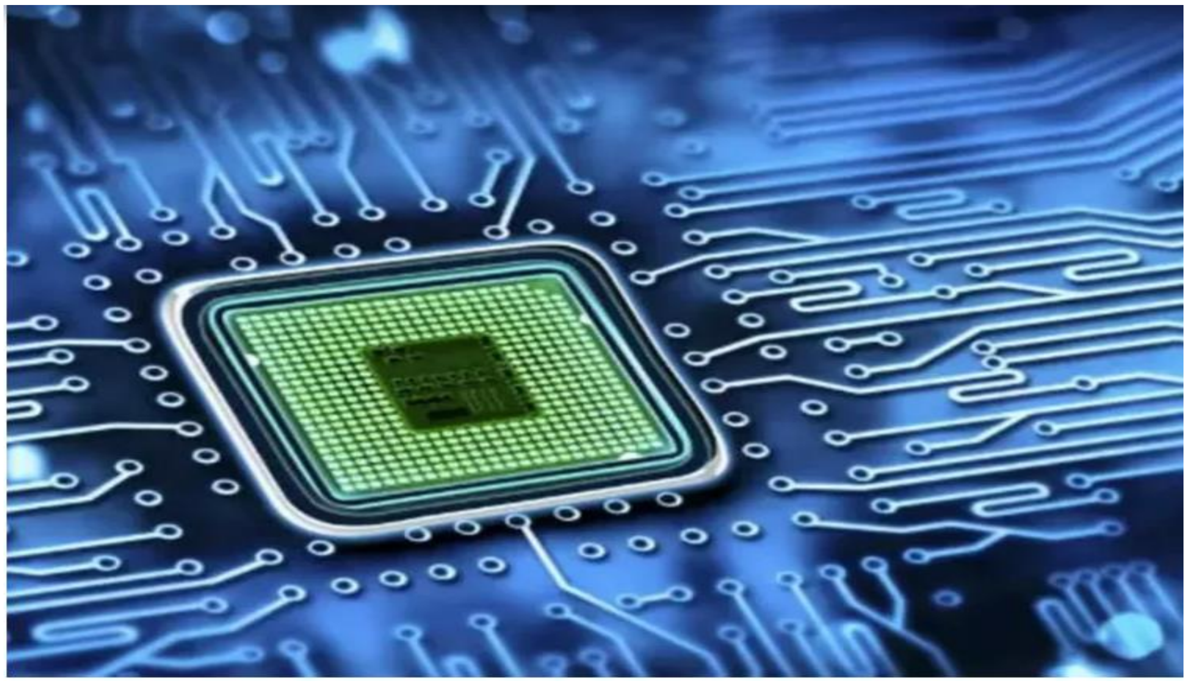


Laboratory 1

Using the STM32 communication peripherals 1

Students' Name: Yousha Mohammed Shamin Yeasher (WBDDO4)

Das Anjan Kumar (D42DQA)

