Tutorial:

The STM32F411 Discovery Kit

By Le Diem Tho

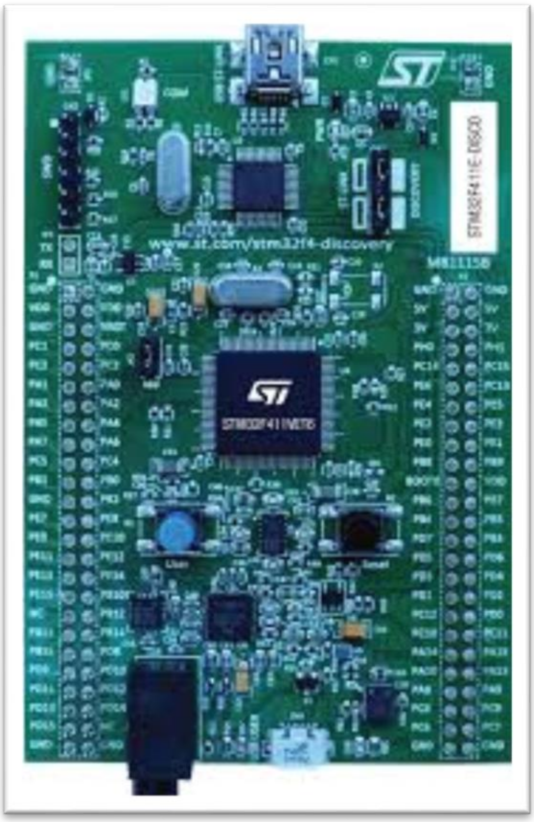

Table of content

Chapter I: Introduction

1.1. Objectives

After completing this lab, you will able to:

- Understanding for STM32F4 microcontrollers
- Creating basic projects
- Communication through USB com virtual port

1.2. Pre-lab Requirement

1.2.1. Installing Keil MDK:

Download and install the Keil-Lite MDK: http://www.keil.com/mdk5/editions/lite/

After installing, a shortcut of "Keil uVision" is created in the desktop.

Figure 1: Keil uVision icon.

"MDK-Lite is intended for product evaluation, small projects, and the educational market. It is restricted to 32 kbyte code size."

❖ **Install necessary packs:**

- "After the MDK Core installation is complete, the Pack Installer is started automatically, which allows you to add supplementary Software Packs."
- "As a minimum, you need to install a Software Pack that supports your target microcontroller device."
- If the Pack Installer is not started or you want to open it again, then open "Keil uVision", From the menu, select "Project → Manage → Pack Installer…" to open the "Pack Installer".

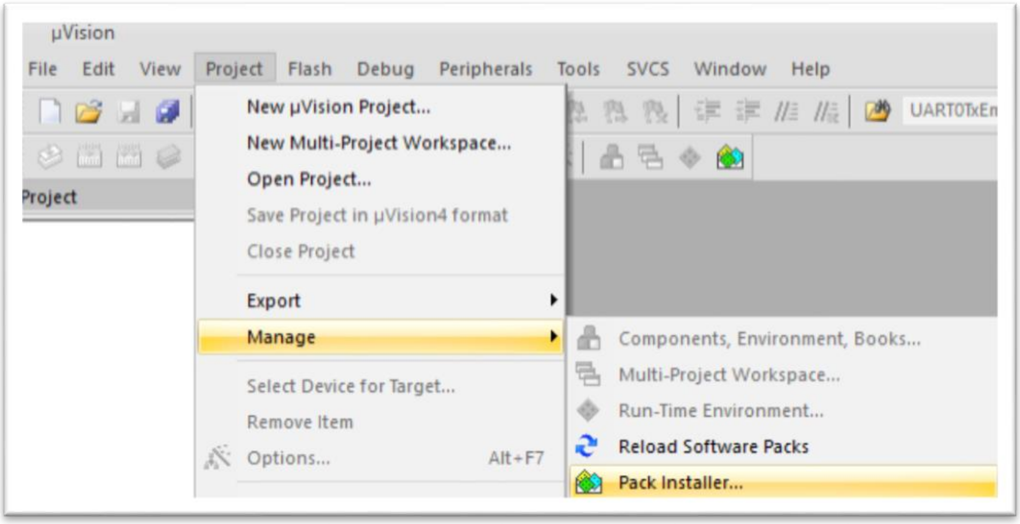

Figure 2: Open Pack Installer from Keil uVision menu.

- Now, in the Pack Installer, search for "STM32F4 series" and select it. In the "Pack" panel, and press "Install" button in the "Action" tab to install this microcontroller series.

1.2.2. Learn about the STM32F4 Discovery Kit

Figure 3. Discovery kit with STM32F411VE MCU

The STM32F411VET microcontrollers is the kind of the STM32 Dynamic Efficiency™ lines. The Discovery kit for STM32F411 helps research and create the project with efficiency. It provides many features for beginners and experienced users. It runs at 100 MHz with low energy consumption values in running and stopping moment. Moreover, the Discovery kit integrates sensors and communication ports. The STM32F411 Discovery board includes the following features:

The STM32F411VET6 microcontroller has Flash memory (512 KB) and RAM (128 KB)

- The power supply is 5 V for the board, 3 V and 5 V for the external.
- ST-LINK/V2 is available on the board.
- L3GD20: It has the 3-axis digital sensor to measure gyroscope.
- LSM303DLHC: ST MEMS has 3D digital linear sensor for measuring acceleration and magnetic.
- There are 8 LEDs with features: USB communication, power on, user, USB OTG.
- There are 2 push buttons (for user and reset) and USB OTG with micro-AB connector.

1.2.3. Installing CubeMX software

The STM32Cube includes two components: STM32Cube MX and STM32Cube HAL. The STM32Cube MX is a graphical tool that supports to configuration of STM32 microcontrollers and create the corresponding initial C code. The STM32Cube HAL is an abstraction layer embedded software. The driver layer with an application programming interfaces (API) supports interacting with the upper layer (Application, Libraries and Stacks). The source code of HAL drivers is developed in Strict ANSI-C.

Figure 4. The STM32Cube MX

1.3. In-lab Requirement

1.3.1. Pinout

The graphic simulation for STM32F411VET microcontroller supports setting pinouts and the peripheral or internal connection.

