 1.25 | $\frac{1000000 * V_{ADC}}{2.5}$ |
| 490kohm - 98kohm | 490kohm | 0.5 | $\frac{490000 * V_{ADC}}{2.5}$ |
| 100kohm - 50kohm | 100kohm | 1.25 | $\frac{100000 * V_{ADC}}{2.5}$ |
| 49k9ohm - 9k98ohm | 49k9ohm | 0.5 | $\frac{49900 * V_{ADC}}{2.5}$ |
| 10kohm - 5kohm | 10kohm | 1.25 | $\frac{10000 * V_{ADC}}{2.5}$ |
| 4k99ohm - 998ohm | 4k99ohm | 0.5 | $\frac{4990 * V_{ADC}}{2.5}$ |
| 1kohm - 500ohm | 1kohm | 1.25 | $\frac{1000 * V_{ADC}}{2.5}$ |
| 500ohm - 0 | 500ohm | / | $\frac{500 * V_{ADC}}{2.5}$ |

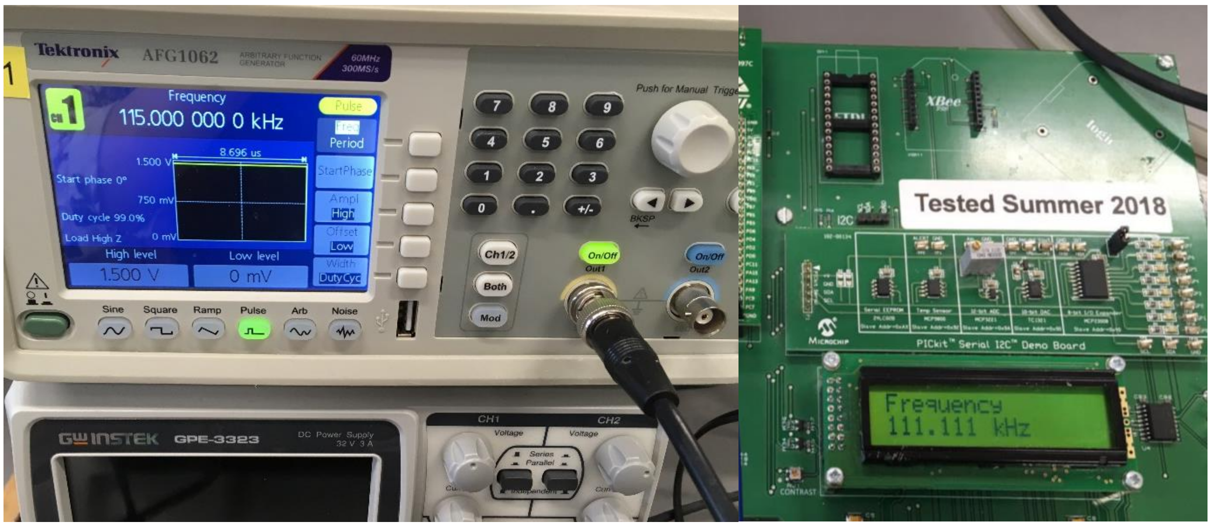
### 3.2.6 Frequency metre

The frequency metre is designed to measure the frequency of a square wave or pulse. The frequency metre works by setting a 2MHz peripheral clock which is gotten from 16MHz TIM3 divided by prescalare(8). This is the maximum frequency of our multimetre can measured. Channel 1 of TIM3 resets at rising edge and CC1R saves at rising edge so that the period can be detected by recoding the rising and falling edge. Data is sent to UART2, which is initialized with default and turned off synchronized clock. By initializing TIM3 firstly, it is like a counter counting numbers from 1 to 65536. By using "stm32f4_discovery.h" library, initialization of GPIOs, related peripheral clock, TIM3 and UART2 can be found in instruction and implemented.