Instructor: Kovács Viktor

Course: Microcontroller Laboratory Exercises VIAUAC08

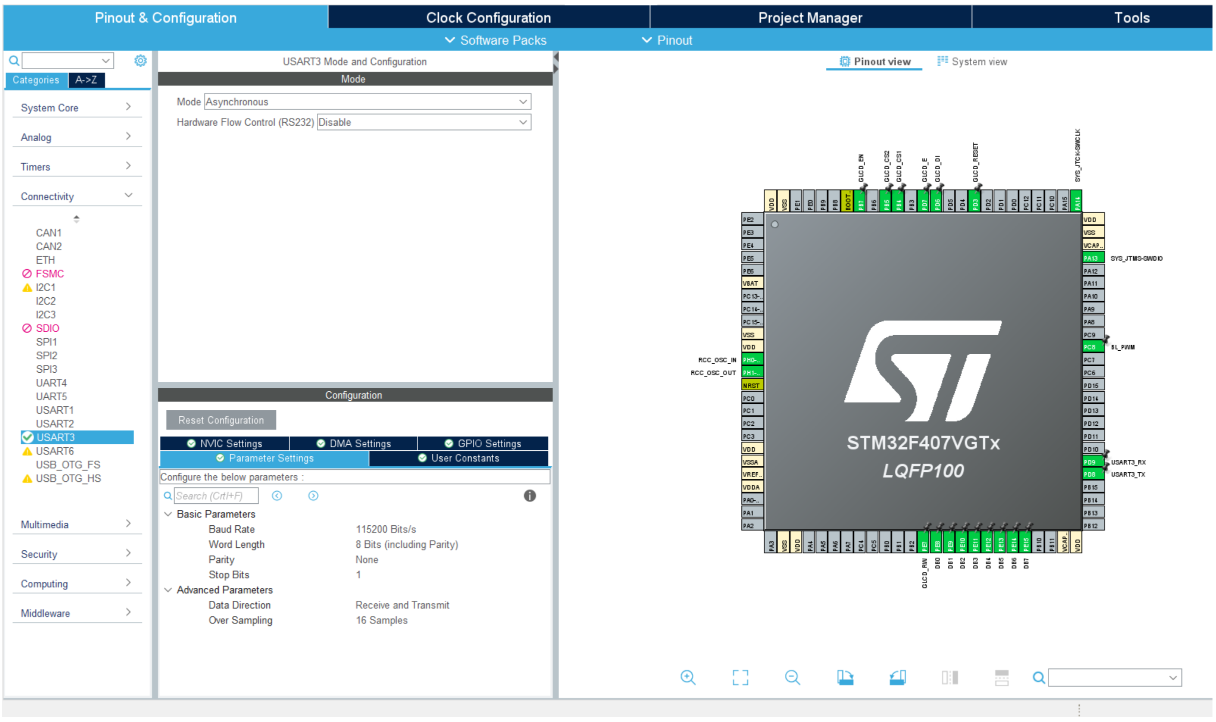
Date: 5th April, 2022

At first we will connect all the devices according to the instruction. We will connect power supply and also the usb cable tot he STM32 board.

The main task of the laboratory is to understand behaviour of UART and I2C communication Interface.

Task 1:

First we wil configure the settings in the graphical interface in the STM32 for UART.



USART3 Mode and Configuration

Mode

Mode Asynchronous

Hardware Flow Control (RS232) Disable

Configure the below parameters :

Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	16 Samples

In this task we will introduce an empty loop to demonstrate that we will not do anything unless untill our conditions are true. To do that we have implemented the below while loop

```
while((huart3.gState != HAL_UART_STATE_READY) && (huart3.gState != HAL_UART_STATE_BUSY_RX)){}
```

After that we will implement the sending of content str over UART. Below is the implementation os the second code:

```
strncpy(txBuffer,str,TXBUFFERSIZE);

HAL_UART_Transmit_IT(&huart3,(uint8_t*)txBuffer, strlen(txBuffer));
```