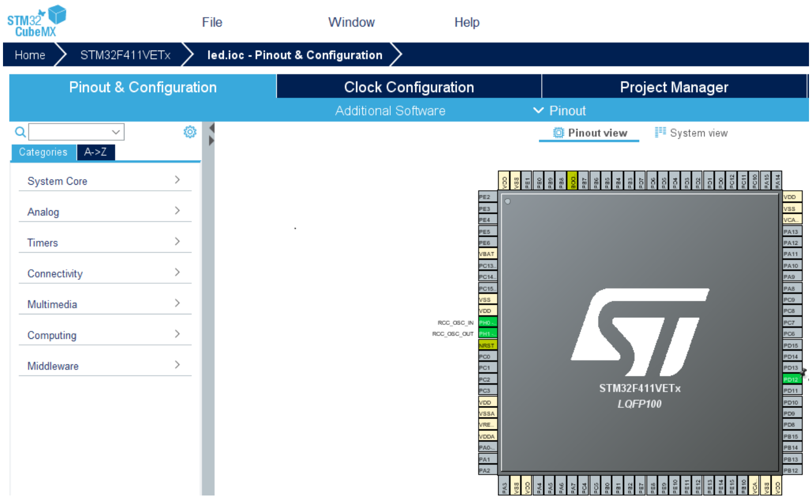

Figure 5. The graphic simulation in STM32CubeMX

In this part, to create the project and configure:

- ✓ Clock configuration
- ✓ Pinout configuration
- ✓ Setting up Timer, PWM,...
- ✓ SPI, I2C, Uart,..interfaces
- ✓ HAL Libraries

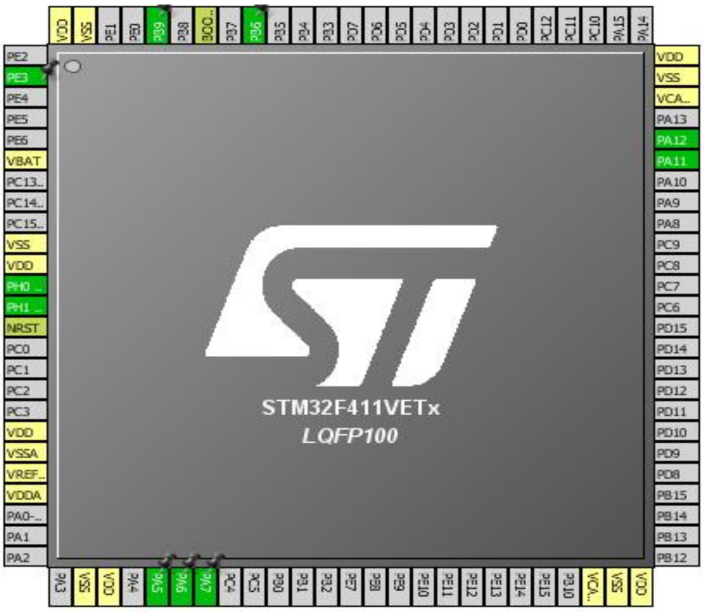

Figure 6. Choosing Pinouts

1.3.2. Programming in Keil MDK

After creating the project, open the generation code by Keil C and start writing commands

main.c*

```c
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"

/* Private variables ---------------------------------------------------------*/

/* Private variables ---------------------------------------------------------*/

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);

/* Private function prototypes -----------------------------------------------*/

int main(void)
{
  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  while (1)
  {
    /* loop Code */
  }

}
```

Figure 7. Generation Code

❖ Learn more about the HAL library:

File	Description
stm32f4xx_hal_ppp.c/.h	peripheral driver with portable APIs
stm32f4xx_hal_ppp_ex.c/.h	extended peripheral features APIs
stm32f4xx_hal.c	contains HAL common APIs (HAL_Init, HAL_DeInit, HAL_Delay,…)
stm32f4xx_hal.h	HAL header file, it should be included in user code
stm32f4xx_hal_conf.h	config file for HAL, should be customized by user to select the peripherals to be included
stm32f4xx_hal_def.h	contains HAL common typedefs and macros

Figure 8. HAL file components

1.3.3. Build project:

- Build project: To build the project for the target (LPC1768 microcontroller), from the menu of Keil uVision, select *"Project → Build target"*.

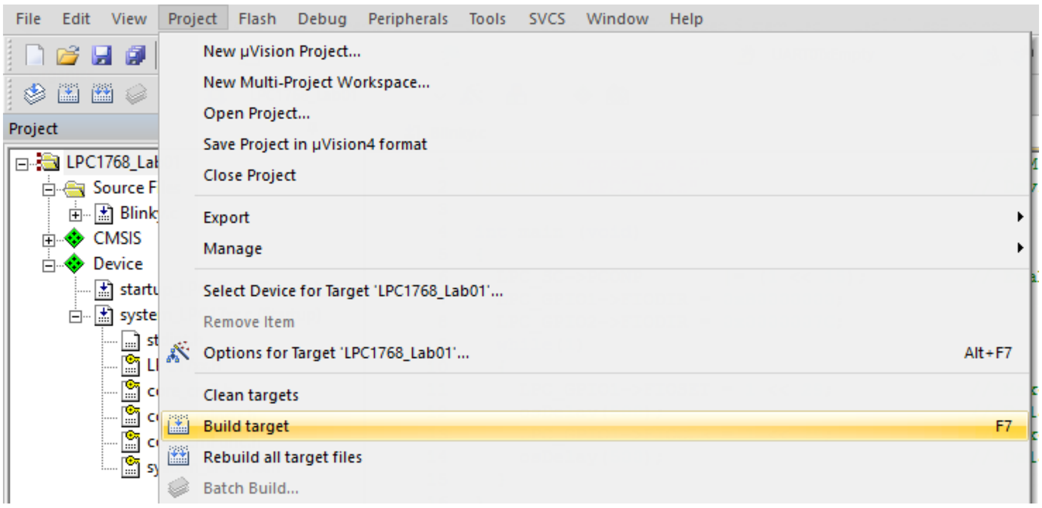

Figure 9. Build project for the target

- Download to the board: To download the project for the target, from the menu of Keil uVision, select *"Flash → Build Download"*.