The period by comparing the rising and falling edge can be acquired by function *TIM_GetCapture1(TIMx)*. To display them in kHz on LCD display, considering the 0.5us tolerance the equation was discovered as follows:

$$f = \frac{1}{period * 0.0005}$$

By testing with a square wave about 1-3V with frequency between 31Hz and 2MHz, as the frequency is increasing, the error could be big. The worst case for the error is about 3% but the reading can be more precise when input signal has the frequency with a factor of 10, which is because a period of 2MHz reference clock just finished.



## 4. Reflective summary of group work

In conclusion, our group members are enthusiastic doing jobs arranged and most of them showed up in lab session every week. Hardware team finished the design of voltage and resistance measuring circuits by the end of spring term, but the implementation hasn't been finished by the end of report deadline. Software implementation as well as testing were done during the Easter holiday separately with hardware. Since our group requested a PCB connected with the STM32 board, time of testing was wasted waiting for components and PCB arrived.

However, in the corporation, our hardware group suggested separating hardware and software work, the combination is quite hard because the results are not as we expected. Software team needs to wait for the circuits' design of hardware team so that I tested the board including LCD display and ADC by giving voltage below 3V through an external ADC and a buffer designed by myself and programmed the algorithms by some random rules. Due to the absence of Dave, how the current circuits worked wasn't discussed so in the code, the current function is only be filled by one linear function without signal controlling auto-ranging of analogue circuits.

Although our hardware group designed the circuits very quickly, the correctness of the circuits cannot be promised since it wasn't connected with stm32 board before it was made as a PCB. Lab sessions are made full use of to discuss the linear equations for programming but

Ben and Will didn't show up sometimes. Hence, they actually did little work. Meanwhile, software implementation was totally designed on my own out of lab session so that from my perspective, it is a bit difficult for the group communication out of labs.

According to the project requirement that our group satisfied, the detailed breakdown and peer assessment are listed below.

Hardware group:

| Task | Contributor(s) | Testing | Implementation |
|---|---|---|---|
| Measure DC voltage | Will & David | Will (breadboard) | Greg |
| Measure DC current | David | / | Greg |
| Measure resistance | Will & David | Greg | Greg |
| Power management | David | Will & Greg | Greg |
| PCB design | Greg | / | Greg |

Software group:

| Task | Contributor(s) | Testing |
|---|---|---|
| Read from ADC, display values on LCD screen | Naijia | Naijia |
| Interface design and selectable display | Naijia | Naijia |
| Initialization of lights and lightening lights | Naijia | Naijia |
| Calculate resistance from ADC reading | Naijia & James | Naijia |
| Frequency metre design and implementation | Naijia | Naijia |
| Calculate current from ADC reading | Naijia | Naijia |
| Calculate voltage from ADC reading | Naijia & James | Naijia |
| Implementation of measuring circuits in code | Naijia | Naijia |
| Implementation of auto-ranging for different measurements | Naijia & James | / |

Peer assessment:

Note: David told us that he applied for a period of absence so that he didn't provide much help after spring term and even didn't show up in the demonstration. Ben just showed up several times for labs but didn't do anything for the group. He didn't get in touch anymore and so other group members didn't exactly know whether he was still in our group. Will didn't show up in some group meetings and missed some lab sessions.

Naijia Liu: 40  Greg Guyll: 15  David Denman: 30  James Sweeney: 10  Will Nightingale: 5

Ben Coleclough: 0

## 5. Reference

[1] Second lecture: Multimeter principles (part one) *[ONLINE]* Available at: https://www.elec.york.ac.uk/internal_web/meng/yr2/modules/Design_Construction_and_Test/Lecture_1a_Multimeters_One_2019.pdf [Accessed 15 April 2019]

[2] ADS1148-Q1 Automotive, 16-bit, 2-kSPS, Analog-to-Digital Converter with Integrated Programmable Gain Amplifier (PGA), Reference, and Oscillator *[ONLINE]* Available at:

http://www.ti.com/lit/ds/symlink/ads1148-q1.pdf [Accessed 16 April 2019]