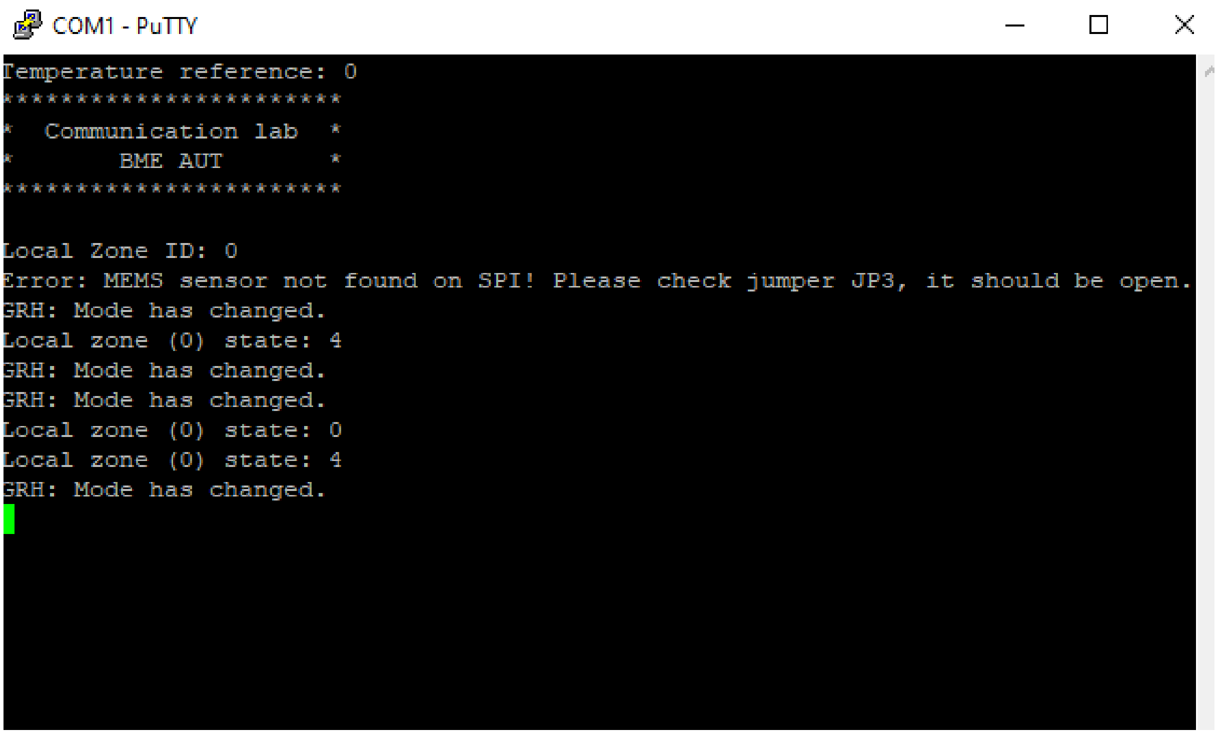
Here we are first copying the str to txbuffer so that we dont lose the previous value and after that we will transmit the str over UART.

After that we will receive the transmitted message in the UART.

To do that we will implement the below code:

```
MX_USART3_UART_Init();
HAL_UART_MspInit(&huart3);

HAL_UART_Receive_IT(&huart3, (uint8_t*)&rxBuffer, sizeof(uint8_t));
```

After running the code we can see the following output below in the screen shot:



```
Temperature reference: 0
***********************
*  Communication lab  *
*       BME AUT       *
***********************

Local Zone ID: 0
Error: MEMS sensor not found on SPI! Please check jumper JP3, it should be open.
GRH: Mode has changed.
Local zone (0) state: 4
GRH: Mode has changed.
GRH: Mode has changed.
Local zone (0) state: 0
Local zone (0) state: 4
GRH: Mode has changed.
```

And according to our expected outcome we can see that when we toggle the switches connected with the STM32, we can observe changes in the Putty.

Task 2:

In this task we are going to work with UART to change log levels and we will use as instructed.