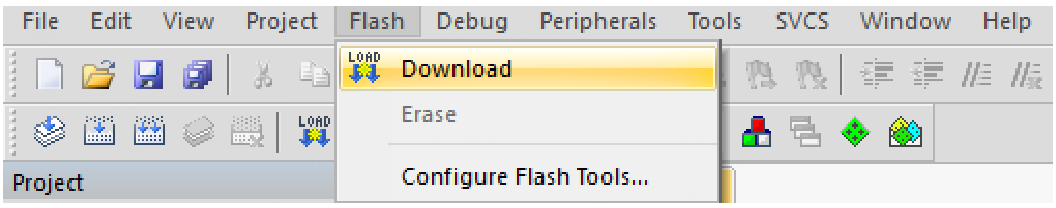

Figure 10. Flash the program into the controller

Chapter II: Projects

2.1. GPIOs and LEDs

- **Topic: Controls LEDs that are soldered on the board**
- Open CubeMX and choosing GPIO output for PD12, PD13, PD14, PD15

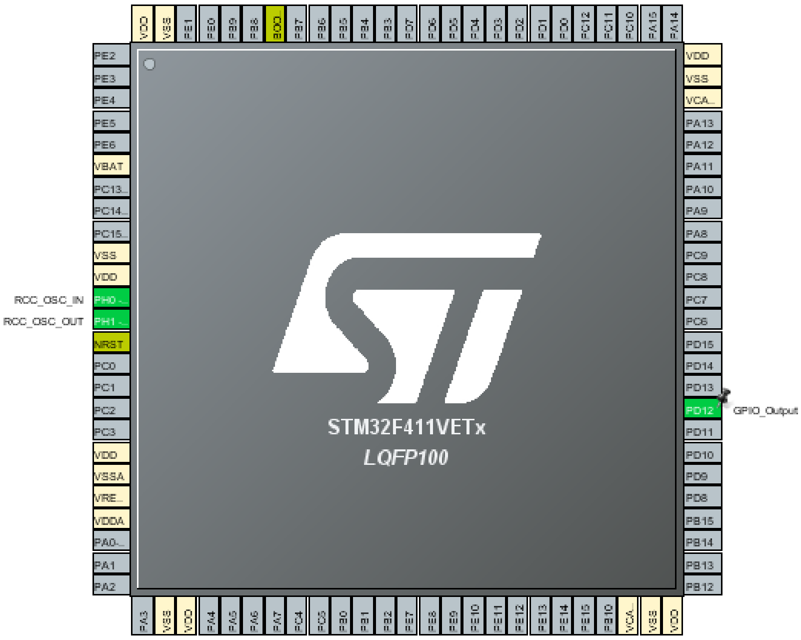

Figure 11: Choosing in Ex.1

- Configure clock and pinouts as follows

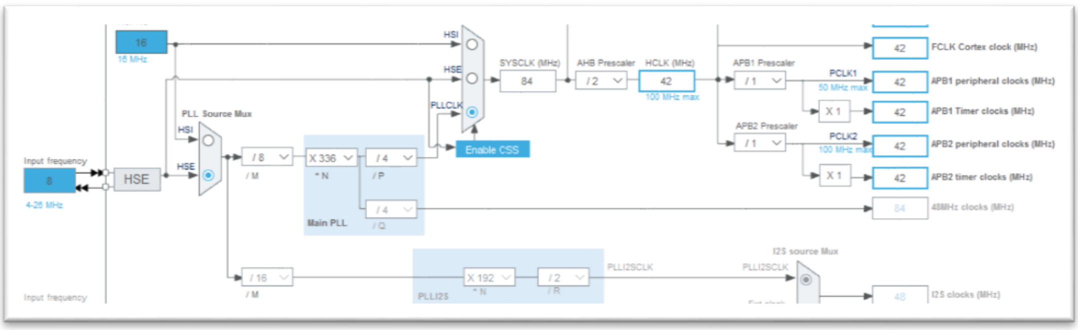

Figure 12. Configure Clock

The RCC parameters is configured as follows:

- The high-speed internal (I) calibration value is 16 MHz
- The high-speed external (HSE) Startup timeout value (ms) is 100
- The low-speed external (LSE) Startup timeout value (ms) is 5000
- Main phase-locked loop (PLL) clock with division factors: M=8, N=336, and P=4. The set clock speed (SYSCLK) is calculated as:

$$\text{SYSCLK} = \frac{\frac{HSE}{M}N}{P} = \frac{\frac{8}{8}336}{4} = 84\ (MHz)$$

