TEE2028 Microcontroller Programming and Interfacing

Lab Assignment 2

Group 3

Lecturer: Gu Jing

Group members:
Low Feng Rui (A0214782A)
Celine Oi Shu Jun (A0214749X)

Table of Contents Page

1. **Introduction and Objective**

This project is about implementing a system which does enhanced monitoring of COVID19 patients, especially elderly patients. This system is referred to as COvid Patients Enhanced MONitoring(COPEMON) and utilises various sensors on the board for data capture and transmission to an IoT server known as CHIP Associated Clou Unit(CHIPACU).

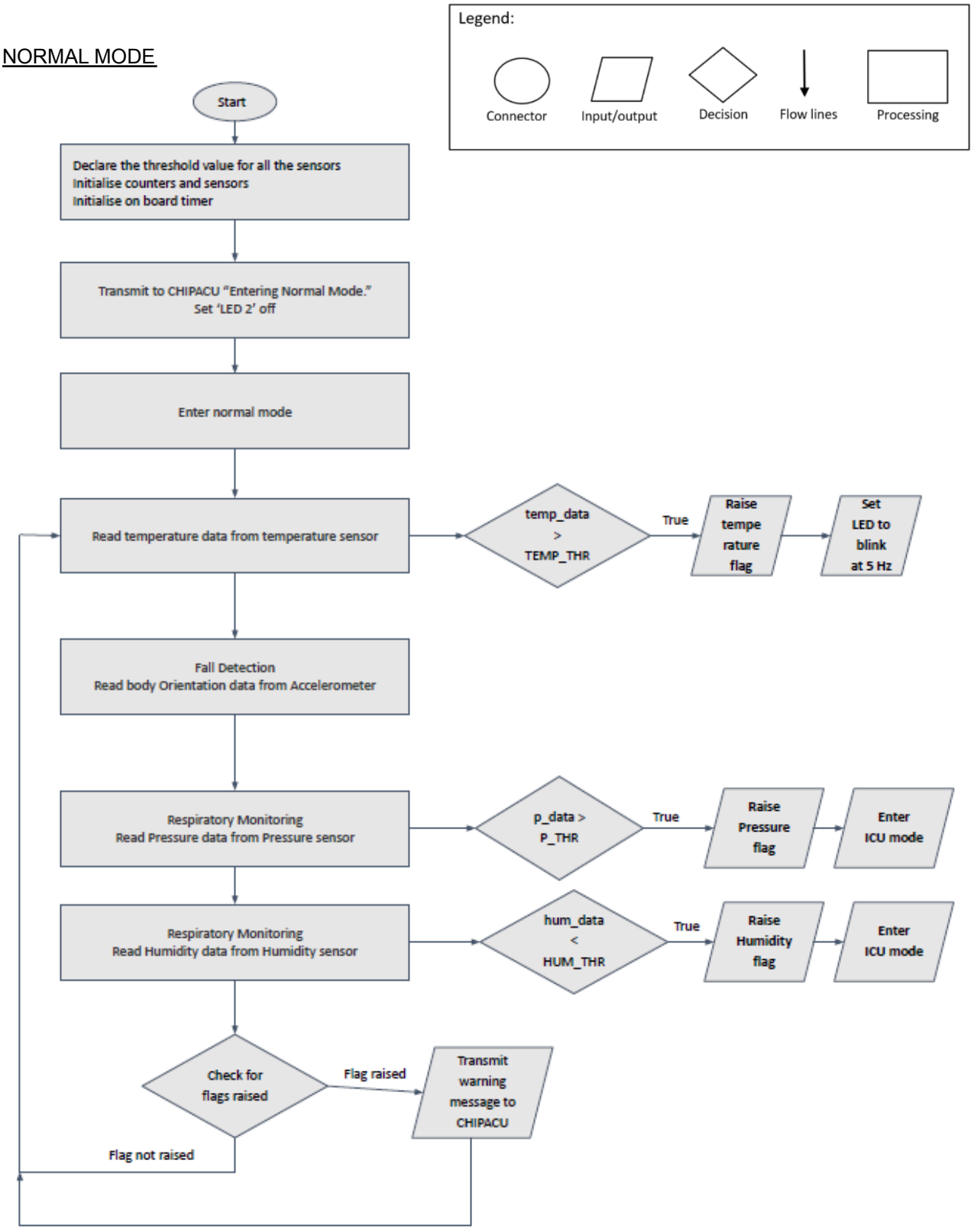
COPEMON has two modes of operation; Normal and Intensive Care(ICU) mode. Under each mode, COPEMON will capture data using different sensors and transmit to CHIPACU if certain conditions are met.

