UTM

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF ELECTRICAL ENGINEERING

MKEL1123-05 - ADVANCED MICROPROCESSOR SYSTEM

DESIGN A NON-INVASIVE PULSE OXIMETER DEVICE ON STM32 MICROCONTROLLER

MILESTONE 4

Prepared by:

Group 2

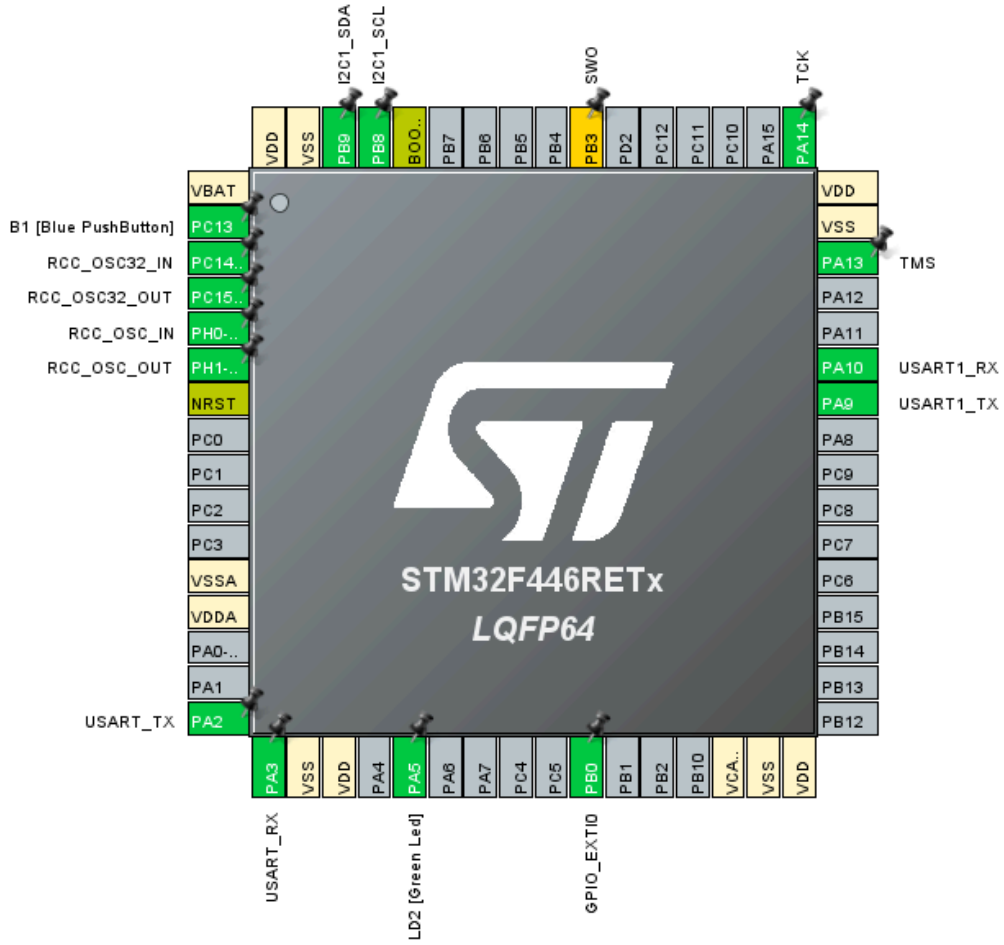
Name	Matric No.
Yvonne Wong Liang Liang	MKE231014
Koay Yi Yun	MKE231012
Aiman Farhan Mohd Foazi	MKE231020

Prepared for: **Dr. Mohd Afzan Othman**

1. Software/Tools setup

The software that was used is the STM32 Cube IDE which enables the usage of Nucleo Board F466RE.In the IDE we are able to set up the GPIO, clock cycle and pn configuration. In the IDE, first we need to find the board which we are currently using.

