EE2028 Microcontroller Programming and Interfacing

Assignment 2

Group 14
Lab day: Tuesday 2pm-5pm
Group members:
Sun Qiyang A0242744E
Darryl See A0216216N

Table of Content

Introduction and objectives

In this project, we will be implementing a system to have enhanced monitoring of COVID patients.

The system will be called Covid Patients Enhanced Monitor, COPEMON for short. COPEMON measures different sets of data from the Covid Patients and sends it back to a CHIP Associated Cloud Unit, CHIPACU for short.

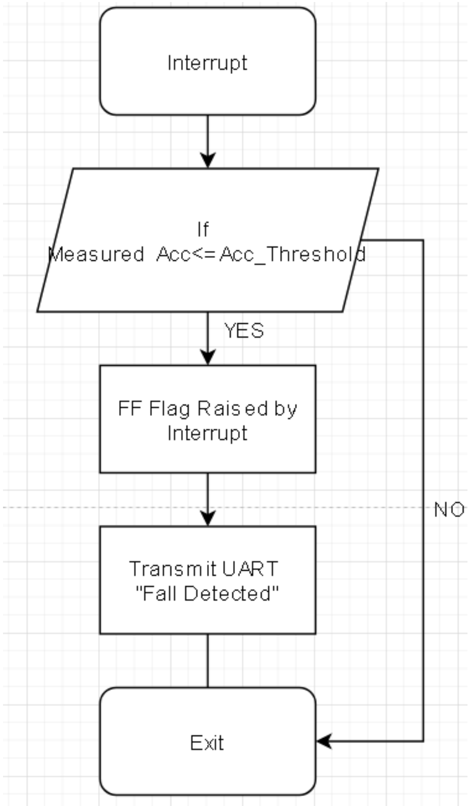
The objective for COPEMON is for it to be able to monitor COVID patients, especially elderly COVID patients, and return data on their condition regularly.

COPEMON should have 2 modes, Normal and Intensive Care Mode.

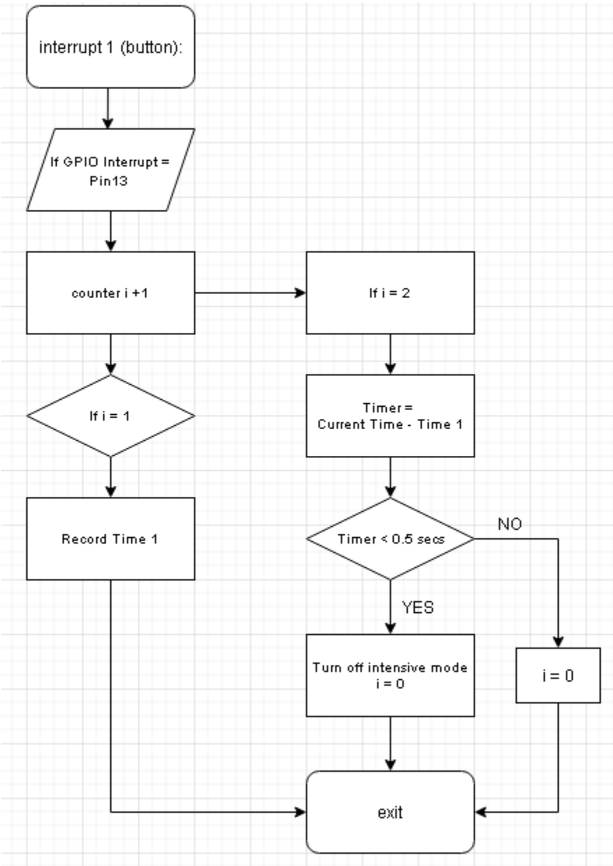
In Normal Mode, COPEMON will be returning data on the patient's temperature, posture, humidity and pressure in the lungs.

In Intensive Care Mode, COPEMON will be measuring pain detection and orientation of the patient, in addition to the data measured from normal mode.

