EE2028 Assignment2 Report

Group 1, Lab: Wednesday

Josiah Chua (A0238950X) , Teoh Jing Yang (A0164524H)

Table of Contents

Introduction & Objectives 1

System Design: Flowcharts 1

Detailed Implementation 6

Reflections 9

Conclusion 9

Introduction & Objectives

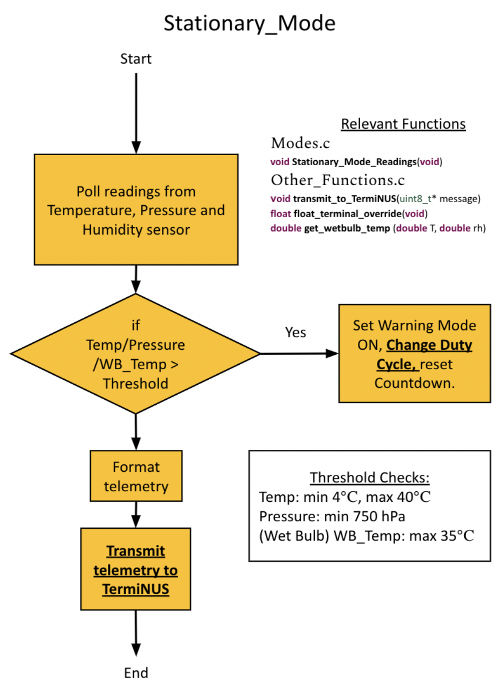
In EE2028 Assignment 2, the B-L475E-IOT01A board was used as a prototype for NUSpace, a manned space flight system. The basic design of NUSpace would minimally require interaction with sensor devices, data transmission to a remote terminal, the ability to operate in different modes, as well as the interactions with human controllers. This was modelled on the board using a STM32L475 microcontroller, with seven built-in sensors, a UART device, one LED and a USER PUSHBUTTON. The prototype was also required to operate in and transition between three modes: STATIONARY, LAUNCH and RETURN, each with specific requirements.

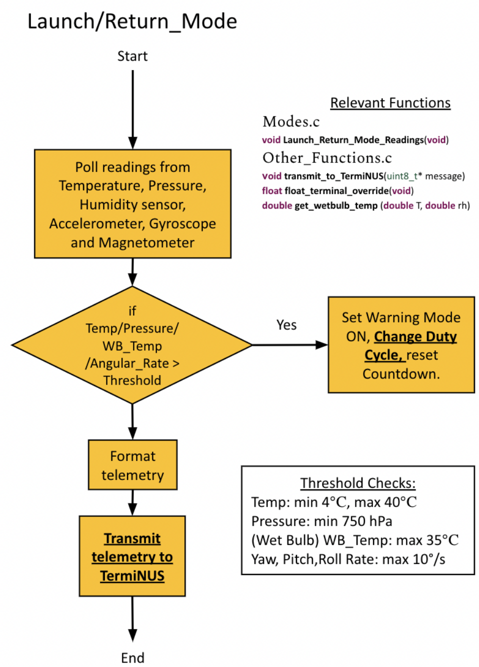
In this assignment, HAL and BSP library functions were used modularly to exploit the board's basic features, including SysTick and Sensor Interrupts. Modifications were made to template C files (such as **stm32l4xxit.c**) for fine-tuning sensor sensitivities & operating ranges, as well as for configuring event interrupts. Final deliverables include **modes.c** which determine device behavior within each mode, and **Other_Functions.c**, containing functions for UART-based communication, event interrupts, and LED toggling. Finally, an **IR sensor** was incorporated into the system (as enhancement), to trigger the automatic opening of the door out of the cabins.