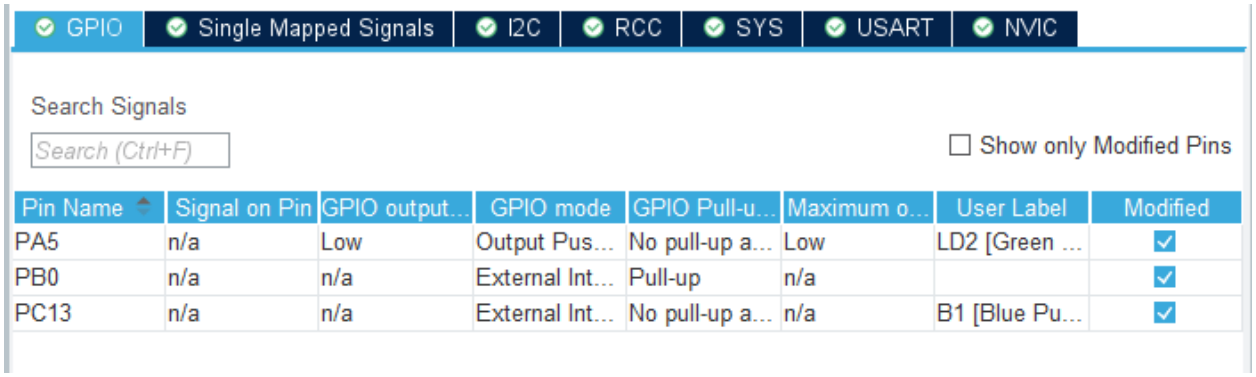
As shown below, the ioc will show the pin setup and configuration



2. Configuration Steps

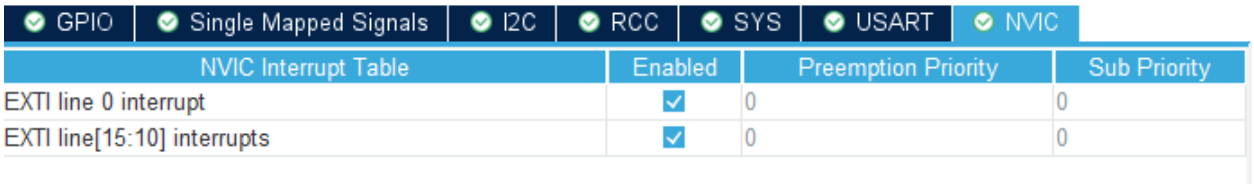
For the configuration steps:

1. In STM32CubeMX, set I2C1 to "I2C" and USART1 to "asynchronous"
2. Set up an external interrupt pin (say PB0) in GPIO settings, use "external interrupt mode with falling edge trigger detection" and "pull-up" settings.



Pin Name	Signal on Pin	GPIO output...	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PA5	n/a	Low	Output Pus...	No pull-up a...	Low	LD2 [Green ...	☑
PB0	n/a	n/a	External Int...	Pull-up	n/a		☑
PC13	n/a	n/a	External Int...	No pull-up a...	n/a	B1 [Blue Pu...	☑

3. Activate the external interrupt in NVIC settings by checking the corresponding box.



NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
EXTI line 0 interrupt	☑	0	0
EXTI line[15:10] interrupts	☑	0	0

4. Connect the INT# pin of your MAX30102 to this external interrupt pin.
5. Save and generate code.

The code which is the main.c file will be generate with the UART and I2C setup as shown below:

1. I2C Setup

```
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  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();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}
```

2. UART Setup:

```
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}
```

The code generated will also have the interrupt:

```
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */

  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */

  /* USER CODE END EXTI0_IRQn 1 */
}
```

3. Steps for firmware development

The firmware consists of main.c file and also includes the MAX30100_for_STM32_HAL.h file and MAX30100_for_STM32_HAL.c file. These files are open source C library intended for MAX30100 sensor implementation with STM32 boards. In the main file, the object max30100_t is declared and initiated in order to pass the I2C handle.

```
// max30100 object
max30100_t max30100;
```

In the initialization part, reset the sensor and clear the first-in-first-out (FIFO) pointers. Configure the FIFO setting with samples of 8 bits.

```
// Initiation
max30100_init(&max30100, &hi2c1);
max30100_reset(&max30100);
max30100_clear_fifo(&max30100);
max30100_set_fifo_config(&max30100, max30100_smp_ave_8, 1, 7);
```

Next, we need to do some settings on the MAX30100 sensor. The LED pulse width is set to 16 bits with analog to digital converter resolution of 2048 bits. The sampling rate of the sensor is set to 800us. Also, we need to set the measurement mode for SpO2.

```
// Sensor settings
max30100_set_led_pulse_width(&max30100, max30100_pw_16_bit);
max30100_set_adc_resolution(&max30100, max30100_adc_2048);
max30100_set_sampling_rate(&max30100, max30100_sr_800);
max30100_set_led_current_1(&max30100, 6.2);
max30100_set_led_current_2(&max30100, 6.2);

    // Enter SpO2 mode
    max30100_set_mode(&max30100, max30100_spo2);
    max30100_set_a_full(&max30100, 1);
```