[3] Fifth lecture: ADC and timers on the STM32F407 [*ONLINE]* Available at:

https://www.elec.york.ac.uk/internal_web/meng/yr2/modules/Design_Construction_and_Test/Lecture_2b_ADC_Timers_2019.pdf [Accessed 15 April 2019]

[4] Getting started with Keil MDK v5 and the STM32F4 Discovery Board *[ONLINE]* Available at:

https://www.elec.york.ac.uk/internal_web/meng/yr2/modules/Design_Construction_and_Test/dajp/GSW_ARM/Chapter_11.pdf

## 6. Appendix – programming C code

### 6.1 Main.c

```c
#include <stdio.h>
#include <stdint.h>
#include "PB_LCD_Drivers.h"
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include "stm32f4xx.h"
#include "utils.h"

int clear = 1;
int j = 0;
uint16_t delayCount = 0;

/* Store character into string array and display them on LCD */
char buffer[50];
char buffer_2[20];

void SysTick_Handler (void){
    clear = 1 - clear;
    delayCount++;
}

// Blue button pushed
double CheckPushbutton(){
    unsigned char check_button;

    check_button = GPIOA->IDR & 0x00000001;
    if(check_button){
        return 1;}
    else{
        return 0;}
}

// Blue button released
double CheckReleasebutton(){
    unsigned char check_button;
    check_button = GPIOA->IDR & 0x00000001;
    if(check_button != 1){
        return 1;}
    else{
        return 0;}
}

// Use switch SW8
double CheckButton(){
    unsigned char check_button;
    check_button = (GPIOE->IDR >> 8);
    if(check_button & (1<<7)){
        return 8;}
    else if(check_button & (1<<6)){
        return 7;}
    else{
        return 0;}
}

/* Initialize LEDs */
void initialiseLEDs(){
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN; // Enable clock register
    // Set PD13, PD12, PD14, PD15 to digital outputs
    GPIOD->MODER = (GPIOD->MODER & 0x55FFFFFF) | 0x55000000;
}

/* Initialize enable for analogue switches */
void initialiseEnalbe(){
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN; // Enable clock register
    // Set enable
    unsigned char enable = GPIOC->IDR & 0x00000001;
}

/* Initialize pressed buttons */
void initialiseButton(){
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // Enable clock register
    // Set the pushbutton PA0
    GPIOA->MODER = GPIOA->MODER & 0xFFFFFFFC;
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOEEN;
    // Set the pushbutton PE8 connecting with GPI0
    GPIOE->MODER = GPIOE->MODER & 0xFFFCFFFF;
}

/* Read voltage */
float ReadVoltage(int value){
    // voltage is to store the value displayed on LCD which should be input value
    // value_voltage is to store transferred binary value to decimal value
    float voltage;
    float value_voltage;

    value_voltage = ((float)value/57600) * 2.5; // Calculate voltage by (ADC value
/ ADC resolution) * Reference voltage

    /* If value_voltage is between 0-1.125 or 1.375-2.5, use gain = 0.123
        it is a large signal and it can be just multiplied by the smallest gain
0.123 */
    if((value_voltage >= 0.2 && value_voltage <= 1.125) || (value_voltage >= 1.375
&& value_voltage <= 2.5)){
        /* By linear function Vin = 8*Vread - 10, transfer it back to the input
voltage */
        voltage = 8 * value_voltage - 10; // To make the reading precise
    }
    /* Fly back to 0 volt when there is no inputs */
    else if (value_voltage < 0.2)
    {
        voltage = 0;
    }

    /* If value_voltage is between 1.125-1.375, use gain = 1 firstly and then if
the signal is still very smal, use a larger gain
```

```
        Because we regard this signal is small and we need to multiply a larger
gain to make the signal readable by ADC */
    else{
        /* Reread value from ADC */
        value = read_ADC1();    /* Gets a 12 bit right-aligned value from the ADC
*/
        value = (value << 4) & 0xFF00;  /* Shift and AND to isolate bits 15-12 */