System Design: Flowcharts

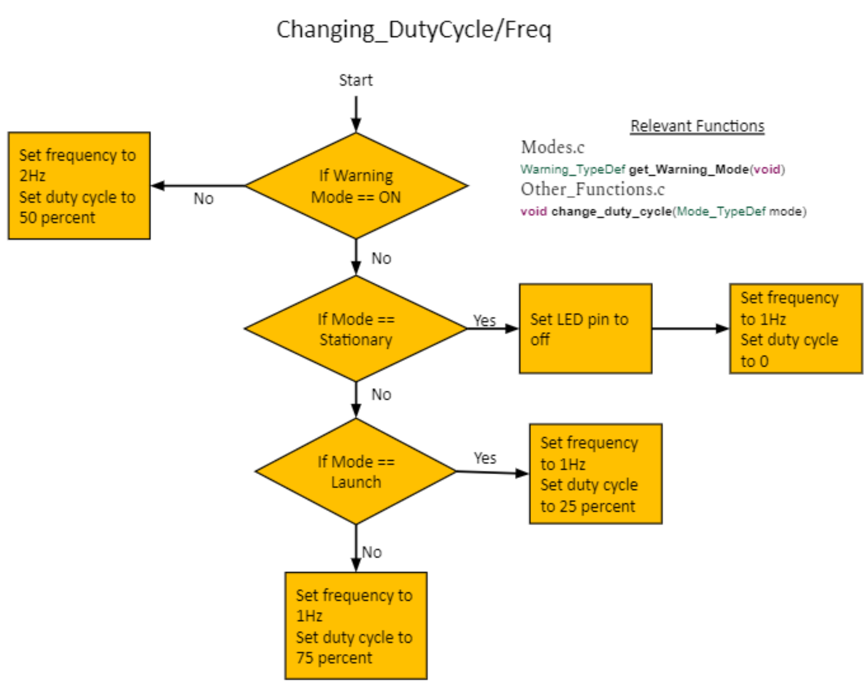
The system is best understood through three layers of abstraction, and in the following manner: bottom, top and middle. The bottom layer comprises standalone chunks detailing system behavior during a specific action / event: Stationary_Mode, Launch/Return_Mode, Changing_DutyCycle/Freq, Execute_Door_Movement, Door_Sensor_Interrupt, Auto_Close_Door, SysTick_Interrupt_Handler_LED, Stationary_Mode_Countdown, & FreeFall_Interrupt_Handler. These chunks often manifest as a combination of functions from **modes.c** or from **Other_Functions.c**. The individual instructions are assembled together to form the top layer: Main_Loop. Finally, the middle layer, containing Changing_Modes_PushButton and Changing_Modes_SysTick merges the top and bottom layer together coherently.