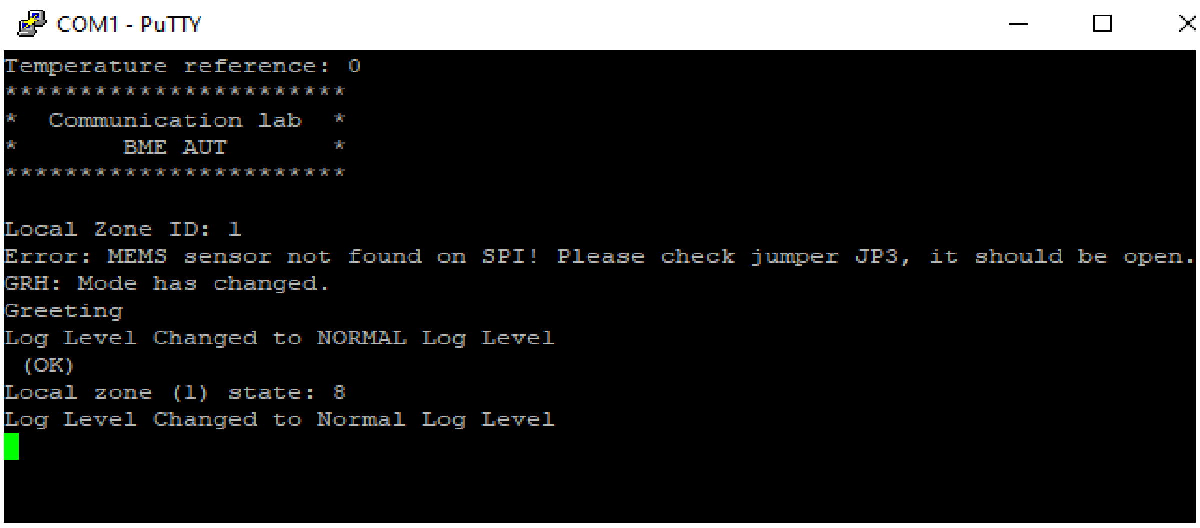
The code used in bsp_uart.c:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *handle)
{

lastReceivedUartCommand = rxBuffer;

HAL_UART_Receive_IT(handle, (uint8_t*)&rxBuffer, 1);

}
```

The switch cases used in the mainlop.c:

```
if(lastReceivedUartCommand != 0)
      {


             switch(lastReceivedUartCommand)

             {

             case 'g': Log_LogString("Greeting\n\r",LOGLEVEL_NORMAL); break;

             case 'n': currentLogLevel = LOGLEVEL_NORMAL; Log_LogString("Log Level
Changed to Normal Log Level\n\r",LOGLEVEL_NORMAL);break;

             case 'd': currentLogLevel = LOGLEVEL_DETAILED; Log_LogString("Log
Level Changed to Detailed Log Level\n\r",LOGLEVEL_DETAILED);break;

             case 'q': Log_LogStringAndHalStatus("Log Level Changed to NORMAL Log
Level\n\r",LOGLEVEL_NORMAL,HAL_OK);break;

             case 'v': ToggleVirtualTestSwitch();break;

             case 'r': repeatZoneDataEnabled = repeatZoneDataEnabled;break;

             default : Log_LogString("ERROR\n\r", LOGLEVEL_NORMAL);
             }

             lastReceivedUartCommand = 0;


