Lab Worksheet 3

CSE360: Computer Interfacing
Department of Computer Science and Engineering

Lab 03: Introduction to I2C protocol using a DHT22 sensor and LCD display on STM32.

I. Topic Overview

This lab worksheet is to introduce the I2C protocol on an STM32 microcontroller, using it to receive data from a DHT22 sensor and displaying it to the LCD display. This involves setting up the I2C communication on the STM32, interfacing with the DHT22 sensor, and writing the necessary code to read and display the temperature and humidity data.

II. Learning Outcome

After this lab, students will be able to:

1. Understand the basic principles of I2C communication and its configuration on an STM32 microcontroller.
2. Interface a DHT22 sensor with an STM32 and implement the protocol to read data from the sensor.
3. Process the sensor data to obtain meaningful temperature and humidity readings.
4. Transmit the sensor data over I2C to a terminal or PC and display it in a human-readable format.

III. Materials

- STM32 development board (e.g., STM32F446RE)
- DHT22 Sensor
- LCD display

- Jumper wires
- Breadboard

IV. STM32F446RE Pins

The board consists of 64 pins out of which 38 pins are used and depicted in the following diagram. The relevant ports, pins and registers for today's lab are described later.

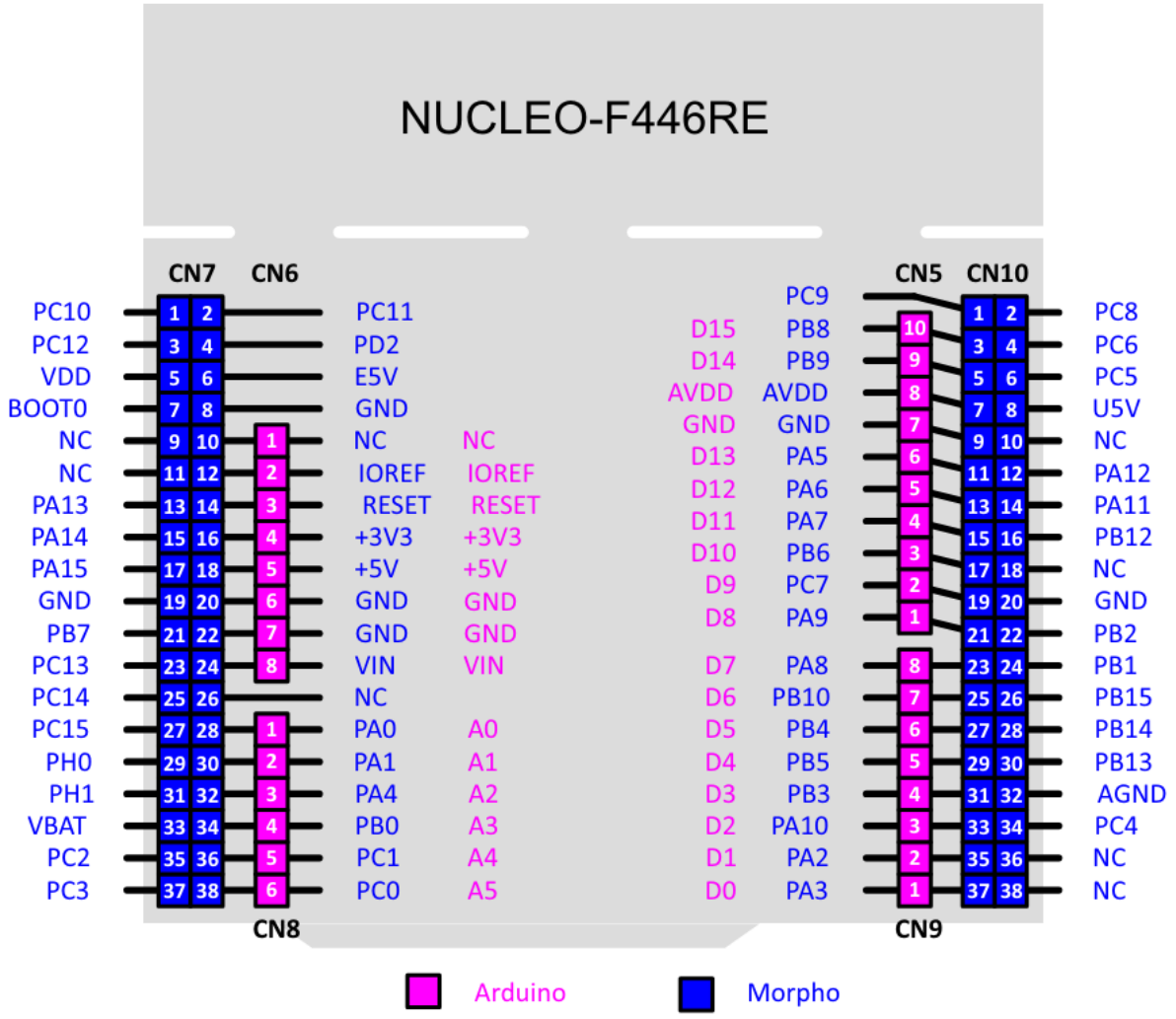


Figure 3: Pin diagram of STM32 development board

V. Registers: There are many registers related to I2C in stm32 but we only need a few of them for this lab.

Registers to Configure GPIO:

GPIO Mode Registers(MODER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO Output Type Registers(OTYPER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO Output Speed Registers(OSPEEDR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3[1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO Pullup-Pulldown Registers(PUPDR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO Alternate Function Low Register for Pin(0 to 7) (AFRL)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO Alternate Function High Register for Pin(8 to 15) (AFRH)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Registers to Configure I2C:

I2C Control Register One(CR1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SW RST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res.	SM BUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

I2C Control Register Two(CR2)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	LAST	DMA EN	ITBUF EN	ITEVT EN	ITERR EN	Res.	Res.	FREQ[5:0]					
			rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

I2C Status Register One(SR1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIMEOUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

I2C Clock Control Register(CCR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Res.	Res.	CCR[11:0]											
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

I2C TRISE Register (TRISE)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TRISE[5:0]					
										rw	rw	rw	rw	rw	rw

VI. Steps to Configure GPIO and I2C:

We Have to follow some particular steps to configure GPIO and I2C to work.

GPIO:

1. Enabling Clock for the GPIO port we need. (for this lab it is GPIOB)
2. We have to configure the mode of SDA and SCL pin as Alternate Function for this lab (Pin 8 = SCL, Pin 9 = SDA)
3. W**e have to configure the output type as open drain**
4. Set output speed high
5. **Enable pull up for SDA and SCL pin Using PUPDR**
6. We have to set the alternate function number in the AFR register. (for this lab. Our alternate function number is 4).

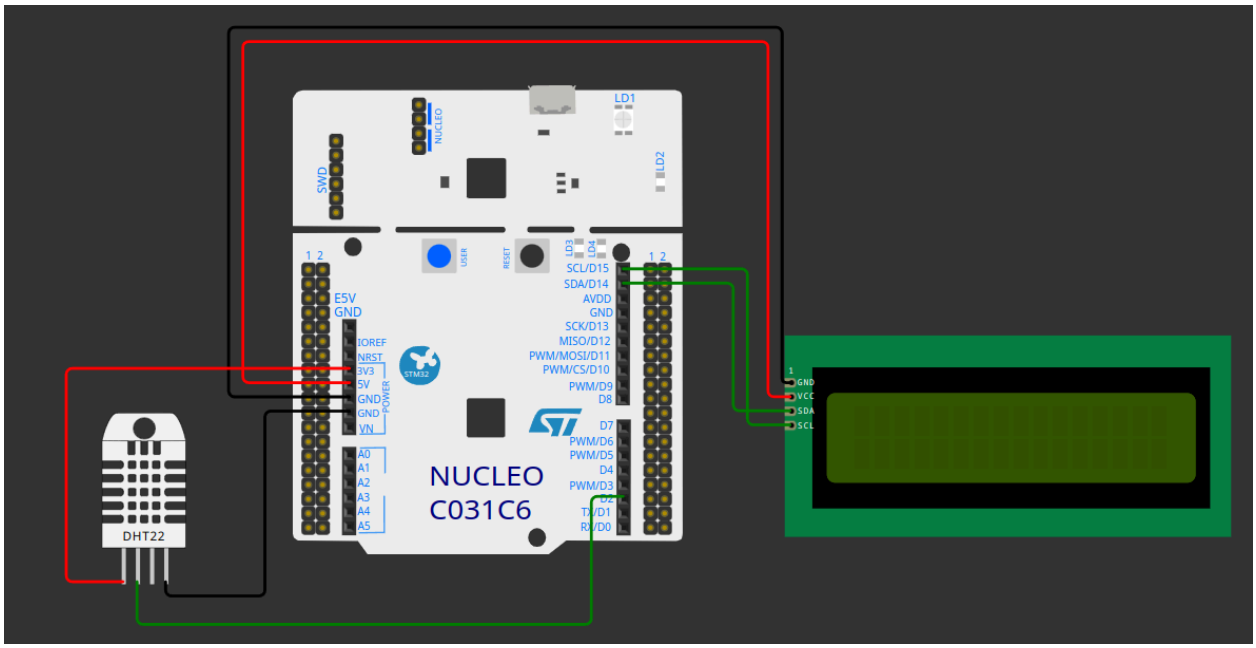
I2C:

1. Enabling Clock for I2C peripheral. (I2C1 for this lab)
2. Resetting the I2C Peripheral
3. Configuring the Clock Frequency at CR2
4. Configuring the CCR value
5. Configuring the TRISE Value
6. Enabling the Peripheral

VII. Open the Keil µVision5 IDE and follow the steps.

1. From the project tab in the menu bar, open a new project.
2. In the select device for target window, select STM32F446RETx under STMicroelectronics and click ok.
3. In the managed run-time environment window, select Core under CMSIS and Startup under Devices and click ok.
4. To create files, right click on the Source Group 1 tab under Target 1 and select Add new item. A pop up window will appear for file type. Create the following files with the mentioned type and name. Paste the following codes in their respective files.
5. From the options for target menu (magic icon), go to debug window, select ST-Link Debugger.

VIII. Circuit Diagram



IX. Code

i) File name: i2c.h, file type: header file (.h)

```
#ifndef _INC_I2C_H
#define _INC_I2C_H
#include "stm32f446xx.h"
```

```
void I2C_GPIO_Init();
void I2C_Init();
void I2C_Start();
void I2C_SendAddress(uint8_t address, uint8_t rw);
void I2C_Stop();
void I2C_WriteByte(uint8_t data);
uint8_t I2C_ReadByte();
void I2C_Write_Buffer(uint8_t address, uint8_t *buffer, uint8_t len);
void I2C_ACK_Enable();
void I2C_ACK_Disable();
#endif
```

ii) File name: i2c.c, file type: C file (.c)

// Datasheet - STM32F205xx STM32F207xx - Arm®-based 32-bit MCU, 150 DMIPs, up to 1 MB Flash/128+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces and camera

```c
#include "i2c.h"

void I2C_GPIO_Init()
{
   // Enabling Clock for GPIOB
   RCC->AHB1ENR |= (1 << 1); // RCC_AHB1ENR_GPIOBEN

   // Configure PB8 and PB9 for I2C1 SCL and SDA
   GPIOB->MODER &= ~((3 << (8 * 2)) | (3 << (9 * 2))); // Clear MODER bits
   GPIOB->MODER |= (2 << (8 * 2)) | (2 << (9 * 2)); // Set alternate function mode

   GPIOB->OTYPER |= (1 << 8) | (1 << 9); // Set open-drain output type

   GPIOB->PUPDR &= ~((3 << (8 * 2)) | (3 << (9 * 2))); // Clear PUPDR bits
   GPIOB->PUPDR |= (1 << (8 * 2)) | (1 << (9 * 2)); // Set pull-up resistors

   GPIOB->OSPEEDR |= (3 << (8 * 2)) | (3 << (9 * 2)); // Set high speed

   // Set alternate function to I2C1 (AF4)
   GPIOB->AFR[1] |= (4 << ((8 - 8) * 4)) | (4 << ((9 - 8) * 4));
}

void I2C_Init()
{
   // Enabling Clock for I2C1
   RCC->APB1ENR |= (1 << 21); // RCC_APB1ENR_I2C1EN

   // Reset I2C1
   I2C1->CR1 |= (1 << 15); // I2C_CR1_SWRST
   I2C1->CR1 &= ~(1 << 15); // Clear the SWRST bit

   // Configure I2C1
   I2C1->CR2 = 16 << 0; // Set APB1 clock frequency in MHz //100KHz
   I2C1->CCR = 80; // Set the clock control register ccr = fclk / (2 * i2c_freq)
   I2C1->TRISE = 17; // Maximum rise time

   // Enable I2C1
   I2C1->CR1 |= (1 << 0); // I2C_CR1_PE
}
```

```c
void I2C_Start()
{
   I2C1->CR1 |= (1 << 8); // I2C_CR1_START
   while (!(I2C1->SR1 & (1 << 0))); // I2C_SR1_SB  00001 & 000001
}

void I2C_Stop()
{
   I2C1->CR1 |= (1 << 9); // I2C_CR1_STOP
}

void I2C_ACK_Enable()
{
   I2C1->CR1 |= (1 << 10); // I2C_CR1_ACK
}

void I2C_ACK_Disable()
{
   I2C1->CR1 &= ~(1 << 10); // I2C_CR1_ACK
}

void I2C_SendAddress(uint8_t address, uint8_t rw) // R:1, W:0 //ADDR:0x27
{
   I2C1->DR = (address << 1) | rw;
   while (!(I2C1->SR1 & (1 << 1))); // I2C_SR1_ADDR=1
   (void)I2C1->SR1;// Clear ADDR flag
   (void)I2C1->SR2; // Clear ADDR flag
}

void I2C_WriteByte(uint8_t data)
{
   while (!(I2C1->SR1 & (1 << 7))); // I2C_SR1_TXE
   I2C1->DR = data;
   while (!(I2C1->SR1 & (1 << 2))); // I2C_SR1_BTF
}


uint8_t I2C_ReadByte()
```

```
{
   while (!(I2C1->SR1 & (1 << 6))); // I2C_SR1_RXNE
   return I2C1->DR;
}

void I2C_Write_Buffer(uint8_t address, uint8_t buffer[], uint8_t len)
{
   I2C_Start();
   I2C_SendAddress(address, 0); // W=0, R=1
   for (int i = 0; i < len; i++)
   {
       I2C_WriteByte(buffer[i]);
   }
   I2C_Stop();
}
```