<u>Bottom Layer</u>: **Stationary_Mode**, **Launch / Return Mode**



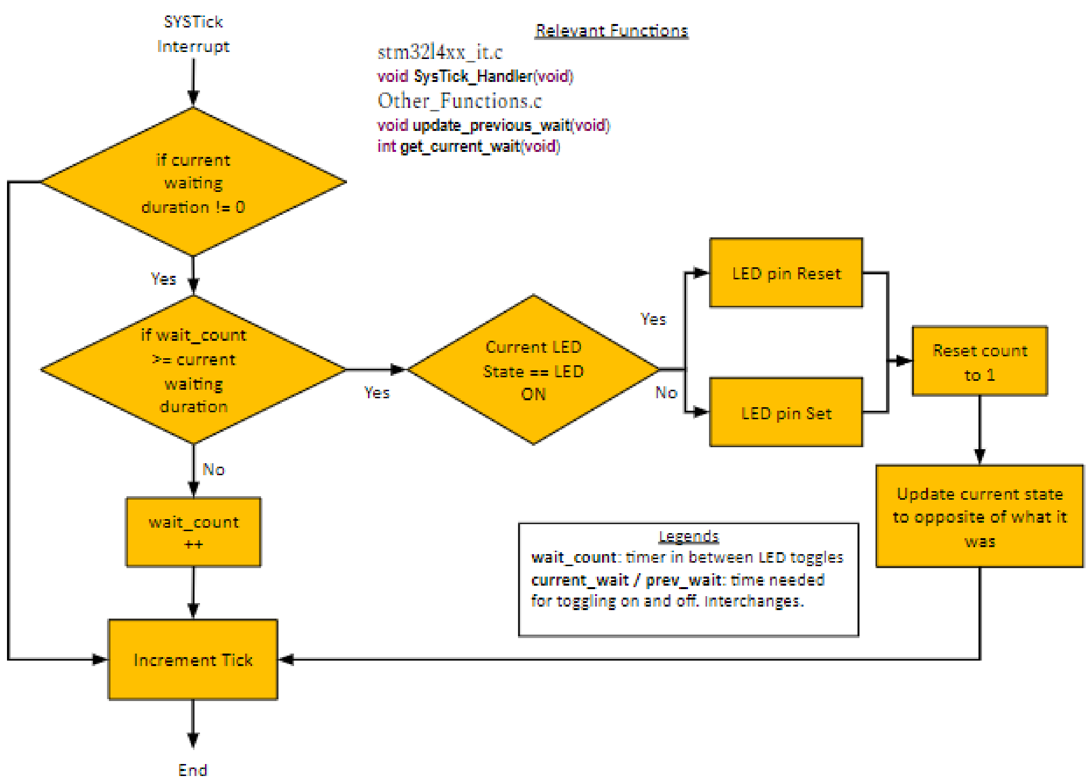


<u>Bottom Layer</u>: **Changing_DutyCycle/Freq**



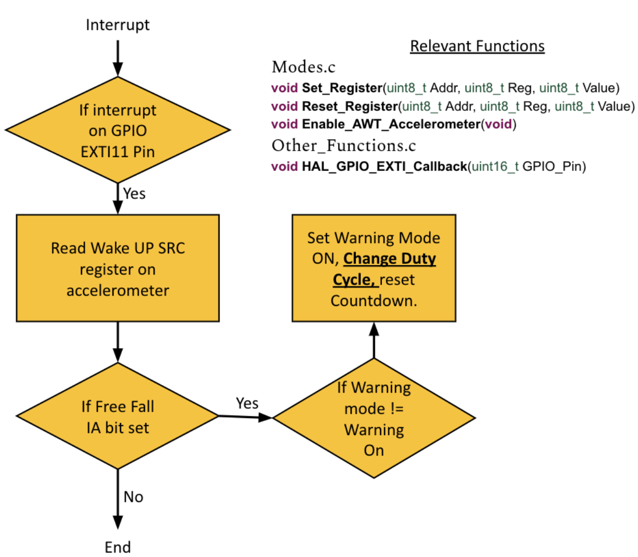
Bottom Layer: **SysTick_Interrupt_Handler_LED**