      }
```

After implementing the code, we can see the below screen shot after pushing the appropriates keys in the ley board:

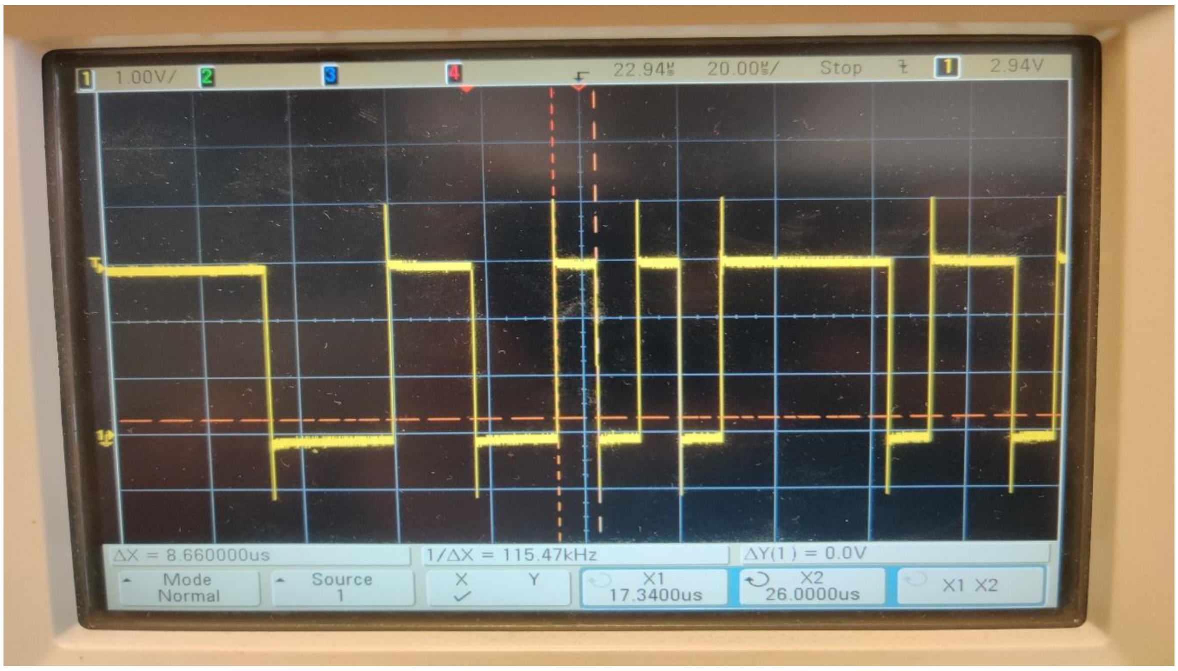


```
Temperature reference: 0
***********************
*  Communication lab  *
*       BME AUT       *
***********************