To acquire the measurement, we use an interrupt function to query for the measurement. In the while loop, always check if the interrupt flag is activated. If the interrupt flag is set to 1, run the interrupt handler to read FIFO.

```
while (1)
{
  if (max30100_has_interrupt(&max30100))
  {
    max30100_interrupt_handler(&max30100);
  }
```

```
void max30100_interrupt_handler(max30100_t *obj)
{
    uint8_t reg[2] = {0x00};
    // Interrupt flag in registers 0x00 and 0x01 are cleared on read
    max30100_read(obj, max30100_INTERRUPT_STATUS_1, reg, 2);

    if ((reg[0] >> max30100_INTERRUPT_A_FULL) & 0x01)
    {
        // FIFO almost full
        max30100_read_fifo(obj);
    }

    if ((reg[0] >> max30100_INTERRUPT_PPG_RDY) & 0x01)
    {
        // New FIFO data ready
    }

    if ((reg[0] >> max30100_INTERRUPT_ALC_OVF) & 0x01)
    {
        // Ambient light overflow
    }

    if ((reg[1] >> max30100_INTERRUPT_DIE_TEMP_RDY) & 0x01)
    {
        // Temperature data ready
        int8_t temp_int;
        uint8_t temp_frac;
        max30100_read_temp(obj, &temp_int, &temp_frac);
        // float temp = temp_int + 0.0625f * temp_frac;
    }
```

Using the I2C peripherals, the sensor will read and write the data to and from I2C slave device. The values will be written to the registers.

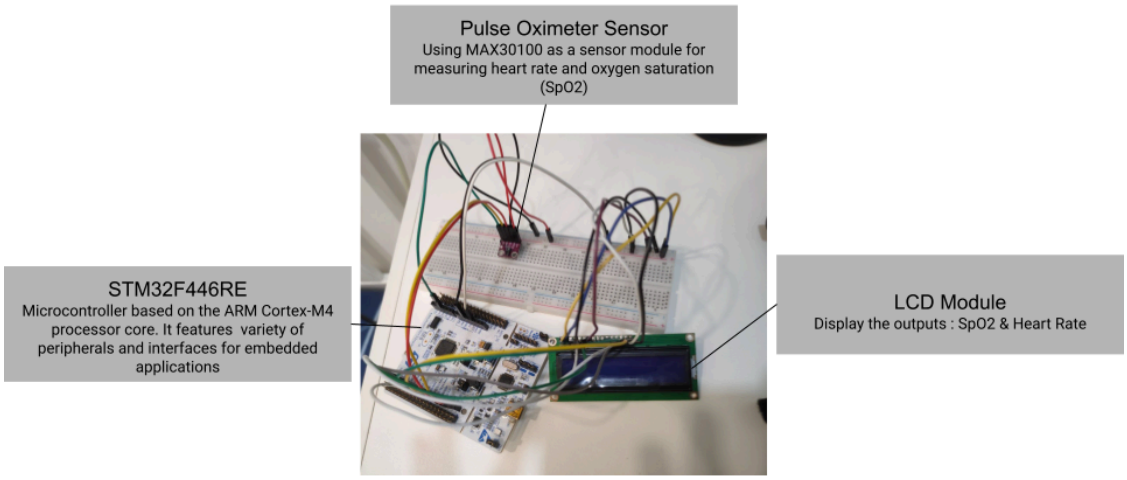
```
void max30100_write(max30100_t *obj, uint8_t reg, uint8_t *buf, uint16_t buflen)
{
    uint8_t *payload = (uint8_t *)malloc((buflen + 1) * sizeof(uint8_t));
    *payload = reg;
    if (buf != NULL && buflen != 0)
        memcpy(payload + 1, buf, buflen);
    HAL_I2C_Master_Transmit(obj->_ui2c, max30100_I2C_ADDR << 1, payload, buflen + 1, max30100_I2C_TIMEOUT);
    free(payload);
}

void max30100_read(max30100_t *obj, uint8_t reg, uint8_t *buf, uint16_t buflen)
{
    uint8_t reg_addr = reg;
    HAL_I2C_Master_Transmit(obj->_ui2c, max30100_I2C_ADDR << 1, &reg_addr, 1, max30100_I2C_TIMEOUT);
    HAL_I2C_Master_Receive(obj->_ui2c, max30100_I2C_ADDR << 1, buf, buflen, max30100_I2C_TIMEOUT);
}
```