SysTick_Interrupt_Handler_LED

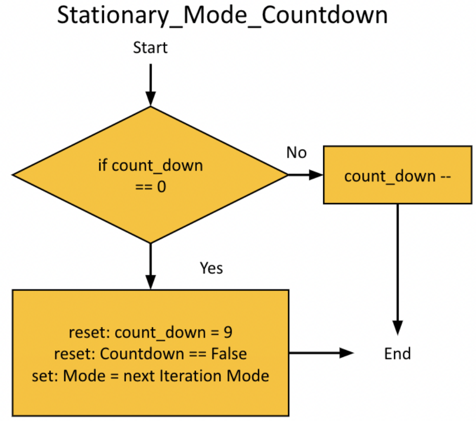


Bottom Layer: **FreeFall_Interrupt_Handler, Stationary_Mode_Countdown**

FreeFall_Interrupt_Handler

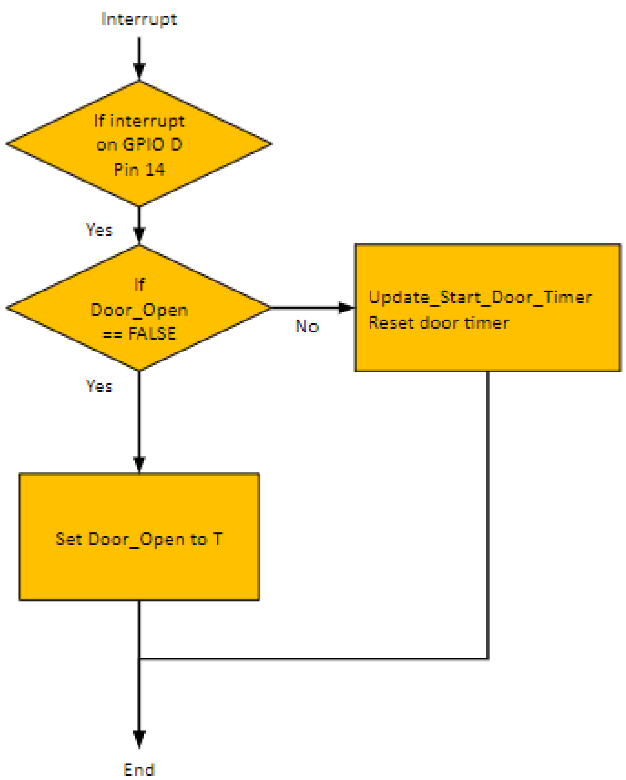


Stationary_Mode_Countdown

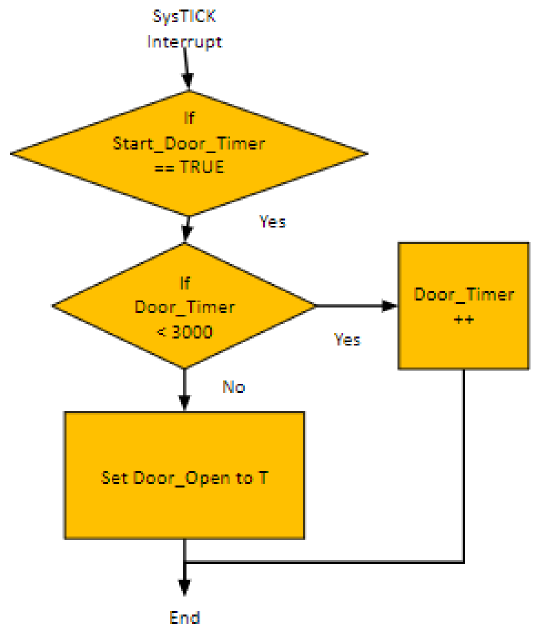


Bottom Layer: **Door_Sensor_Interrupt, Auto_Close_Door**

Door_Sensor_Interrupt

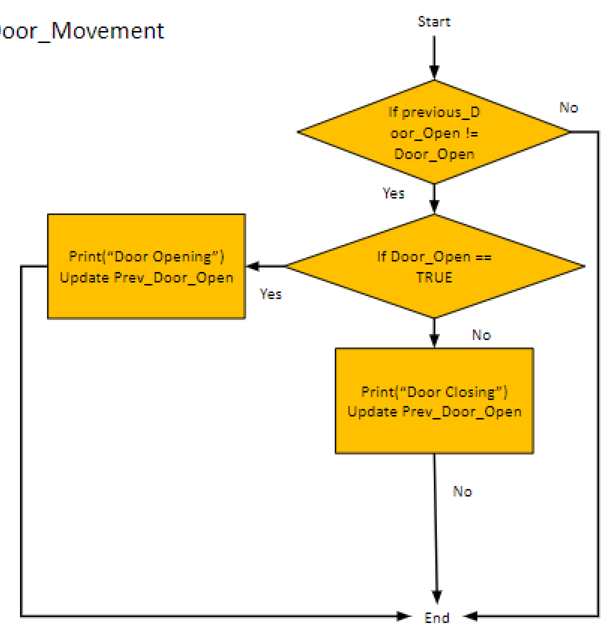


Auto_Close_Door



Bottom Layer: **Execute_Door_Movement**

Execute_Door_Movement



Relevant Functions

Modes.c

```
Boolean check_Door_Open(void);
Boolean check_prev_Door_Open(void);
void close_door(void);
void open_door(void);
void update_prev_Door_Open(void);
```