Local Zone ID: 1
Error: MEMS sensor not found on SPI! Please check jumper JP3, it should be open.
GRH: Mode has changed.
Greeting
Log Level Changed to NORMAL Log Level
 (OK)
Local zone (1) state: 8
Log Level Changed to Normal Log Level
```

Oscilloscope experiment:

In this experiment we will see the UART frame that has been sent by the controller. First we will connect to the PD8 and the ground point and then connect to the oscilloscope. After that we will set the options in the oscilloscope to NORMAL mode, FALLING edge and SOURCE 1.

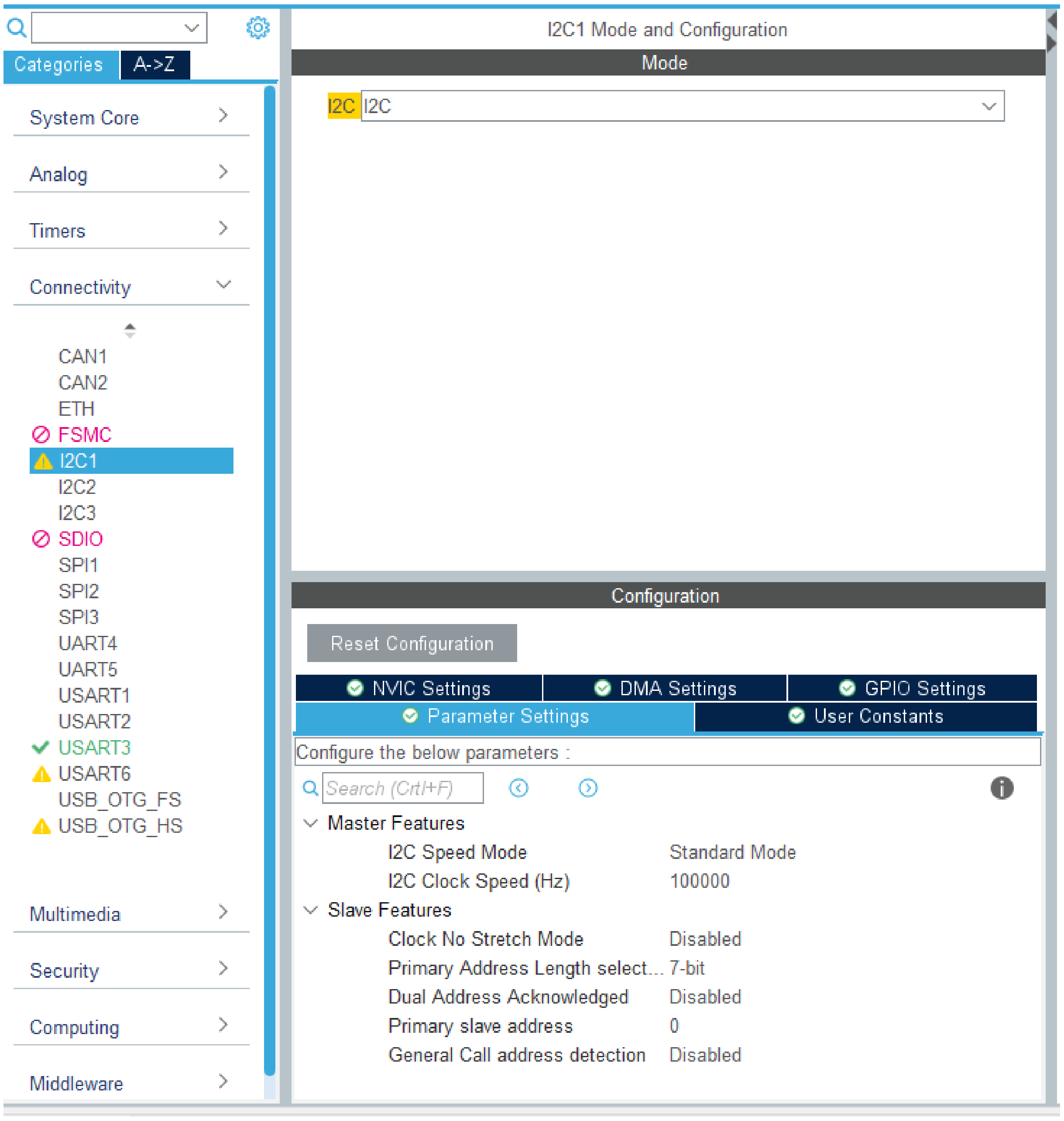
Then we can see the frame of the UART in the oscilloscope. We can look at the screenshoit below:



Here we can see that the start byte is initiated with a falling byte. And we have also measured the value of bit frequency by taking the difference from the x axis and inversing it. From a look up table we can also find the value of the bit using that frequency.

TASK 3:

In this task we will be reading out the value of the temperature using the I2C interface. To do that we will set up the graphical GPIO pins using the instructions and after that we will implement the fuctions almost like the UART interface like. initializing the interface and so on.



I2C I2C

Configure the below parameters :

Master Features

I2C Speed Mode	Standard Mode
I2C Clock Speed (Hz)	100000

Slave Features

Clock No Stretch Mode	Disabled
Primary Address Length select...	7-bit
Dual Address Acknowledged	Disabled
Primary slave address	0
General Call address detection	Disabled

Code for initialization:

```
	MX_I2C1_Init();
	HAL_I2C_MspInit(&hi2c1);
```

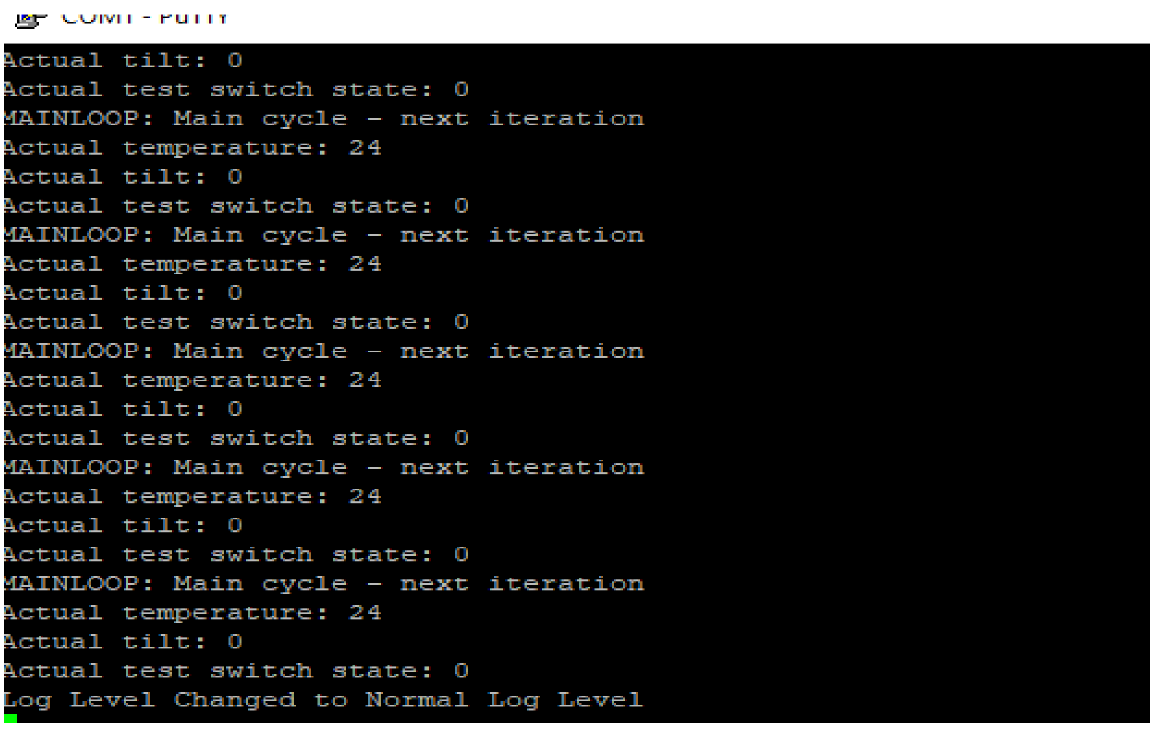
Code for reading :

```
HAL_StatusTypeDef I2C_ReadRegister(uint16_t deviceAddress, uint8_t
registerAddress, uint8_t *pData, uint16_t dataSize)
{
	HAL_I2C_Mem_Read(&hi2c1 , deviceAddress , registerAddress,
sizeof(uint8_t) , pData , dataSize , i2c_timeout);