        value_voltage = ((float)value/57600) * 2.5; // Calculate voltage by (ADC
value / ADC resolution) * Reference voltage

        /* Use the first gain=1 to get reading by linear function Vin = Vread -
1.25 */
        voltage = value_voltage - 1.25;
        /* If the signal is still so small that ADC is difficult to read we use the
second gain=10 to get reading by linear function
            Vin = 0.1*Vread - 0.125 */
        if (value_voltage >= 1.125 && value_voltage <= 1.375){
            /* Reread value from ADC */
            value = read_ADC1();    /* Gets a 12 bit right-aligned value from the
ADC */
            value = (value << 4) & 0xFF00;  /* Shift and AND to isolate bits 15-12
*/
            value_voltage = ((float)value/57600) * 2.5; // Calculate voltage by
(ADC value / ADC resolution) * Reference voltage

            /* Use the second gain=10 to get reading by linear function Vin =
0.1*Vread - 0.125 */
            voltage = 0.1 * value_voltage - 0.125;
            /* If the signal is still so small between -0.0125-0.0125 that ADC is
difficult to read we use the third gain=100 to get reading
                by linear function Vin = 0.01*Vread - 0.0125 */
            if(value_voltage >= 1.125 && value_voltage <= 1.375){
                /* Reread value from ADC */
                value = read_ADC1();    /* Gets a 12 bit right-aligned value from
the ADC */
                value = (value << 4) & 0xFF00;  /* Shift and AND to isolate bits
15-12 */
                value_voltage = ((float)value/57600) * 2.5; // Calculate voltage by
(ADC value / ADC resolution) * Reference voltage
                /* Use the third gain=100 to get reading by linear function Vin =
0.01*Vread - 0.0125 */
                voltage = 0.01 * value_voltage - 0.0125;
            }
        }
    }
    return voltage;
}

/* Read current */
float ReadCurrent(int value){
    // Current is to store the value displayed on LCD
    // value_voltage is to store transferred binary value to decimal value
    float current;
    float value_voltage;

    value_voltage = ((float)value/57600) * 2.5;  // Calculate voltage by (ADC value
/ ADC resolution) * Reference voltage
    // implement the linear function to change ADC reading back to the input
current
    // The linear function is y = 0.8x-1
    //current_scaled = value_voltage/10;
    current = 0.8*value_voltage-1;

    return current;
}

/* Read resistance */
float ReadResistance(int value){
    // resistance is to store the value displayed on LCD
    // value_voltage is to store transferred binary value to decimal value
    float resistance;
    float value_voltage;

    value_voltage = ((float)value/57600) * 2.5; // Calculate voltage by (ADC value
/ ADC resolution) * Reference voltage

    /* Switch using resistance from 1Mohm until using 500 ohms to get constant
current
    By means of potential divider: Runkonow = Rrange*Vread / 2.5 */
    // If the reading is smaller than 1.25V (1/2*2.5) then we use 490kOhms
reference resistance
    if(value_voltage <= 1.25 && value_voltage >= 0.2){
        /* Reread value from ADC */
        value = read_ADC1();    /* Gets a 12 bit right-aligned value from the ADC
*/
        value = (value << 4) & 0xFF00;  /* Shift and AND to isolate bits 15-12 */
        value_voltage = ((float)value/57600) * 2.5; // Calculate voltage by (ADC
value / ADC resolution) * Reference voltage

        resistance = 490000*value_voltage/2.5;
        // If the reading is smaller than 0.5V (1/5*2.5) then we use 100kOhms
reference resistance
        if(value_voltage <= 0.5 && value_voltage > 0.2){
            /* Reread value from ADC */
            value = read_ADC1();    /* Gets a 12 bit right-aligned value from the
ADC */
            value = (value << 4) & 0xFF00;  /* Shift and AND to isolate bits 15-12
*/