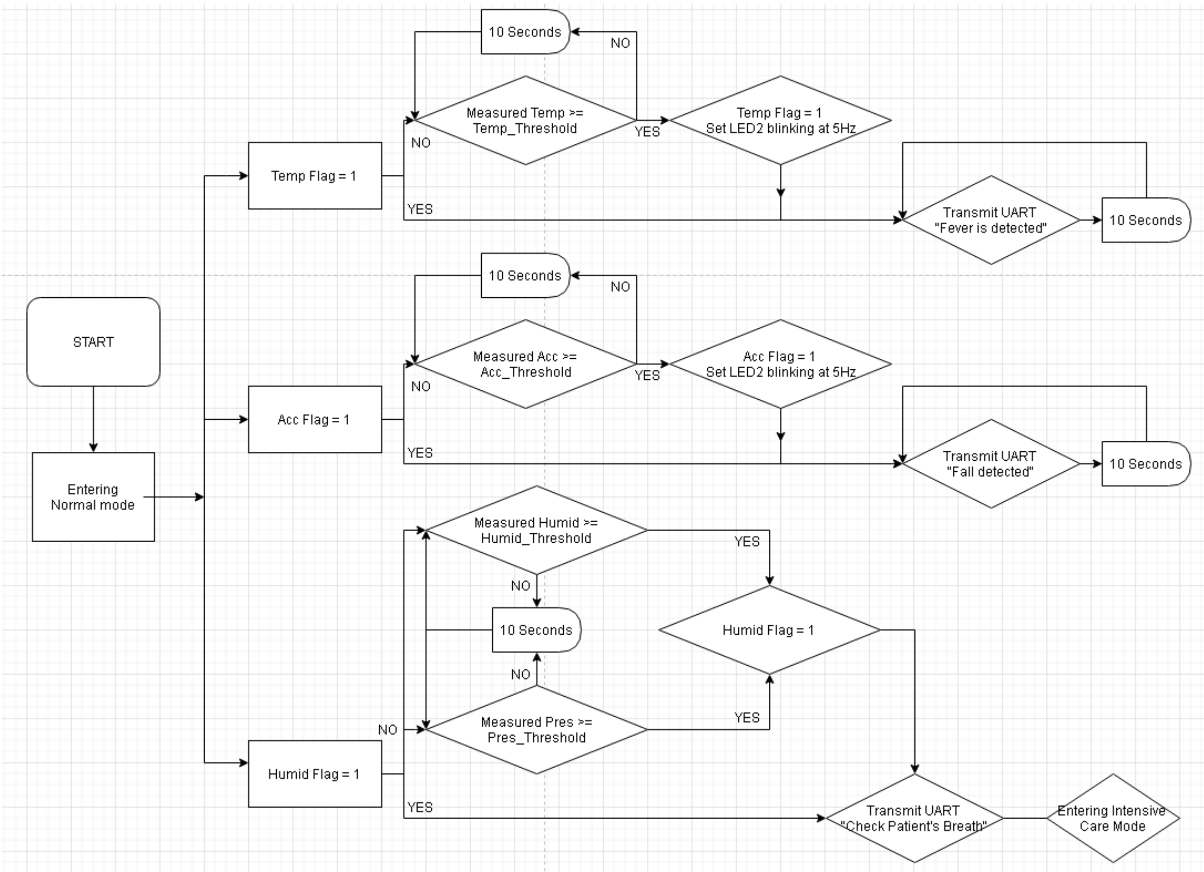
Flowcharts describing the system design and processes