Other_Functions.c

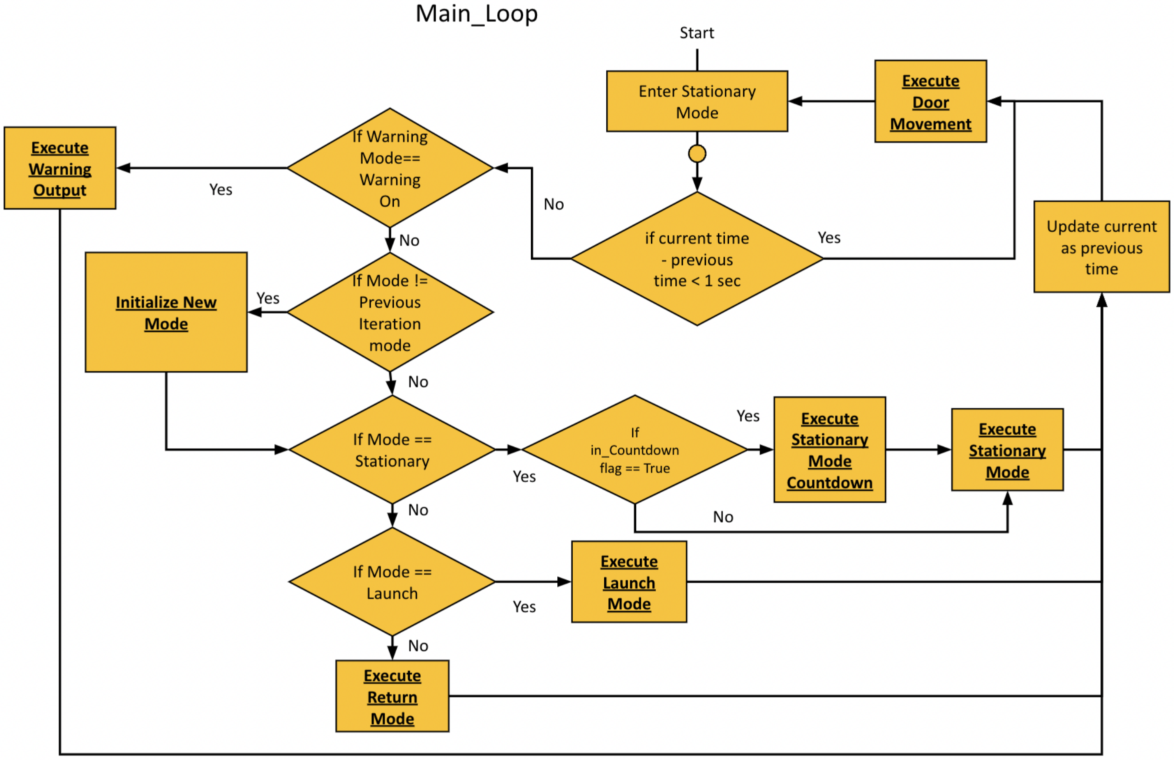
```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
```

stm32l4xx_it.c

```
void SysTick_Handler(void)
void reset_door_timer(void)
Boolean check_Start_Door_Timer(void);
void update_Start_Door_Timer(void);
```