            value_voltage = ((float)value/57600) * 2.5; // Calculate voltage by
(ADC value / ADC resolution) * Reference voltage

            resistance = 100000*value_voltage/2.5;
            // If the reading is smaller than 1.25V (1/2*2.5) then we use 49k9Ohms
reference resistance
            if(value_voltage <= 1.25){
                /* Reread value from ADC */
                value = read_ADC1();    /* Gets a 12 bit right-aligned value from
the ADC */
                value = (value << 4) & 0xFF00;  /* Shift and AND to isolate bits
15-12 */
                value_voltage = ((float)value/57600) * 2.5; // Calculate voltage by
(ADC value / ADC resolution) * Reference voltage

                resistance = 49900*value_voltage/2.5;
                // If the reading is smaller than 0.5V (1/5*2.5) then we use
10kOhms reference resistance
                if(value_voltage <= 0.5){
                    /* Reread value from ADC */
                    value = read_ADC1();    /* Gets a 12 bit right-aligned value
from the ADC */
                    value = (value << 4) & 0xFF00;  /* Shift and AND to isolate
bits 15-12 */
                    value_voltage = ((float)value/57600) * 2.5; // Calculate
voltage by (ADC value / ADC resolution) * Reference voltage

                    resistance = 10000*value_voltage/2.5;
                    // If the reading is smaller than 1.25V (1/2*2.5) then we use
4k99Ohms reference resistance
                    if(value_voltage <= 1.25){
                        /* Reread value from ADC */
                        value = read_ADC1();    /* Gets a 12 bit right-aligned
value from the ADC */
                        value = (value << 4) & 0xFF00;  /* Shift and AND to isolate
bits 15-12 */
                        value_voltage = ((float)value/57600) * 2.5; // Calculate
voltage by (ADC value / ADC resolution) * Reference voltage
```

```
                        resistance = 4990*value_voltage/2.5;
                        // If the reading is smaller than 0.5V (1/5*2.5) then we
use 1kOhms reference resistance
                        if(value_voltage <= 0.5){
                            /* Reread value from ADC */
                            value = read_ADC1();    /* Gets a 12 bit right-aligned
value from the ADC */
                            value = (value << 4) & 0xFF00;  /* Shift and AND to
isolate bits 15-12 */
                            value_voltage = ((float)value/57600) * 2.5; //
Calculate voltage by (ADC value / ADC resolution) * Reference voltage

                            resistance = 1000*value_voltage/2.5;
                            // If the reading is smaller than 1.25V (1/2*2.5) then
we use 500Ohms reference resistance
                            if(value_voltage <= 1.25){
                                /* Reread value from ADC */
                                value = read_ADC1();    /* Gets a 12 bit right-
aligned value from the ADC */
                                value = (value << 4) & 0xFF00;  /* Shift and AND to
isolate bits 15-12 */
                                value_voltage = ((float)value/57600) * 2.5; //
Calculate voltage by (ADC value / ADC resolution) * Reference voltage

                                resistance = 500*value_voltage/2.5;
                            }
                        }
                    }
                }
            }
        }
    }
    // Fly back to 0 when there is no input
    else if (value_voltage <= 0.2)
    {resistance = 0;}

    // If the reading is larger than (1/2*2.5) then we use 1Mohms reference
resistance
    else{
        // Use 1Mohm resistor to get initial prediction of resistance
        resistance = 1000000*value_voltage/2.5;
    }

    return resistance;
}


//PC6 PWM INPUT (EXT_PC6 on motherboard)
//TIM3 CH1 Reset at rising edge
//TIM3 CC1R Save at rising edge
//50Hz to 2Mhz