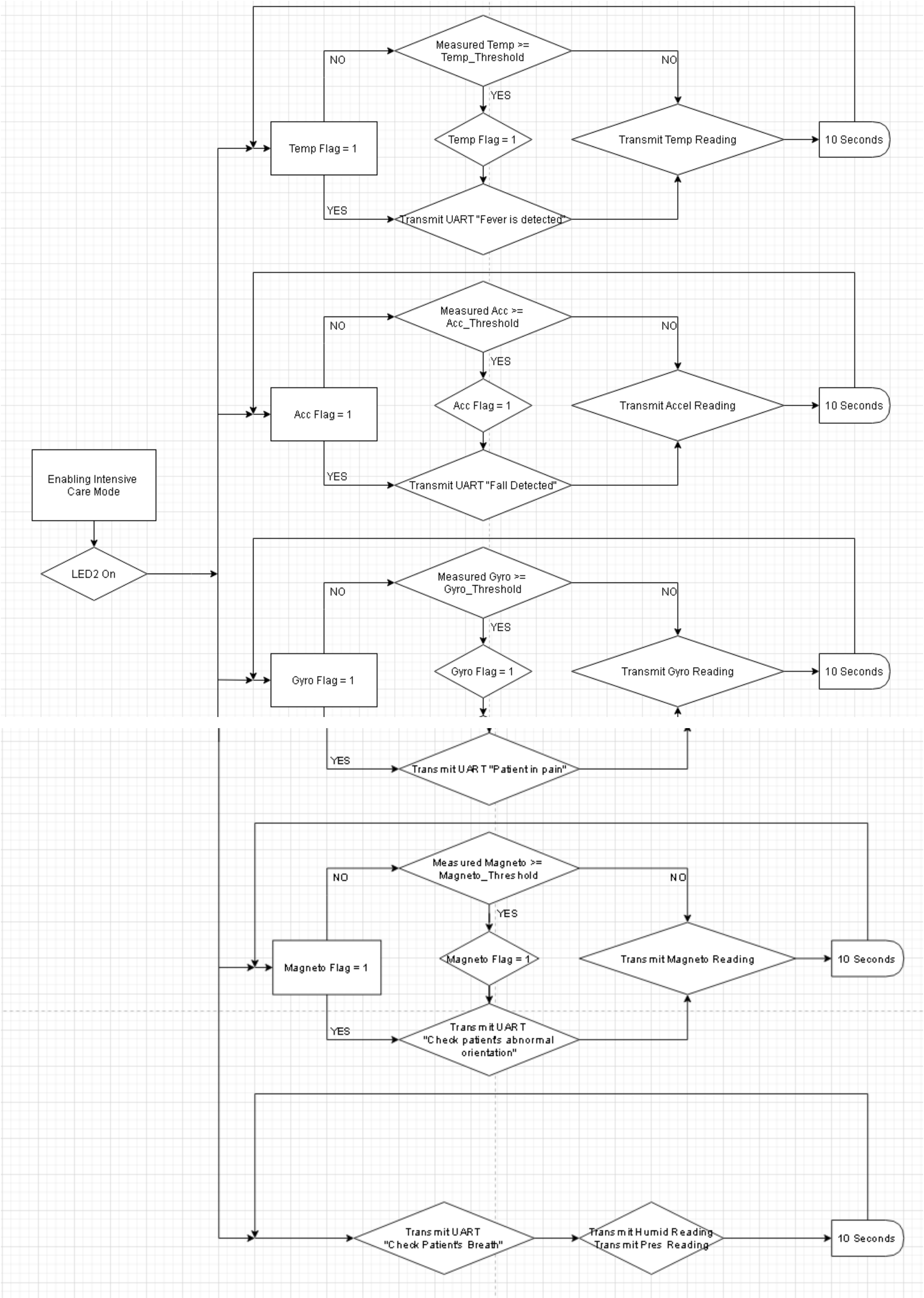
Flow Chart for Accelerometer Interrupt



Flow Chart for Push Button Interrupt



Flow Chart for Normal Mode



Flow Chart for Intensive Mode

Detailed Implementation

Before starting off the main code, we need to initialise all the peripheral sensors, GPIO, UART and Interrupts. We also need to declare all the fixed variables and strings we will be repeating in the program outside the main function.

Inside the main function, we call new variables to store values our sensors read. We then initialise the peripherals using the BSP functions, enabling NVIC for interrupts.

Once we finish setting up all the sensors, we can start our main loop in accordance with our flowchart to demonstrate our design.

We use both polling mode and interrupt mode in our design. For interrupt, we use two interrupts, one is the button, the other is accelerometer sensor interrupt.

The main difference between interrupt and polling is that **in interrupt, the device notifies the CPU that it requires attention** while, in polling, the CPU continuously checks the status of the devices to find whether they require attention.

For the timer:

1. Sensors collects the data every 1s, Write an if condition for it,
   HAL_GetTick()-time_one > 1000, and update time_one, time_one = HAL_GetTick();
   at the end of the if block.

2. Send the warning message every 10s, Write an if condition for it,
   HAL_GetTick()-time_ten > 10000, and update time_ten, time_ten = HAL_GetTick();
   at the end of the if block.

3. If Led needs to blink 5Hz, 5Hz = 0.2s blink once. Write an if condition for it,
   HAL_GetTick()-time_two > 200, and update time_two at the end of the if block.