	return HAL_OK;
}
```

Code for writing :

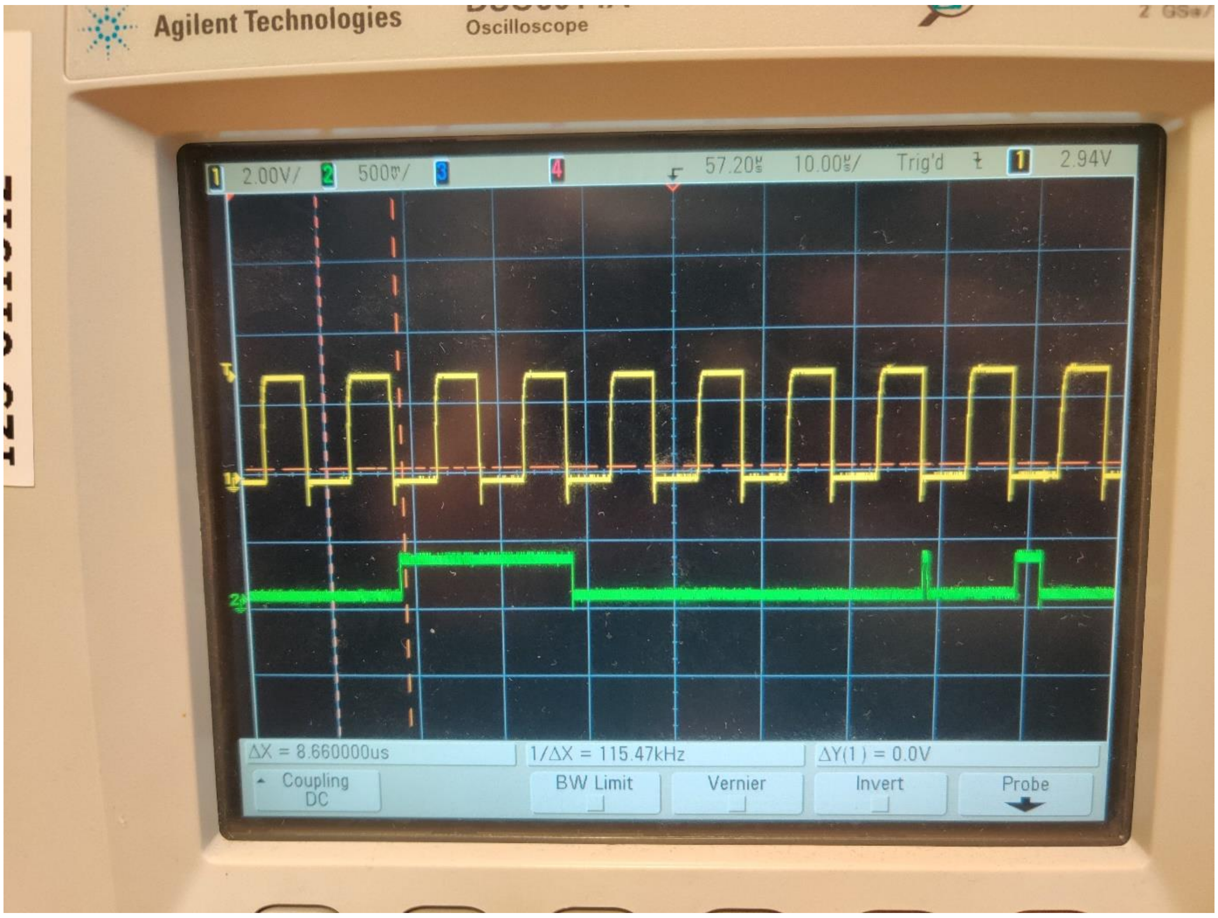
```
HAL_StatusTypeDef I2C_WriteRegister (uint16_t deviceAddress, uint8_t
registerAddress, uint8_t *pData, uint16_t dataSize)
{
	HAL_I2C_Mem_Write(&hi2c1 , deviceAddress , registerAddress,
sizeof(uint8_t) , pData , dataSize , i2c_timeout);

	return HAL_OK;
}
```

After that we will be able to see the temperature values in the putty interface on screen. We can see the below screenshot for the Interface :



```
Actual tilt: 0
Actual test switch state: 0
MAINLOOP: Main cycle - next iteration
Actual temperature: 24
Actual tilt: 0
Actual test switch state: 0
MAINLOOP: Main cycle - next iteration
Actual temperature: 24
Actual tilt: 0
Actual test switch state: 0
MAINLOOP: Main cycle - next iteration
Actual temperature: 24
Actual tilt: 0
Actual test switch state: 0
MAINLOOP: Main cycle - next iteration
Actual temperature: 24
Actual tilt: 0
Actual test switch state: 0
MAINLOOP: Main cycle - next iteration
Actual temperature: 24
Actual tilt: 0
Actual test switch state: 0
Log Level Changed to Normal Log Level
```

Oscilloscope View of the temperature:



We can see the SCL and SDA lines of the I2C in the oscilloscpe. We can observe a phenomena in the channel 2 of the oscilloscope, which is SDA of the I2C , we can see a sharp and quick green spike which can be identified as the acknowledge bit of the frame, but the main reason of this incident is the push pull drain and use of pull up resistor. This happens when the master(s) to pull SDA to HIGH while they are idle, and then tell one master to start sending data, it will pull SDA low.