Sensor	Usage and Potential Implementations
Accelerometer	Monitors the posture of the patient, specifically for fall detection.
Magnetometer (only in ICU mode)	Monitors the orientation of the patient lying on the bed. Proper orientation is important in ensuring that other monitoring / life-saving equipment remains connected properly.
Pressure Sensor	Simulates the pressure of air in the patient's lungs. A change in pressure can be varied by varying the height at which the sensor is held.
Temperature Sensor	Monitor body temperature.
Humidity Sensor	Measures the relative humidity of the air passed into the patient's lungs.
Gyroscope (only in ICU mode)	Measures patient's movement; specifically to sense the patient's sudden twisting/twitching which could be an indication that the patient is in pain. A sudden movement can also cause issues with the other monitoring / life-saving equipment such as ventilators

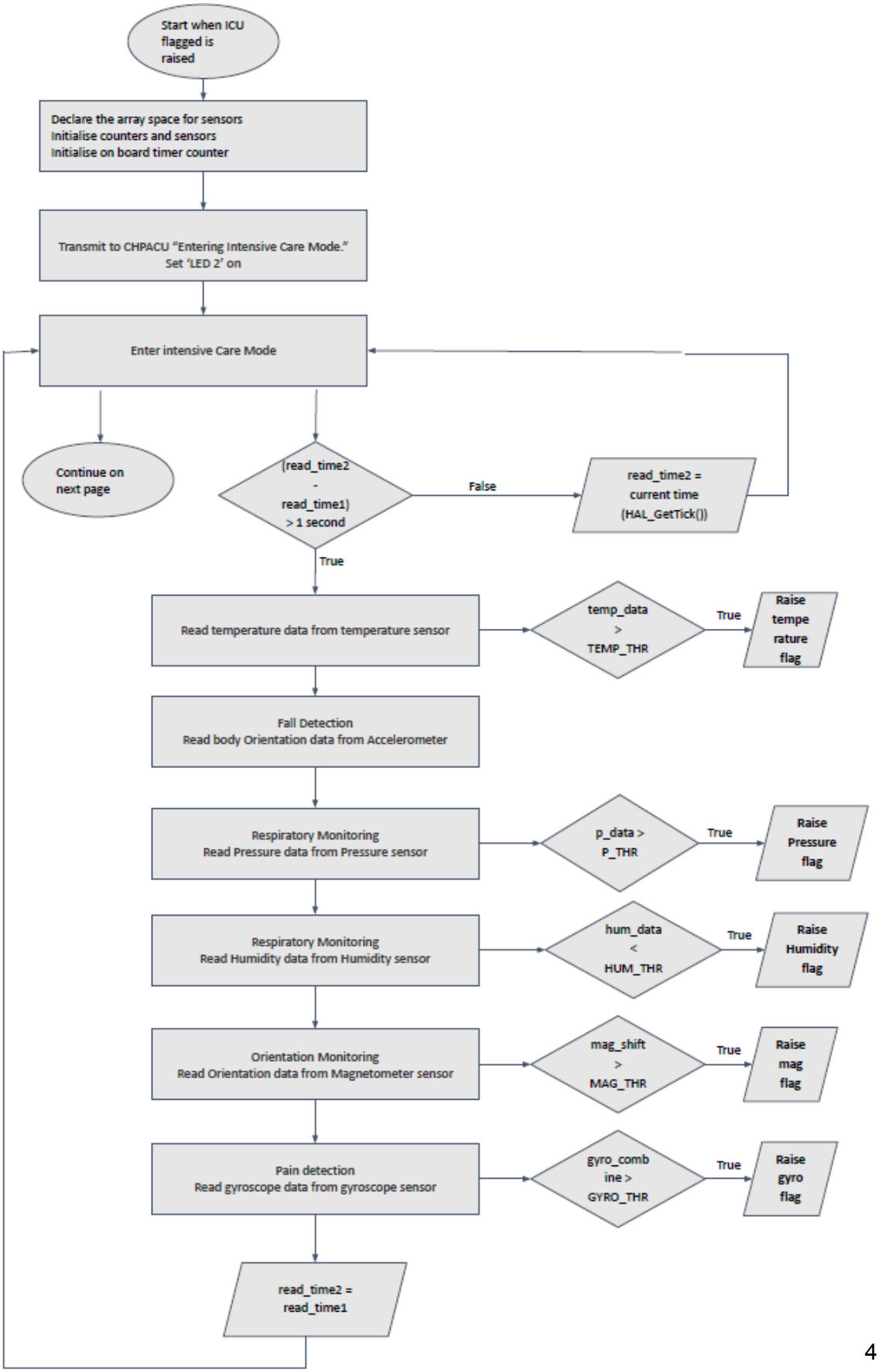
To ensure safety and well-being of the patients, threshold values have been set for each of the sensors to differentiate if the patient is well or requires medical attention. Should there be any threshold crossing detected, COPEMON will transmit warning messages to CHIPACU to alert the relevant personnel.