Devices used:

ACCELEROMETER: 3D accelerometer LSM6DSL

LSM6DSL_INT1_EXTI11, PD 11, INT 1
Accelerometer is used for sensor interrupt. There are three steps for

1. **Enable the free fall interrupt**

```
void accelero_interrupt_config(void){
	uint8_t Buffer;
	Buffer = 0x80; // |= (1<<7) 1000 0000 set bit[7] to enable basic interrupts
	SENSOR_IO_Write(LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW,
LSM6DSL_ACC_GYRO_TAP_CFG1, Buffer);
LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW// SAD[0] I2C address
LSM6DSL_ACC_GYRO_TAP_CFG1// Register address
```

9.75 TAP_CFG (58h)

Enables interrupt and inactivity functions, configuration of filtering and tap recognition functions (r/w).

Table 183. TAP_CFG register

INTERRUPTS_ENABLE	INACT_EN1	INACT_EN0	SLOPE_FDS	TAP_X_EN	TAP_Y_EN	TAP_Z_EN	LIR

Table 184. TAP_CFG register description

INTERRUPTS_ENABLE	Enable basic interrupts (6D/4D, free-fall, wake-up, tap, inactivity). Default 0. (0: interrupt disabled; 1: interrupt enabled)
INACT_EN[1:0]	Enable inactivity function. Default value: 00 (00: disabled 01: sets accelerometer ODR to 12.5 Hz (low-power mode), gyro does not change; 10: sets accelerometer ODR to 12.5 Hz (low-power mode), gyro to sleep mode; 11: sets accelerometer ODR to 12.5 Hz (low-power mode), gyro to power-down mode)

enable the interrupt.

```
Buffer = 0x08; // 0000 1000 FF_Dur [4:0] = 00001, FF_Ths [2:0] = 000
	SENSOR_IO_Write(LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW, LSM6DSL_ACC_GYRO_FREE_FALL, Buffer);
```

9.80 FREE_FALL (5Dh)

Free-fall function duration setting register (r/w).

Table 194. FREE_FALL register

FF_DUR4	FF_DUR3	FF_DUR2	FF_DUR1	FF_DUR0	FF_THS2	FF_THS1	FF_THS0

Table 195. FREE_FALL register description

FF_DUR[4:0]	Free-fall duration event. Default: 0 For the complete configuration of the free fall duration, refer to FF_DUR5 in *WAKE_UP_DUR (5Ch)* configuration
FF_THS[2:0]	Free fall threshold setting. Default: 000 For details refer to *Table 196*.

Table 196. Threshold for free-fall function

FF_THS[2:0]	Threshold value
000	156 m*g*
001	219 m*g*
010	250 m*g*
011	312 m*g*
100	344 m*g*
101	406 m*g*
110	469 m*g*
111	500 m*g*

choose the threshold for 159 mg which is default threshold

```
Buffer = SENSOR_IO_Read(0xD4,0x5E);
	Buffer |= (1 << 4); //set bit[4] to route FF interrupt to INT1
	SENSOR_IO_Write(LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW, LSM6DSL_ACC_GYRO_MD1_CFG, Buffer);
```

9.81 MD1_CFG (5Eh)

Functions routing on INT1 register (r/w).

Table 197. MD1_CFG register

INT1_INACT_STATE	INT1_SINGLE_TAP	INT1_WU	INT1_FF	INT1_DOUBLE_TAP	INT1_6D	INT1_TILT	INT1_TIMER

Table 198. MD1_CFG register description

INT1_INACT_STATE	Routing on INT1 of inactivity mode. Default: 0 (0: routing on INT1 of inactivity disabled; 1: routing on INT1 of inactivity enabled)
INT1_SINGLE_TAP	Single-tap recognition routing on INT1. Default: 0 (0: routing of single-tap event on INT1 disabled; 1: routing of single-tap event on INT1 enabled)
INT1_WU	Routing of wakeup event on INT1. Default value: 0 (0: routing of wakeup event on INT1 disabled; 1: routing of wakeup event on INT1 enabled)
INT1_FF	Routing of free-fall event on INT1. Default value: 0 (0: routing of free-fall event on INT1 disabled; 1: routing of free-fall event on INT1 enabled)
INT1_DOUBLE_TAP	Routing of tap event on INT1. Default value: 0 (0: routing of double-tap event on INT1 disabled; 1: routing of double-tap event on INT1 enabled)
INT1_6D	Routing of 6D event on INT1. Default value: 0 (0: routing of 6D event on INT1 disabled; 1: routing of 6D event on INT1 enabled)
INT1_TILT	Routing of tilt event on INT1. Default value: 0 (0: routing of tilt event on INT1 disabled; 1: routing of tilt event on INT1 enabled)
INT1_TIMER	Routing of end counter event of timer on INT1. Default value: 0 (0: routing of end counter event of timer on INT1 disabled; 1: routing of end counter event of timer event on INT1 enabled)

```
}
```