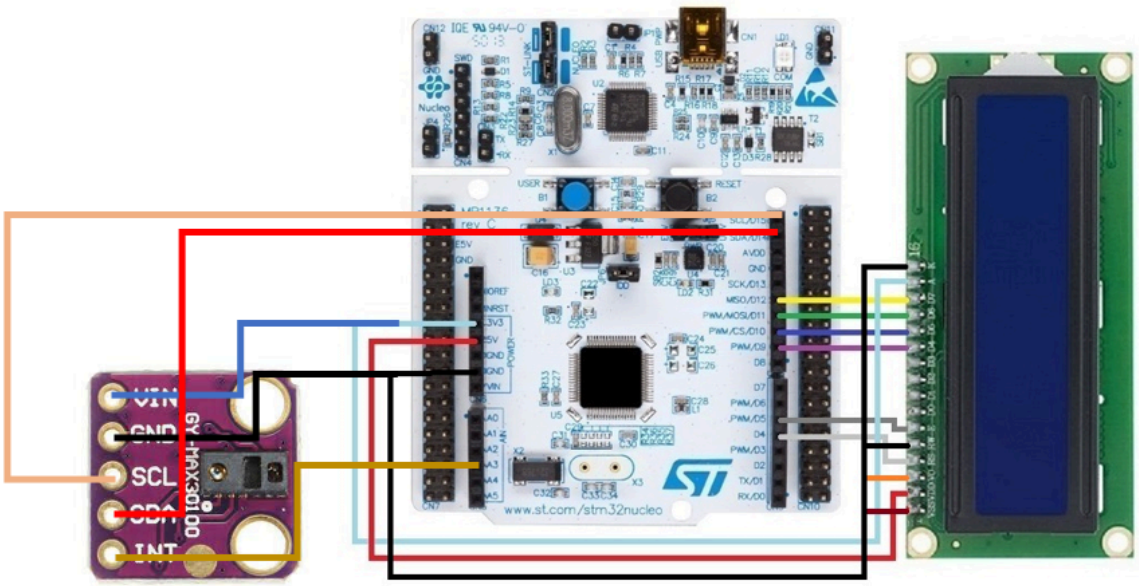
To display the measurement in the LCD, lcd.h and lcd.c files are included in the design files. LCD ports are initialized as GPIO_Output in the main.c file according to the wiring diagram.

```
Lcd_PortType ports[] = { GPIOC, GPIOB, GPIOA, GPIOA };
  // Lcd_PinType pins[] = {D4_Pin, D5_Pin, D6_Pin, D7_Pin};
  Lcd_PinType pins[] = {GPIO_PIN_7, GPIO_PIN_6, GPIO_PIN_7, GPIO_PIN_6};
  Lcd_HandleTypeDef lcd;
  // Lcd_create(ports, pins, RS_GPIO_Port, RS_Pin, EN_GPIO_Port, EN_Pin, LCD_4_BIT_MODE);
  lcd = Lcd_create(ports, pins, GPIOB, GPIO_PIN_5, GPIOB, GPIO_PIN_4, LCD_4_BIT_MODE);
  Lcd_cursor(&lcd, 1,1);
```

4. Steps for hardware development

A circuit is involved in the product design, which houses the main components: microcontroller (STM43F446RE), pulse oximeter sensor (MAX30100) and LCD Module as shown in the figure below. The SCL (Serial Clock) and SDA (Serial Data) pins are part of the I2C (Inter-Integrated Circuit) interface on the MAX30100 sensor, which is used for communication between the sensor and a microcontroller. The microcontroller generates clock pulses on the SCL line to control the timing of data transfer. The SDA pin is the data line for bidirectional communication between the MAX30100 sensor and the microcontroller. Data is transferred serially between devices on the I2C bus through the SDA line.





MAX30100
Pulse Oximeter Sensor

STM32F446RE

LCD Display

5. References

Design a non-Invasive Pulse Oximeter Device Based on PIC Microcontroller | IEEE Conference Publication | IEEE Xplore. (n.d.). Ieeexplore.ieee.org. Retrieved January 18, 2024, from https://ieeexplore.ieee.org/document/8925314

dxwy. (2024, January 12). *dxwy/MAX30100_for_STM32_HAL*. GitHub. https://github.com/dxwy/MAX30100_for_STM32_HAL

Lab, M. (2022, May 10). *MAX30100 Pulse Oximeter and Heart Rate Sensor with ESP32*. Microcontrollers Lab. https://microcontrollerslab.com/max30100-pulse-oximeter-heart-rate-sensor-esp32/

Pan. (2024, January 12). *eepj/MAX30102_for_STM32_HAL*. GitHub. https://github.com/eepj/MAX30102_for_STM32_HAL

Strogonovs, R. (n.d.). *Implementing pulse oximeter using MAX30100 - MORF - Coding And Engineering*. Morf.lv. https://morf.lv/implementing-pulse-oximeter-using-max30100/