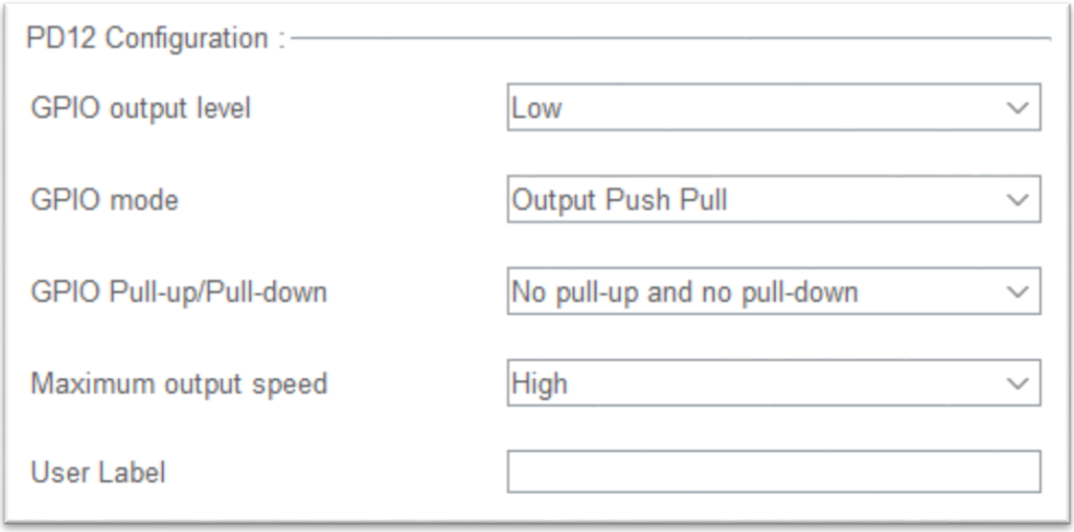

Figure 13. The Pinout configuration

- Setting up the project manager

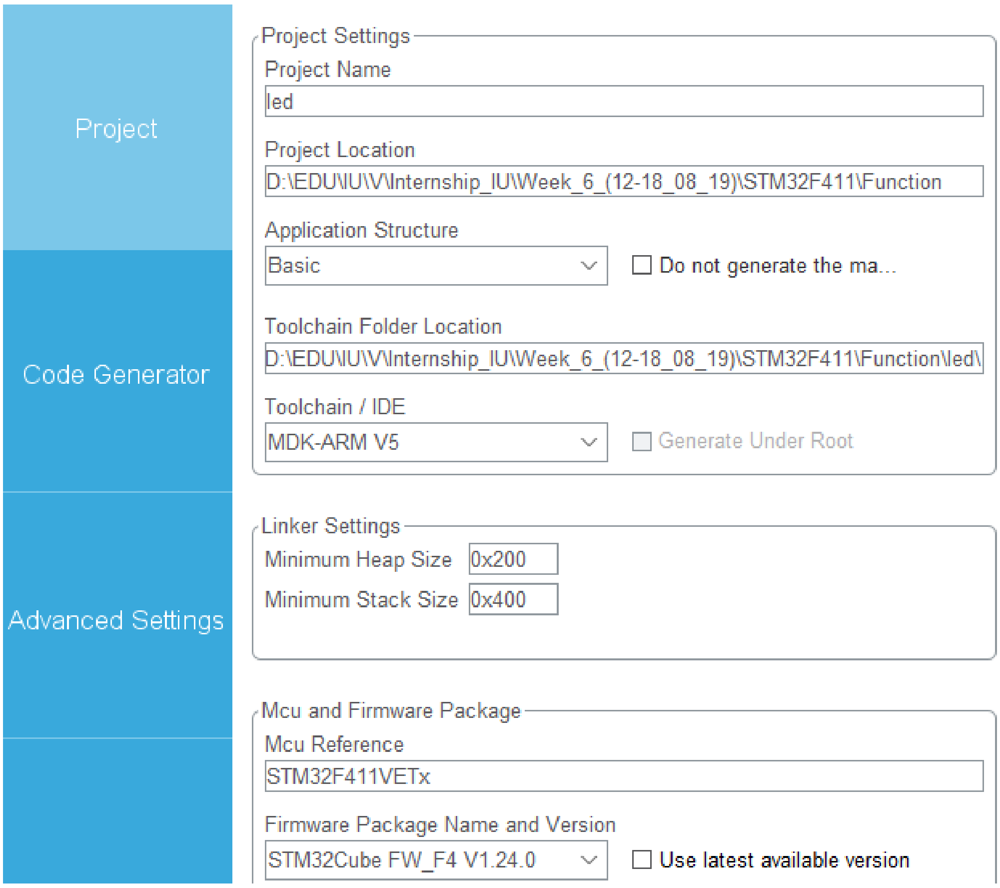

Figure 14. Code Generator

- Writing commands in Keil C

```c
int main(void)
{
  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  while (1)
  {
  HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12|GPIO_PIN_14|GPIO_PIN_13|GPIO_PIN_15);
  HAL_Delay(500);
  }

}
```

Figure 15. For blinking LEDs

- Download to the kit and the result:

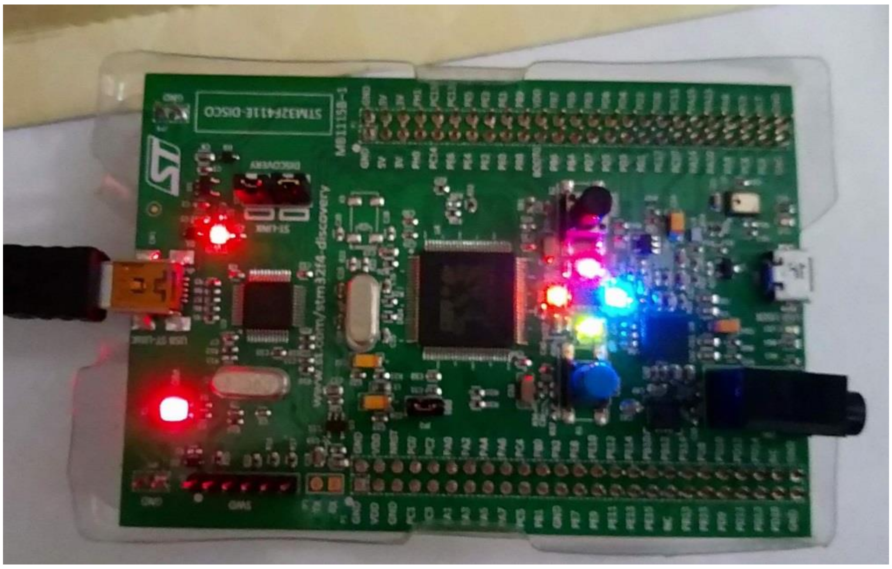

Figure 16. The result of Ex. 1

2.2. Pulse width modulation (PWM)

- **Topic: Using PWM to adjust LEDs**
- Open CubeMX and configure PD12, Pd13, PD14, PD15 as follow as

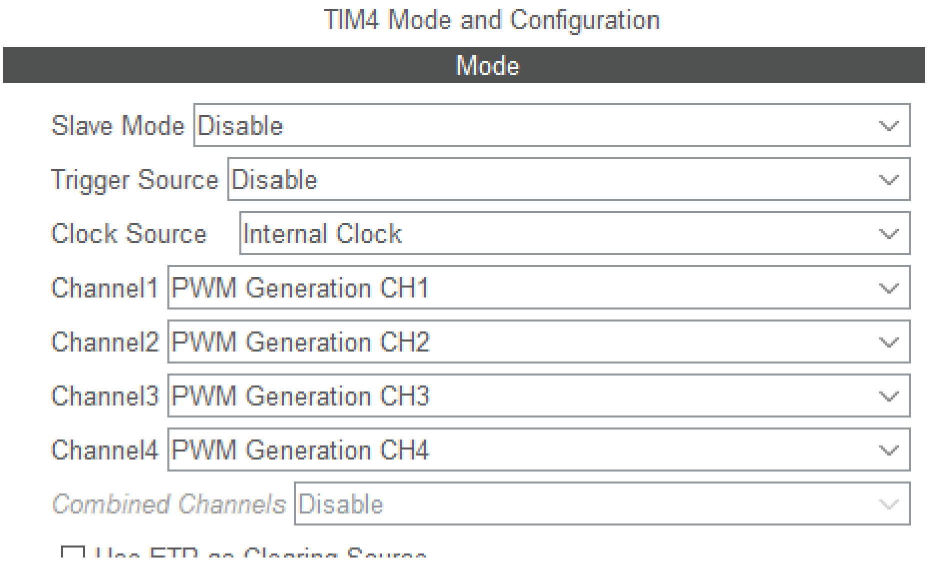

Figure 17. Choosing Timer 4

- Configure clock and pinouts as follows

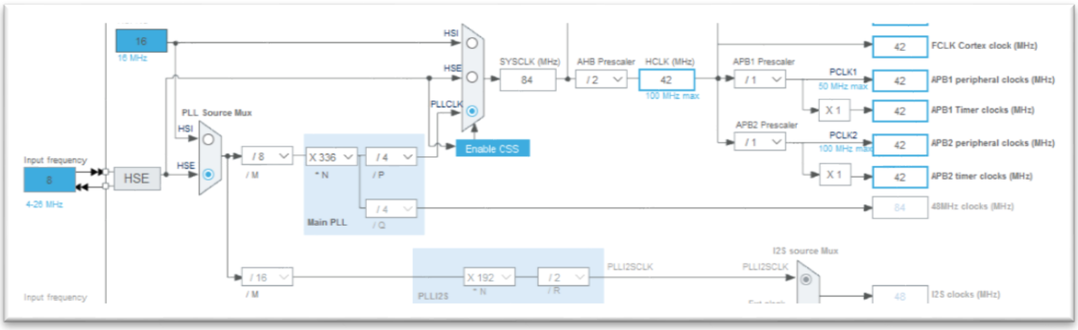

Figure 18. Configure Clock

The RCC parameters is configured as follows:

  - The high-speed internal (I) calibration value is 16 MHz
  - The high-speed external (HSE) Startup timeout value (ms) is 100
  - The low-speed external (LSE) Startup timeout value (ms) is 5000
  - Main phase-locked loop (PLL) clock with division factors: M=8, N=336, and P=4. The set clock speed (SYSCLK) is calculated as:

$$\text{SYSCLK} = \frac{\frac{HSE}{M}N}{P} = \frac{\frac{8}{8}336}{4} = 84\ (MHz)$$

- Setting on TIM4

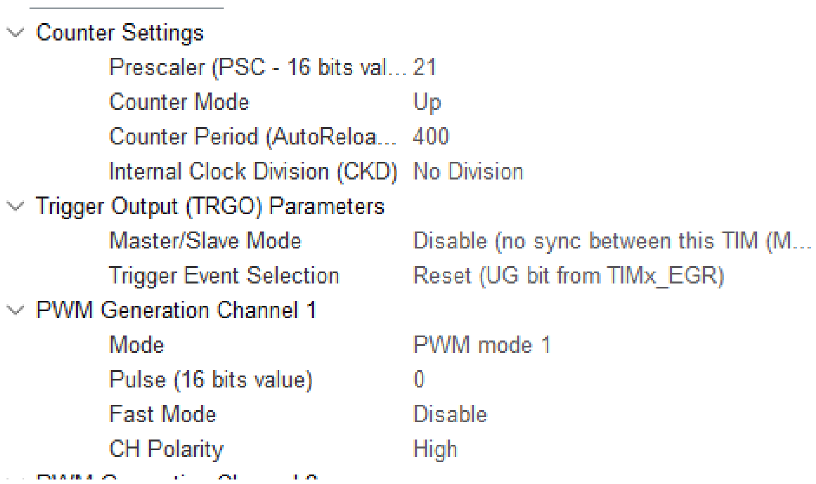

Figure 19. Setting PWM

- Setting up the project manager as Ex. 1
- Writing commands in Keil C

```c
MX_TIM4_Init();
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_2);
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_4);

while (1)
{
__HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_1,duty);
__HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_2,duty);
__HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3,duty);
__HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_4,duty);

duty+=fade;

if((duty==0)||(duty==400))
  {
  fade=-fade;
  }
  HAL_Delay(50);
}

}
```

Figure 20. For adjusting LEDs

- Download to the kit and the result:

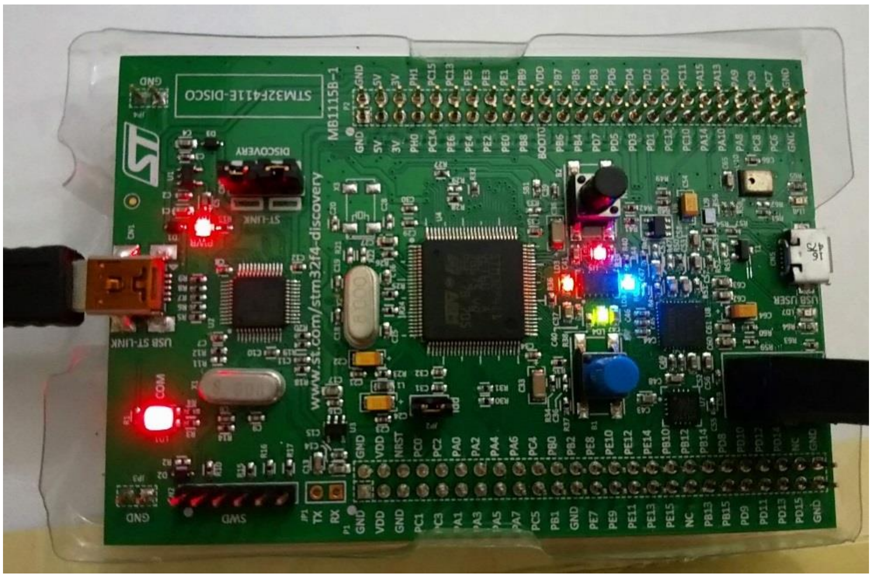

Figure 21. The result of Ex. 2

2.3. External Interrupt

- **Topic: Button and Internal Interrupt**
- Open CubeMX and configure PD12 with GPIO output and Button with GPIO input as following the datasheet.

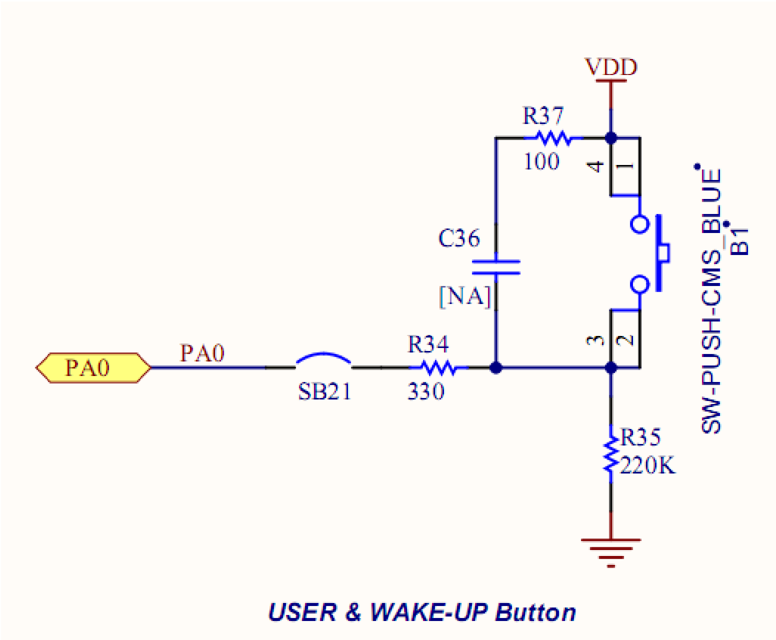

Figure 22. The button diagram

- To configure clock and set up the project manager as Ex. 2
- Writing commands in Keil C