2. **Flowcharts**

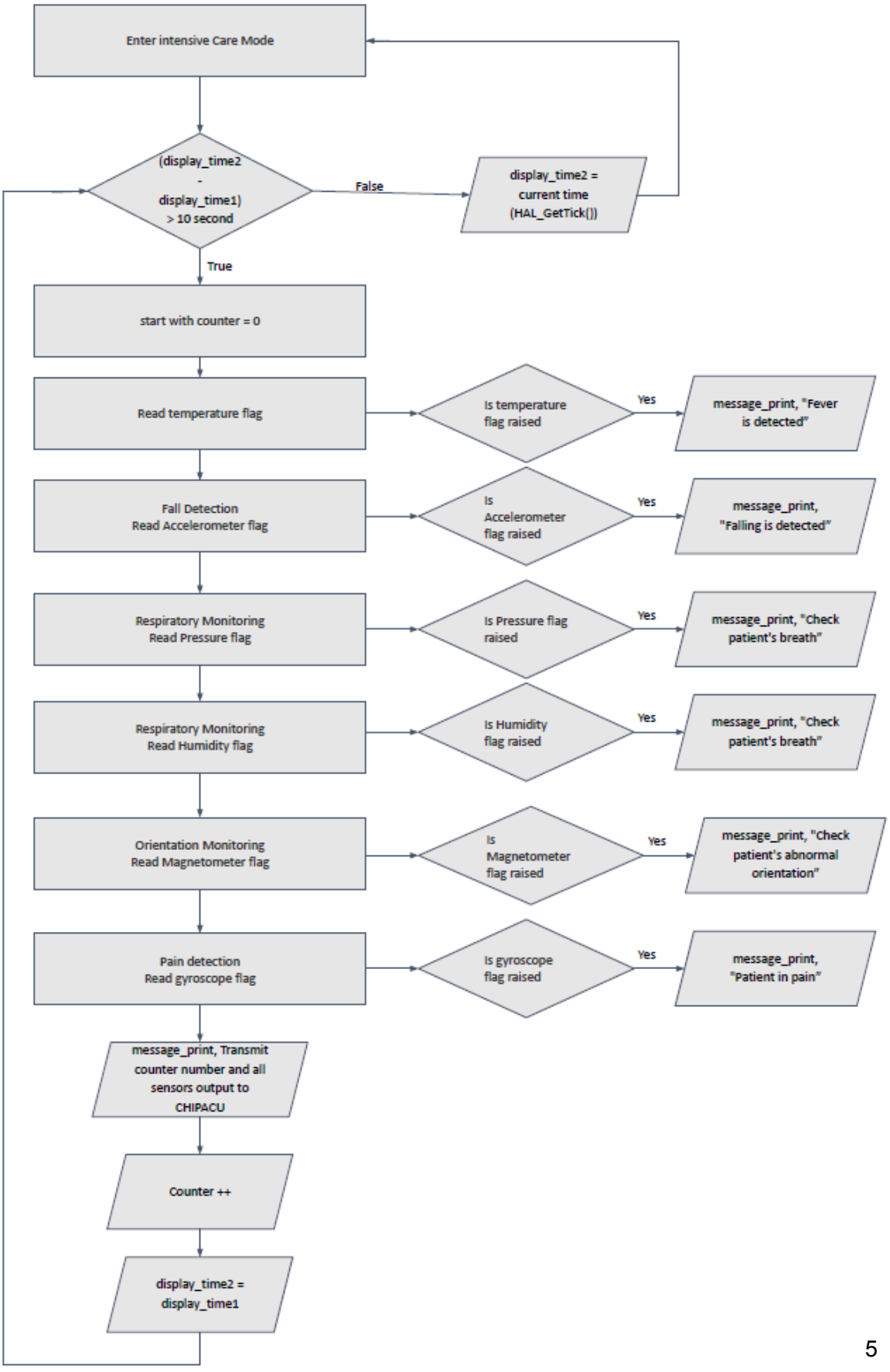
NORMAL MODE



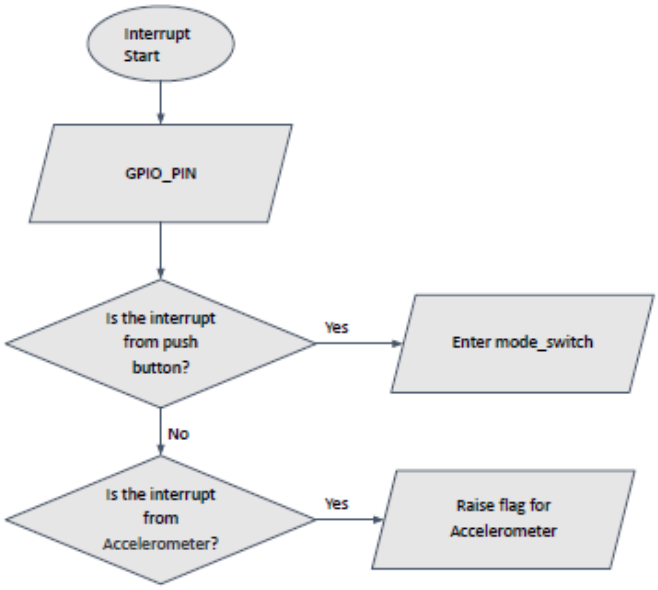
ICU MODE


Start when ICU flagged is raised
Declare the array space for sensors
Initialise counters and sensors
Initialise on board timer counter
Transmit to CHPACU "Entering Intensive Care Mode."
Set 'LED 2' on
Enter intensive Care Mode
Continue on next page
(read_time2 - read_time1) > 1 second
False
read_time2 = current time (HAL_GetTick())
True
Read temperature data from temperature sensor
temp_data > TEMP_THR
True
Raise tempe rature flag
Fall Detection
Read body Orientation data from Accelerometer
Respiratory Monitoring
Read Pressure data from Pressure sensor
p_data > P_THR
True
Raise Pressure flag
Respiratory Monitoring
Read Humidity data from Humidity sensor
hum_data < HUM_THR
True
Raise Humidity flag
Orientation Monitoring
Read Orientation data from Magnetometer sensor
mag_shift > MAG_THR
True
Raise mag flag
Pain detection
Read gyroscope data from gyroscope sensor
gyro_comb ine > GYRO_THR
True
Raise gyro flag
read_time2 = read_time1