2. **configure GPIO Pin lsm6dsl in MX_GPIO_Init**
3. **enable NVIC EXTI interrupt // HAL_NVIC_EnableIRQ(*EXTI1_IRQn*);**

4. **In HAL_GPIO_EXTI_Callback**

```
if (GPIO_Pin == LSM6DSL_INT1_EXTI11_Pin){
    uint8_t temp;
```

```
   temp = SENSOR_IO_Read(LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW+1,
LSM6DSL_ACC_GYRO_WAKE_UP_SRC); //I2C read address so it needs +1;
  temp &= 0x20; //read bit[5] to determine if FF flag was raised by device
  if (temp){
     accflag = WARNING;
    sprintf(message_print, "\r\nFall detected\r\n");
     HAL_UART_Transmit(&huart1, (uint8_t*)message_print, strlen(message_print),0xFFFF);
  }
}
```

GYROSCOPE: 3D gyroscope LSM6DSL

LSM6DSL_INT1_EXTI11, PD 11, INT 1

HUMIDITY_SENSOR: capacitive digital sensor HTS221

HTS221_DRDY_EXTI15, PD15

TEMPERATURE_SENSOR: capacitive digital sensor HTS221

HTS221_DRDY_EXTI15, PD15

for all sensors, there are some steps, take gyroscope as an example

1. #include
   #include "../../Drivers/BSP/B-L475E-IOT01/stm32l475e_iot01_gyro.h"
2. BSP_GYRO_Init(); //initailzation
3. BSP_GYRO_GetXYZ(gyro_data_i16); //get the data

LED: Gre_DRDY_EXTI15en LED2

1. Configuration of LED2 in MX_GPIO_Init() block
2. in the main(), if we need to toggle it, we use function below
   HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);

MODE_TOGGLE & WARNING_CLEAR: USER Button (blue), read using interrupts

BUTTON_EXTI13, PC13/WKUP2, User button interrupt need following 3 steps

1. initailize it in MX_GPIO_Init()
2. detect it in HAL_GPIO_EXTI_Callback()
   check if GPIO_Pin == BUTTON_EXTI13_Pin

CHIPACU: Terminal program (Tera Term) on a personal computer/laptop displaying Telemetry

1. Initialise UART1 using **static void UART1_Init(void)**; UART_HandleTypeDef huart1;

```
sprintf(message_print, "Fever is detected\r\n");
```

2. By formatting a string in sprintf, we can use the function HAL_UART_Transmit to send information to Tera Term.

```
HAL_UART_Transmit(&huart1, (uint8_t*)message_print, strlen(message_print),0xFFFF);
```

Enhancement

We did not implement any enhancement.

Problems Encountered and Solutions Proposed

One problem we encountered was how to quantify the data of the gyroscope in order to tell if the patient is in pain.
We used the root mean square value of the XYZ values of the gyroscope data to calculate the total magnitude of force that was being applied to the COPEMON and compared it to our GYRO_THRESHOLD to tell if the patient is in pain.

A second problem we encountered was that the humidity sensor on our board senses100% of the room's humidity, which is abnormal. Initially, we thought it's our program's problem, but later on, we realised that touching the humidity sensor causes the humidity to decrease.
A method of testing our program was touching the humidity sensor to decrease the humidity reading and pushing it into intensive mode.

Another problem we encountered was that we couldn't find some units of the output generated by the sensor such as the gyroscope.
In the end, we opted to assume that the units are degrees per second(dps).

Issues/Suggestions(can be good or bad)

One issue we faced was a lack of examples to compare to while working on the assignment. The assignment was very complicated and the lack of examples made it such that we had a rough time figuring things out.

One suggestion is that more real examples can be given during lecture time instead of just content from the lecture notes.

Conclusion

Overall, we successfully implemented a system to monitor COVID patients through the use of the sensors on the microcontrollers for the COPEMON system with embedded C programming, and sent information from a microcontroller chip that measures the data of patients to another server, CHIPACU for monitoring.