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
 {
    count++;
    HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
    while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0));

    }
```

Figure 23. For external interrupt

- Download to the kit and the result:

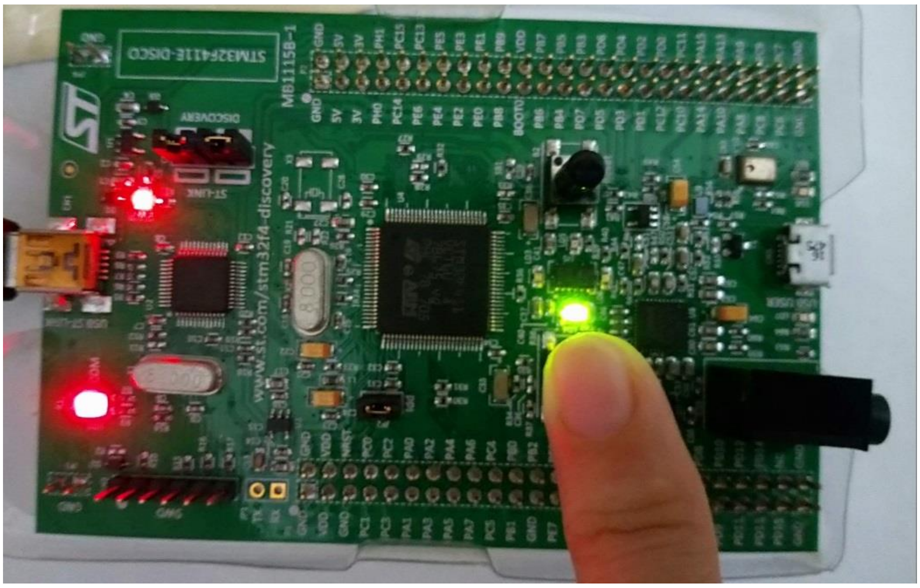

Figure 24. The result of Ex. 3

2.4. Count Edge

- **Topic: Counting up to 10 to turn on LED**
- Open CubeMX and configure PD15 with GPIO output and Button with timer as following the sample code:
- To configure clock and set up the project manager as Ex. 3
- Writing commands in Keil C

```
  while (1)
  {
  count_value=__HAL_TIM_GET_COUNTER(&htim2);
  }
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
  {
  if(htim->Instance==htim2.Instance)
  {
  HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_15);
  }
}
```

Figure 25. For counting value

- Download to the kit and the result:

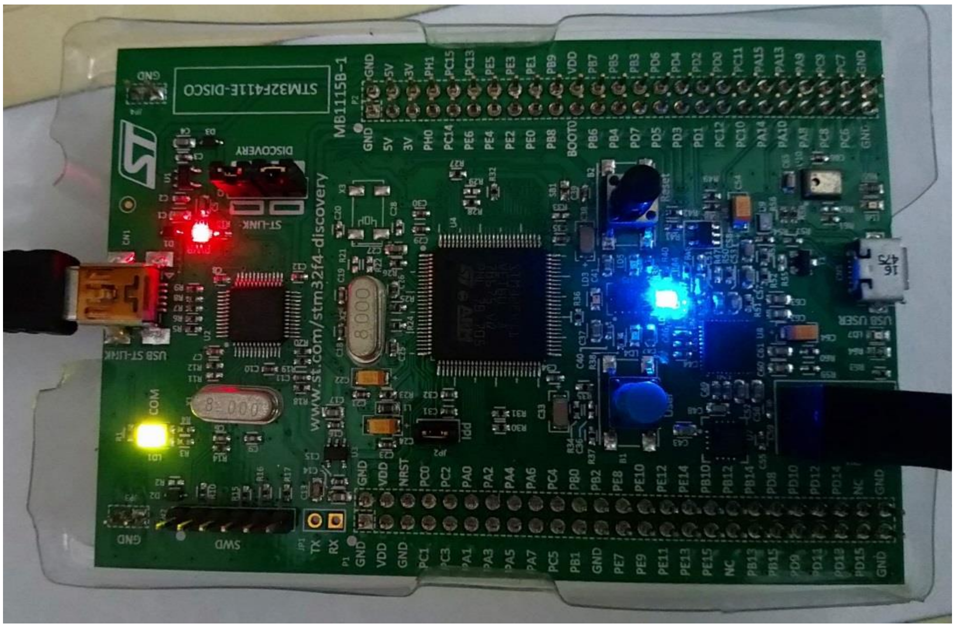

Figure 25. The result of Ex. 4

- Using the Debug feature to see the active status. Choosing “Debug” and adding the value on **watch.** Downloading to the board and click **“Run”:**

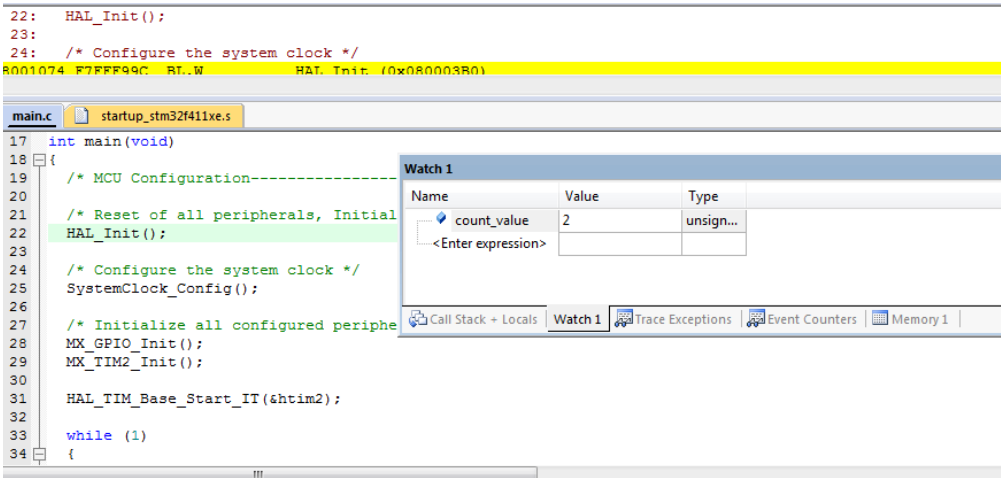

Figure 26. The Debug

2.5. LCD

- **Topic**: To display text line in LCD 16x2
- Connecting the LCD with STM32F4 as follow as:

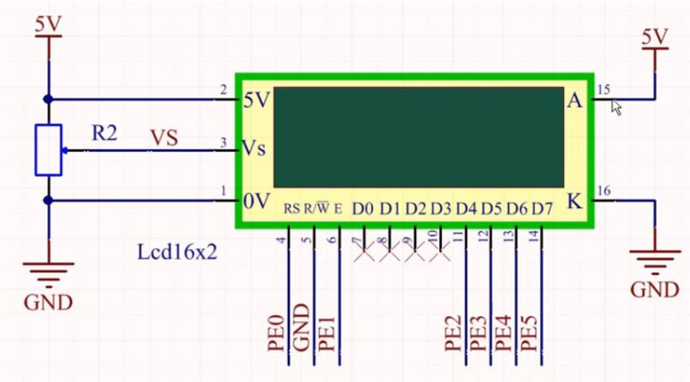

Figure 27. The diagram of Ex. 5

- Similar to the previous examples: configure PE0, PE1, PE2, PE3, PE4, PE5 as GPIO output and add the LCD library (It was attached to this project).
- Writing on Keil C