//tolerance 0.5us
//Data send to UART2(PA2 Tx)
//115200bps

/* Read frequency */
float ReadFrequency(){
    // period of the signal
    uint32_t p=0;
    p=TIM_GetCapture1(TIM3);
    p++;

    // As the frequency of TIM3 is 2MHz so that the maximum frequency we can
measure is 2MHz
    // As the tolerance is 0.5us and we want it to transfer to kHz to represent
    // and by means of relationship of frequency and period, we got
functionf=1/(p*0.0005)
```

```
    return (1.0f/((float)p*0.0005f));
}

/* Switch cases among testing voltages, current and resistance */
void Mode(int i, int value){
    float valueDisplayed;
    float Data_Array[10];

    switch(i){
        // Cases are used to select mode
        case 0:
            // Auto-range
            if(ReadVoltage(value) >= 1 || ReadVoltage(value) <= -1){
                // Display the title
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("Volt: -10V-10V", 14);
                // Display value calculated by ReadVoltage()
                valueDisplayed = ReadVoltage(value);
                Data_Array[0] = ReadVoltage(value);
                // Display the value on LCD
                sprintf(buffer_2, "%0.5f V", valueDisplayed);
            }
            else if((ReadVoltage(value) >= 0.1 && ReadVoltage(value) < 1) ||
(ReadVoltage(value) <= -0.1 && ReadVoltage(value) > -1)){
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("Volt: -1V-1V", 12);
                valueDisplayed = ReadVoltage(value);
                Data_Array[0] = ReadVoltage(value);
                sprintf(buffer_2, "%0.5f V", valueDisplayed);
            }
            else if((ReadVoltage(value) >= 0.01 && ReadVoltage(value) < 0.1) ||
(ReadVoltage(value) <= -0.01 && ReadVoltage(value) > -0.1)){
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("Vol:-100mV-100mV", 16);
                valueDisplayed = ReadVoltage(value);
                Data_Array[0] = ReadVoltage(value);
                sprintf(buffer_2, "%0.5f mV", valueDisplayed*1000);
            }
            else{
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("Vol: -10mV-10mV", 15);
                valueDisplayed = ReadVoltage(value);
                Data_Array[0] = ReadVoltage(value);
                sprintf(buffer_2, "%0.5f mV", valueDisplayed*1000);
            }
            break;

        case 1:
            // Auto-range
            if((ReadCurrent(value) >= -1 && ReadCurrent(value) <= -0.1) ||
(ReadCurrent(value) <= 1 && ReadCurrent(value) >= 0.1)){
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("Amp:-1->1A", 10);
                valueDisplayed = ReadCurrent(value);
                Data_Array[1] = ReadCurrent(value);
                sprintf(buffer_2, "%0.3f A", valueDisplayed);
            }
            else if((ReadCurrent(value) >= -0.1 && ReadCurrent(value) <= -0.01) ||
(ReadCurrent(value) <= 0.1 && ReadCurrent(value) >= 0.01)){
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("Amp:-100m->100mA", 16);
                valueDisplayed = ReadCurrent(value);
                Data_Array[1] = ReadCurrent(value);
                sprintf(buffer_2, "%0.3f mA", valueDisplayed*1000);
            }
            else{
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("Amp:-10m->10mA", 14);
                valueDisplayed = ReadCurrent(value);
                Data_Array[1] = ReadCurrent(value);
                sprintf(buffer_2, "%0.3f mA", valueDisplayed*1000);
            }
            break;

        case 2:
            // Auto-range
            if(ReadResistance(value) >= 100000){
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("R: 1M", 5);
                valueDisplayed = ReadResistance(value);
                Data_Array[2] = ReadResistance(value);
                sprintf(buffer_2, "%0.3f MOhm", valueDisplayed/1000000);