iii) File name: main.c, file type: C file (.c)

```
#include "stm32f446xx.h"
#include "dht.h"
#include "lcd.h"


DHT dht = {GPIOA, 10};

int main()
{
   TimerInit();
   I2C_GPIO_Init();
   I2C_Init();
   DHT_begin(dht);//provided in driver
   LCD_Init();//provided in driver

   float temp, hum;
```

```
   while (1)
   {
       temp = DHT_ReadTemperature(dht, false);
       hum = DHT_ReadHumidity(dht);
       LCD_ClearDisplay();
       LCD_SetCursor(0, 0);

       LCD_Printf("Temp: %.2f", temp);
       LCD_SetCursor(1, 0);

       LCD_Printf("Humidity: %.2f", hum);
       DelayMS(1000);
   }
   return 0;
}
```

Driver code files: *[Thanks to Asraful Islam Taj for his special contribution]*

DHT22 Sensor Library Code from Github

After going to the github link there will be two files dht22.c and dht22.h create two files with the same name into your keil project and copy those code from that library use these 3 functions

```
void DHT_begin(DHT Config); // To initialize the Sensor
float DHT_ReadTemperature(DHT Config, bool isFahrenheit); // TO read Temperature
float DHT_ReadHumidity(DHT Config); // To Read Humidity
```

In Lab 2 we only use the DHT22 sensor so one library was enough but now we also have to use LCD display that's why we will also another library

I2C LCD Library Code from Github

After going to the github link there will be two files lcd.c and lcd.h create two files with the same name into your keil project and copy those code from that library use these 4 functions

```
void LCD_Init(); // TO initialize the LCD
void LCD_Printf(const char *format, ...); //To Print something in the LCD
void LCD_SetCursor(uint8_t row, uint8_t col); // To move the cursor location
void LCD_ClearDisplay(); // to clear the display
```

Note: For Better Understanding Please follow the given main.c file you can also consider that file as an example and try different things by yourself

After creating all the files, save all files > batch build > download.

VIII. Lab evaluation:

No evaluation for lab 3!

IX. References:

1. https://www.st.com/en/microcontrollers-microprocessors/stm32f446re.html
2. https://www.electronicsplanet.ch/en/electronics/resistor-color-code/resistor-color-code.php
3. https://pmdway.com/products/2-54mm-0-1-pitch-dual-row-jumper-cables-various-types-5-pack
4. http://designbuildcode.weebly.com/breadboard-circuits.html