```
  lcd_init();
  lcd_puts(0,0,(int8_t*)"Have a nice day!");
//  sprintf(buffer,"%0.3f",var);
//  lcd_puts(1,0,(int8_t*)buffer);
//  HAL_Delay(2000);
//  lcd_clear();
```

Figure 28. For LCD module

- Download to the kit and the result:

Figure 29. The result of Ex. 5

2.6. USB Virtual COM Port

- **Topic**: Transmit the date through USB virtual COM port
- Create the project with configuring as follow sample code
- Adding the library of virtual COM port. (https://www.st.com)
- Writing on Keil C

```c
  while (1)
  {
    //CDC_Transmit_FS(data,18);
    CDC_Transmit_FS(mystring,16);
    HAL_Delay(1000);

  }

}
```

Figure 30. For USB virtual COM port

- To Check the data by Debug and open Terminal

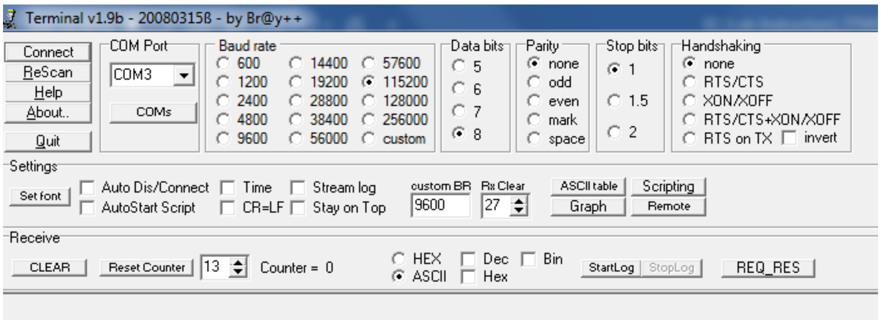

Figure 31. Terminal v1.9b

- Connection and the result

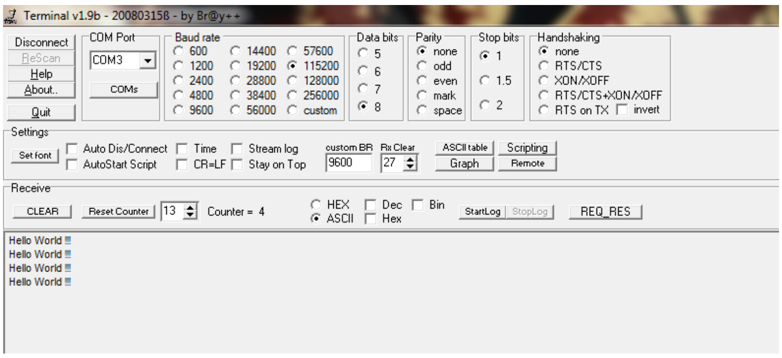

Figure 32. The result of Ex. 6

2.7. Gyroscope

- **Topic**: Reading the sensor, calculating resolution and display on Terminal
- Setting pinouts and the peripheral or internal connection.

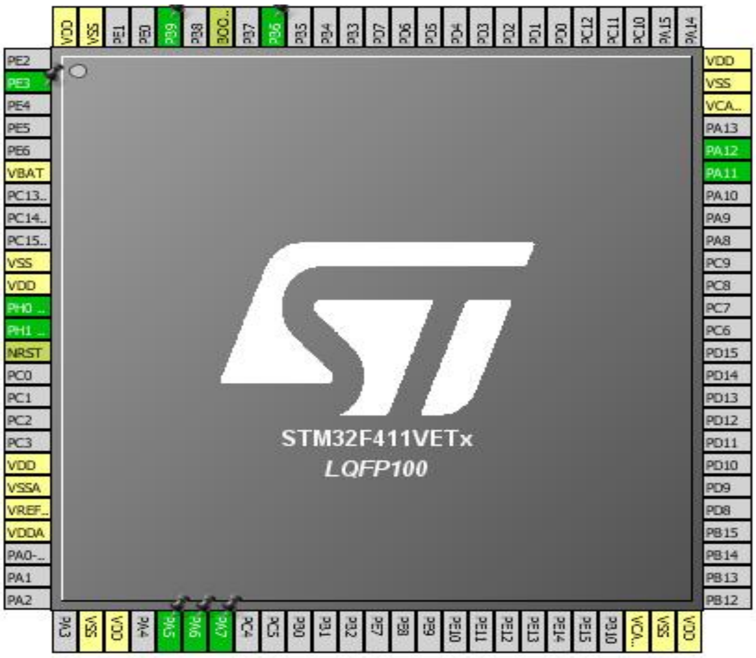

Figure 33. The Ex. 6

Table 1. Pinout configuration

Configuration	Pinout
High Speed Clock	PH0, PH1
SPI (SCK, CS, MISO, MOSI)	PA5, PE3, PA6, PA7
I2C (SDA, SCL)	PB9, PB6
USB_OTG_FS	PA11, PA12

- The communication protocol includes SPI, I2C and USB OTG. Configurations are used to read the data from sensors and transmit information to computer.
- The SPI configuration connects gyroscope sensor. Parameter Settings following table:

Basic Parameters	
Frame Format	Motorola
Data Size	8 Bits
First Bit	MSB First
Clock Parameters	
Prescaler (for Baud Rate)	8
Baud Rate	5.25 MBits/s
Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge
Advanced Parameters	
CRC Calculation	Disabled
NSS Signal Type	Software

Figure 34. Pinout configuration

- The SPI initialization function is written as:

```
static void MX_SPI1_Init(void)
{
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Figure 35. SPI initialization

- I2C configuration connects accelerometer sensor. For Master, the speed mode is standard mode and the clock speed is 100000 Hz. For Slave, the primary address length section is 7-bit.

```c
static void MX_I2C1_Init(void)
{
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Figure 36. I2C initialization function

- USB OTG is used as virtual serial port (or virtual COM port) to connect serial device port (COM port). It is configured with the 12 Mbit/s speed and 64 Bytes max packet size.