            }
            else if(ReadResistance(value) >= 1000 && ReadResistance(value) <
1000000){
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("R: 1k->9.99k", 12);
                valueDisplayed = ReadResistance(value);
                Data_Array[2] = ReadResistance(value);
                sprintf(buffer_2, "%0.3f kOhm", valueDisplayed/1000);
            }
            else if(ReadResistance(value) >= 1 && ReadResistance(value) < 1000){
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("R: 1->999", 9);
                valueDisplayed = ReadResistance(value);
                Data_Array[2] = ReadResistance(value);
                sprintf(buffer_2, "%0.3f Ohm", valueDisplayed);
            }
            else{
                PB_LCD_GoToXY(0,0);
                PB_LCD_WriteString("Resistance: ", 12);
                valueDisplayed = ReadResistance(value);
                Data_Array[2] = ReadResistance(value);
                sprintf(buffer_2, "%0.3f Ohm", valueDisplayed);
            }
            break;

        case 3:
            PB_LCD_GoToXY(0,0);
            PB_LCD_WriteString("Frequency :", 11);
            valueDisplayed = ReadFrequency();
            sprintf(buffer_2, "%4.3f kHz", valueDisplayed);
          Data_Array[3] = ReadFrequency();
            break;

        /* data storage */
        // Store the last value tested in multimeter
        case 4:
            // Display the title
            PB_LCD_GoToXY(0,0);
            PB_LCD_WriteString("Data stored: ", 12);

            if(CheckButton() == 8){
                j++;
                // Limit modes below 4
                    if(j>3){
                        j=0;}
            }

            // Display the data stored
            valueDisplayed = Data_Array[j];
                switch(j){
                    case 0:
                        sprintf(buffer_2, "%0.3f V", valueDisplayed);
                        break;
```

```
                        case 1:
                            sprintf(buffer_2, "%0.3f A", valueDisplayed);
                            break;

                        case 2:
                            sprintf(buffer_2, "%0.3f Ohm", valueDisplayed);
                            break;

                        case 3:
                            sprintf(buffer_2, "%4.3f kHz", valueDisplayed);
                            break;
                        default:
                        PB_LCD_Clear();
                    }
                break;
            default:
                PB_LCD_Clear();
        }
}
int main(void) {
    int i=0;
    SystemCoreClockUpdate();
    SysTick_Handler();
    SysTick_Config(SystemCoreClock/16);

    int value;
    //initialization
    HwInit();
    PB_LCD_Init();
    PB_LCD_Clear();
    initialiseButton();
    initialiseLEDs();
    TIM_Cmd(TIM3,ENABLE);

    while(1)
    {
        value = read_ADC1();    /* Gets a 12 bit right-aligned value from the ADC
*/
        value = (value << 4) & 0xFF00;  /* Shift and AND to isolate bits 15-8 */
        GPIOD->ODR = value | (GPIOD->ODR & 0x0000); /* write to LEDs */

        // Generate an interrupt
        uint32_t currentTimerCount;
        while(delayCount - currentTimerCount < 2) {}
        currentTimerCount = delayCount;
        // Clear the screen
        PB_LCD_Clear();

        // Check if button is pressed to change the mode
        while(CheckPushbutton() == 1){
            PB_LCD_Clear();
            // Check if button is released to change the mode
            if(CheckReleasebutton() == 1){
                i++;
                // Limit modes below 4
                if(i>4){
                    i=0;}
            }
        }
        // Change the mode for measurement
        Mode(i, value);
        PB_LCD_GoToXY(0,1);
        PB_LCD_WriteString(buffer_2, 15);
    }
}
```