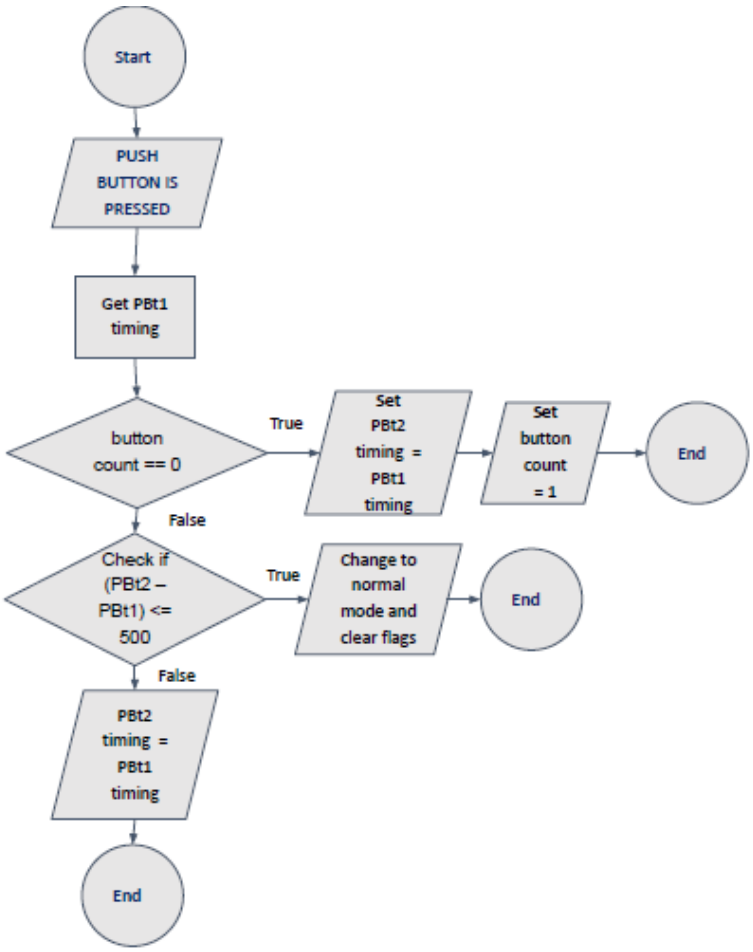
ICU MODE - cont'd



Interrupt (HAL_GPIO_EXTI_Callback)



Pushbutton double press (mode_switch)



3. **Detailed Implementation**

Overview

The system will first enter main, where all used variables, functions, device peripherals, and interrupts are being declared, initialised and configured.Afterward the system will enter into NORMAL_MODE() function, where all polling of sensors, data reading and transmission required under normal mode will be done. All data from relevant sensors are read by polling and warning flags are raised if data crosses threshold, with the exception of accelerometer which is done using device interrupt. When threshold crossing is detected, a warning message is sent to telemetry every 10s and the LED is set to blink at 5Hz or 200ms.

Under this mode, we will also monitor if a patient has difficulty breathing by checking if pressure or humidity readings exceed threshold. If there is threshold crossing detected, the system will send a warning message to telemetry and immediately enter into the ICU_MODE() function, where all requirements under Intensive care mode will be done.

Under intensive care mode, the LED is always on and the code polls the reading of sensor data every 1 second. if any threshold crossing is detected, the warning flag for the sensor(s) will be set. This is with the exception of the accelerometer, where the warning flag is raised using interrupt. Data readings of the sensors and any warning messages are also sent to telemetry every 10s. When the blue button on the board is pressed 2 times within 0.5s, the system will reset itself and return to normal mode.

Systick Interrupt

All the timing of warning messages and LED blinking are done using Systick interrupt to get an accurate timing.

```
if ( (current_time - start_time) > 200)
{   “insert code”
    start_time = current_time;
}
else
{ current_time = HAL_GetTick() }
```

HAL_Getick() function is used to obtain the system clock value. When the time difference is more than 200ms, the “insert code” will be executed and both start_time and current_time will “reset”. This is to prevent cases where t1>t2, resulting in a negative time difference. If the time difference is less than 200ms, the program will continue to update current_time (increasing value) until the time difference reaches 200ms.

Pushbutton Interrupt

To switch back to normal mode from intensive care mode, we created a function called mode_switch(). A variable called button_count is created to count the number of button presses. Systick is also used for timeout 0.5s after the first button press.
During the first button press, button_count is incremented by 1 and the start time, button_t1 is obtained using HAL_GetTick(). During the second button press, the current time, button_t2 is obtained and if the time difference between button_t1 and button_t2 is less than 0.5s, the system

will reset and return to normal mode. However if the time difference is more than 0.5s, the system will go into “timeout” and button_t1 and button_t2 will be “reset”.

Accelerometer (LSM6DSL) interrupt

To raise warning flags for the accelerometer, we made use of interrupts from LSM6DSL. LSM6DSL is connected to the GPIO EXTI1 pin and supports interrupts through the INT1 line. To either configure the interrupt or read if an interrupt is raised, we will write or read to the following addresses.