```c
#define CDC_DATA_HS_MAX_PACKET_SIZE                    64
#define CDC_DATA_FS_MAX_PACKET_SIZE                    64
#define CDC_CMD_PACKET_SIZE                            8
```

Figure 37. The input and output packet size

- The timer is used to read the data and transmit, it is as the breath of this senior project. The timer speed is calculated by Set Clock Speed (SYSCLK), Prescaler (PSC) and Counter Period (CP).

$$\text{The frequency:} \quad \mathrm{f} = \frac{\mathrm{SYSCLK}}{\mathrm{PSC.CP}} = \frac{84}{84.200000} = 5\ (Hz)$$

$$\text{The timer speed:} \quad \mathrm{T} = \frac{1}{f} = \frac{1}{5} = 0.2\ (s) = 200\ \ (ms)$$

```c
static void MX_TIM2_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 84;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 200000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
```

Figure 38. Timer initialization function

- Reading the data from sensors: There are 3 functions in the HAL library that support reading the data directly from the register following uint16_t type, it is an unsigned 16-bit integer (2 Bytes):
- The gyroscope values:

```c
L3GD20_ReadXYZAngRate(gyro_read);
```

- The acceleration values:

```c
LSM303DLHC_AccReadXYZ(acc_read);
```

- The magnetometer values:

```c
LSM303DLHC_MagReadXYZ(mag_read);
```

- Functions are created to calculate results according to physical value and double data type. The reading value from the register (After subtracting the calibration value), it is multiplied by the data resolution. Resolution is calculated by the following formula:

$$\text{Resolution} = \frac{\text{Full scale}}{\text{Number of step}}$$

- For angular rate with full scales of ± 500 degree per second following uint16_t data type:

$$\text{Resolution} = \frac{1000}{2^{16} - 1} = 0.0152590219$$

- For accelerometer and magnetometer with full scales of ± 2 g following uint16_t data type:

$$\text{Resolution} = \frac{4}{2^{16} - 1} = 0.00006103608$$

- Basing on the above result, code lines to get data that are written in Keil C as follows:

```
Gyro_Data[i]=(double)((gyro_read[i]-gyro_calib[i])*0.0152590219 );
```

```
Acc_Data[i]= (double)((acc_read[i])*0.00006103608);
```

```
Mag_Data[i]=(double)((mag_read[i])*0.00006103608);
```

- Main programing

The main structure of program according to the basic structure of C program with sections: link, definition, global declaration, main function and subprogram. The STM32F411 Discovery Kit works following described flowchart as:

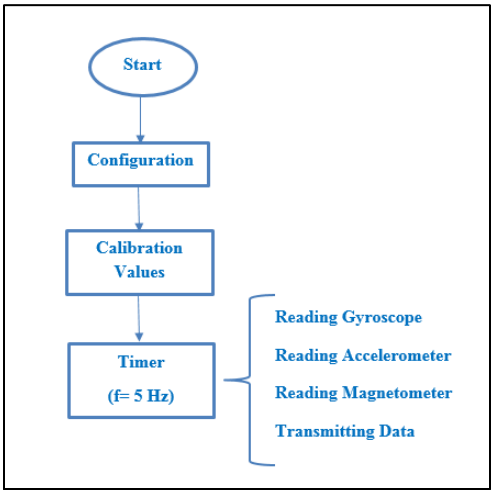

Figure 39. Operation Diagram

- In interrupt timer function section, it gets the data and transmits to COM port each time (Frequency equal to 5 Hz). The transmit code is written in Keil C as:

```
CDC_Transmit_FS(buffer_to_send,strlen((const char*)buffer_to_send));
```

- The result:

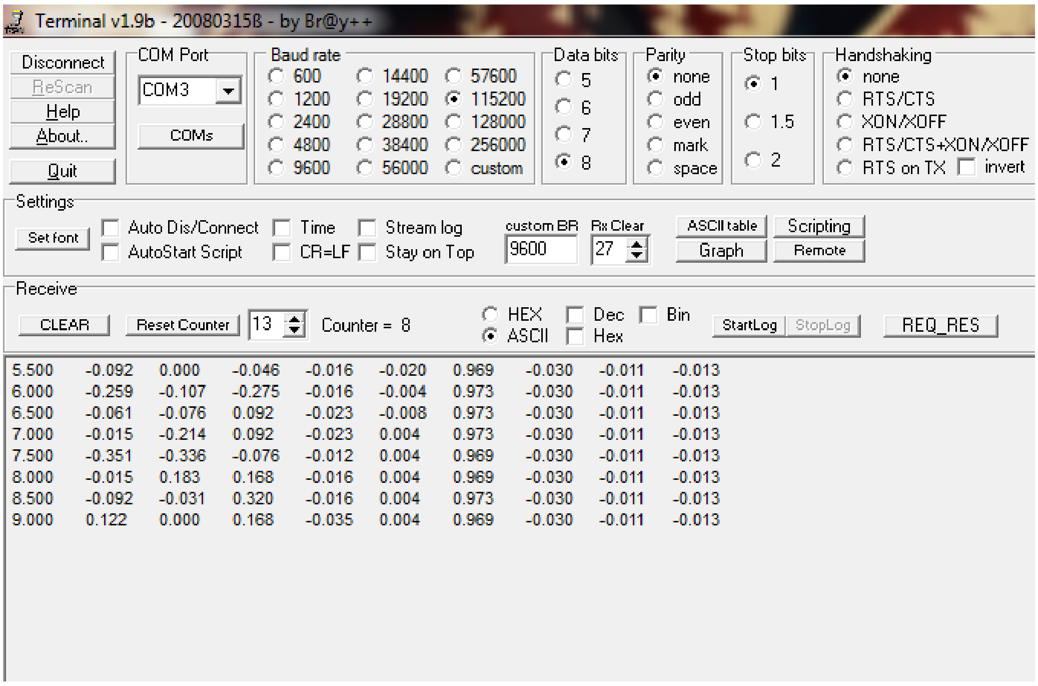

Figure 40. The result of Ex. 7