## 6.2 utils.c

```
#include "stm32f4xx.h"
#include "stm32f4_discovery.h"
#include "utils.h"

#include <stdarg.h>
#include <ctype.h>


//initialize STM32F407 Hardware
void HwInit( void ) {
  SystemCoreClockUpdate();
    //TIMER3 clock = APB1 Timer(16M) /PSC(8)=2M
    initPeriphClock();
    initIO();
    initTIM3();
    vUSART2_Init();// Start up UART2
}

//initialize peripheral clock
void initPeriphClock(void)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOC,ENABLE);

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3|RCC_APB1Periph_USART2,ENABLE);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1,ENABLE);
}

//initialize ALL GPIO in use
void initIO()
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    GPIO_StructInit(&GPIO_InitStructure);
    //TIM3
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init( GPIOC, &GPIO_InitStructure );
  GPIO_PinAFConfig( GPIOC, GPIO_PinSource6, GPIO_AF_TIM3 );
    //ADC1
    GPIO_StructInit(&GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init( GPIOC, &GPIO_InitStructure );

    ADC1->CR1 |= ADC_CR1_DISCEN;//discontinuous conversion mode
    ADC1->CR2 = 0x00;   //disables DMA and makes no external triggers/injected
channels

    ADC1->SQR1 &= ADC_SQR1_L;//one conversion at a time
    ADC1->SQR3 = 14 & ADC_SQR3_SQ1;//the first conversion to come from channel 14
(connected to PC4)

    ADC1->CR2 |= ADC_CR2_ADON;//Enable the ADC
}


//initialize TIM3
void initTIM3( void )
{
        TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
        TIM_ICInitTypeDef  TIM_ICInitStructure;

        //TIM_Cmd(TIM3,DISABLE);
        TIM_TimeBaseStructure.TIM_RepetitionCounter=0;//no use
        TIM_TimeBaseStructure.TIM_Prescaler=8-1;//Prescaler /8 CNT_CLK=2Mhz
        TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;
        TIM_TimeBaseStructure.TIM_Period=0x0FFFF;
        TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;
        TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

        //TIM_ARRPreloadConfig( TIM3, ENABLE );
        TIM_InternalClockConfig(TIM3);
        TIM_SelectInputTrigger(TIM3,TIM_TS_TI1FP1);
        TIM_SelectSlaveMode(TIM3,TIM_SlaveMode_Reset);
        TIM_SelectMasterSlaveMode(TIM3,TIM_MasterSlaveMode_Disable);

        //---
        CLEAR_BIT(TIM3->DIER, TIM_DIER_TIE);//DisableIT_TRIG
        CLEAR_BIT(TIM3->DIER, TIM_DIER_TDE);//DisableDMAReq_TRIG
        MODIFY_REG(TIM3->CR2, TIM_CR2_MMS, 0x00000000U);//TRGO_RESET

        TIM_ICInitStructure.TIM_Channel=TIM_Channel_1;
        TIM_ICInitStructure.TIM_ICFilter=0x0;
        TIM_ICInitStructure.TIM_ICPolarity=TIM_ICPolarity_Rising;
        TIM_ICInitStructure.TIM_ICPrescaler=TIM_ICPSC_DIV1;
        TIM_ICInitStructure.TIM_ICSelection=TIM_ICSelection_DirectTI;
        TIM_ICInit( TIM3, &TIM_ICInitStructure );
}

//initialize USART2
void vUSART2_Init( void ) {
    USART_ClockInitTypeDef USART_ClockInitStruct;
    USART_InitTypeDef USART_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    // Make sure you use 'GPIO_PinSource2' and NOT 'GPIO_Pin_2'.  Using the
    // latter will not work!
    GPIO_PinAFConfig( GPIOA, GPIO_PinSource2, GPIO_AF_USART2 );
    GPIO_PinAFConfig( GPIOA, GPIO_PinSource3, GPIO_AF_USART2 );

    // Setup Tx / Rx pins.
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;           // Tx Pin
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_Init( GPIOA, &GPIO_InitStructure );
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;           // Rx Pin
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init( GPIOA, &GPIO_InitStructure );

    // Make sure syncro clock is turned off.
    USART_ClockStructInit( &USART_ClockInitStruct );
    USART_ClockInit( USART2, &USART_ClockInitStruct  );

    // Setup transmit complete irq.
    USART_ITConfig( USART2, USART_IT_TC, ENABLE );

    // Use defaults (except baud rate).
    USART_StructInit( &USART_InitStructure );
    USART_InitStructure.USART_BaudRate = 115200;
    USART_Init( USART2, &USART_InitStructure );
    USART_Cmd( USART2, ENABLE );
}

/* Read ADC and return values */
unsigned int read_ADC1(void){
    /* set SWSTART to 1 to start conversion */
    ADC1->CR2 |= ADC_CR2_SWSTART;
    /* Wait until end-of-conversion bit goes high */
    while(!(ADC1->SR & ADC_SR_EOC)){}
    /* Return data value */
    return (ADC1->DR);
}
```

## 6.3 utils.h

```
#include "stm32f4xx.h"

void HwInit( void );
void vUSART2_Init( void );

void initTIM3( void );
void initPeriphClock(void);
void initIO(void);
unsigned int read_ADC1(void);
```