7.15 I2C addresses of modules used on MB1297

The *Table 3* displays the I^2C addresses (read and write) for the modules that are connected to the I2C2 bus.

Table 3. I^2C addresses for each module

Modules	Description	SAD[6:0] + R/W	I^2C write address	I^2C read address
HTS221	Capacitive digital sensor for relative humidity and temperature	1011111x	0xBE	0xBF
LIS3MDL	3-axis magnetometer	0011110x	0x3C	0x3D
LPS22HB	MEMS nano pressure sensor	1011101x	0xBA	0xBB
LSM6DSL	3D accelerometer and 3D gyroscope	1101010x	0xD4	0xD5
VL53L0X	Time-of-Flight ranging and gesture detection sensor	0101001x	0x52	0x53
M24SR64-Y	Dynamic NFC/RFID tag IC	1010110x	0xAC	0xAD
STSAFE-A100	-	0100000x	0x40	0x41

`LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW` is defined as 0xD4, hence `LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW+1` will be 0xD5.

Interrupt Configuration

1. To configure LSM6DSL as an interrupt, we need to set bit[7] of TAP_CFG to 1

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)

Figure 1: Enable basic interrupts

2. To configure LSM6DSL as FREE_FALL interrupt, set FF_DUR[4:0] to 00001 for 1 output data per reading. Set FF_THS[2:0] to 0 for setting threshold value to lowest; 156mg.

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*.

Figure 2: Configure FREE_FALL interrupt

3. To route FF interrupt to INT1, set MD1_CFG bit [5] to 1.

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)

The interrupts are then routed to GPIO Port D Pin11 or EXTI11 on the board. To determine if an interrupt has been raised, we can read WAKE_UP_SRC register bit [5].

9.24 WAKE_UP_SRC (1Bh)

Wake up interrupt source register (r).

Table 80. WAKE_UP_SRC register

0	0	FF_IA	SLEEP_ STATE_IA	WU_IA	X_WU	Y_WU	Z_WU

Table 81. WAKE_UP_SRC register description

FF_IA	Free-fall event detection status. Default: 0 (0: free-fall event not detected; 1: free-fall event detected)

3. **Significant problems encountered and solutions proposed**

Systick timing getting “stuck”

We observed that the sending of warning messages and timing of the LED blinking is incorrect. After debugging and observing the updated values of t1 and t2, we realised that at some point of time the value of t1>t2, which resulted in a negative time difference and created problems with the time loop. Hence after we include t1=t2 after every execution when if statement is true so that the starting time difference will always be zero, hence preventing inaccurate or negative calculation results.

Double press on Push-Button

When we first started with the push button, the issue encountered was that when the button was pressed, the counter for push-button_1 kept refreshing itself and could not be increased as we’ve started the code with [(PBt2 – PBt1) <= 0.5s]. Hence, leading to an endless loop where the timer for push-button_1 & push-button_2 is equal value. After some debugging and observation, we realised that we must change the logic of the flowchart and to start the code with a counter instead of a timer. Therefore, the logic of the flow chart was set to count the number of pressed in the push-button. However, we still found that there is an error within the code as the counter would forever be read as 1 instead of 2. To solve this issue, we introduce an additional conditional loop function to start initialising the counter. Hence, finally able to complete the double pressed interrupt.

4. **Conclusion and Feedback**

This project is very challenging for students who do not have much background in coding, there is a lot of information to absorb over a short span of time. This project pushed us to think out of the box and learn the importance of doing research and self-experimentation on our own. For example, we had to figure out how to use device interrupts from various sensors by information from user guides and datasheets. We also had to be courageous enough to even change the entire structure of our code when things fail to work. These are useful skills to help us in becoming independent thinkers as we cannot rely on our TAs or lab notes.

However, many of us could have done better if the lab assignment criteria had been carefully considered and released to us earlier so we have more time for research on what is expected of us instead of cramming it in the last 2 weeks. Given that we are part time students and this is also pair work, there is not a lot of time for us to work together to finish this project. The fact that the criteria keeps changing does not help either as those of us who had done the code earlier have to keep spending time editing it instead of using the time for meaningful research.

That being said, it did put our adaptability to the test, we learnt to restructure our logic flow and rewrite parts of the code quickly. Overall this project was a great exposure for us.