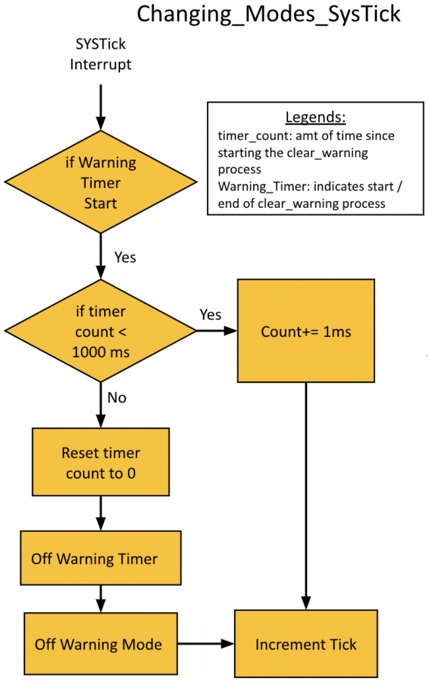
Top Layer: **Main_Loop**

Main_Loop

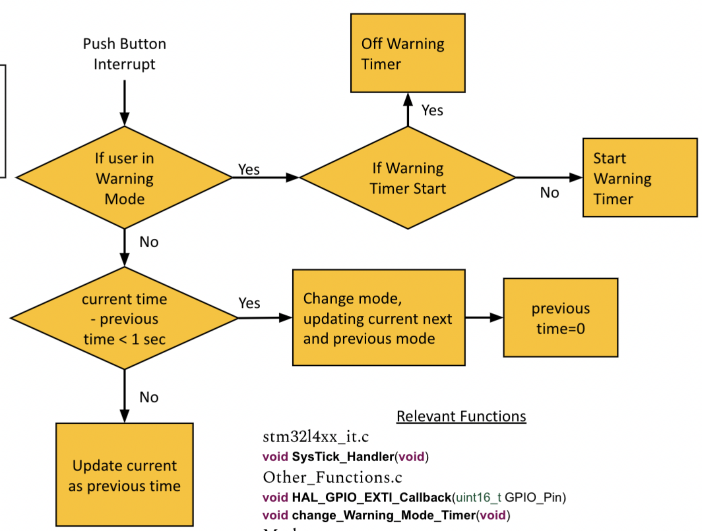


Middle Layer: **Changing_Modes_PushButton, Changing_Modes_SysTick**

Changing_Modes_SysTick



Changing_Modes_PushButton



Relevant Functions

stm32l4xx_it.c
`void SysTick_Handler(void)`
Other_Functions.c
`void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`
`void change_Warning_Mode_Timer(void)`
Modes.c
`void update_Warning_Mode(void)`

Detailed Implementation

STATIONARY_Mode, LAUNCH/RETURN_Mode

The primary mode sequences comprised polling, threshold checks, and telemetry transmission. During polling, the functions **void Stationary_Mode_Readings (void)** & **void Launch_Return_Mode_Readings (void)** call BSP functions such as BSP_PSENSOR_ReadPressure( ) to sample the sensors. Threshold checks are performed, and Warning_Mode is entered on fulfilling any trigger condition. Sensitivities and operating ranges of sensor devices are as per the table below. Standard UART methods were used for telemetry. During DEBUG mode, **float float_terminal_override(void)** was called to take human-readable float inputs from the terminal console to replace polled data.

Sensor	**Operating Characteristics**	**Threshold Checks & Justifications**
Accelerometer (LSM6DSL)	Temp Range: -40 to +85 deg Range: +- 4g Sensitivity: 0.122 mg/LSB BSP output: X mg = X / 100 ms^-2	Trigger Condition Free Fall External Interrupt (GPIOD11). Threshold Value: 156mg (tested at 500mg) Timing: 1sec (tested at 1/52sec) **To detect spacecraft losing power.**
Gyroscope (LSM6DSL)	Temp Range: -40 to +85 deg Range: 2000 degrees /s Sensitivity: 70 mdps/LSB BSP Output: Xmdps = X / 1000 dps Pitch (x-axis), Roll (y-axis), Yaw (z-axis)	Trigger Condition Angular rate (any axis) > 10 dps. **To capture any significant change in rocket trajectory.**
Magnetometer (LIS3MDL)	Range: +- 4 Gauss (FS1 = 0, FS0 = 0) “Sensitivity”: 6842 LSB / Gauss or 0.14mGauss/ LSB BSP Output: X mGauss = X / 1000 Gauss	NA. Threshold checks were not performed for the magnetometer as the magnetic fields of the earth behave differently further from the earth surface, which may render data from the magnetometer incompatible with its usual application - orientation.
Temperature Sensor (HTS221)	Range: 15 - 40 deg Celsius Accuracy: +- 0.5 deg Celsius Operating Temp: -40 to 120 degrees	Trigger Condition Temp < 4℃ or Temp > 40℃ **Humans are reportedly able to live indefinitely above 4℃. Max. temp threshold is placed for instruments onboard.**
Humidity Sensor (HTS221)	Range: 0 - 100% rH Sensitivity: 0.004% rH/LSB Humidity accuracy: ± 3.5% rH, 20 to +80% rH	Trigger Condition Wet bulb temp > 35℃ **Wet-bulb temperature tracks temperature at max humidity saturation. As a function of temperature and humidity, a threshold of ~35℃ is where human survival becomes questionable.**
Pressure Sensor (LSB22HB)	Range: 260 - 1260 hPa Sensitivity: 4096 LSB / hPa Operating Temperature: -40 - 85 deg	Trigger Condition Pressure < 750hPa **This reflects pressure at 8000ft, which is where humans experience altitude-induced ailments.**

If the system is in Stationary_Mode, the main loop checks for a countdown flag and runs the countdown program accordingly before the main polling in that scenario.

LED: Implementation of duty cycles and frequency with SysTick Interrupts

Using SysTick interrupts at every 1ms, running counters could be incremented, with the LED pin and internal waiting times being toggled when the counter exceeds waiting time. When any change in mode occurs, **void change_duty_cycle**(Mode_TypeDef mode) has to be executed to change the waiting time parameters accordingly based on frequency and duty cycle percentage. Additionally, state variables Current_LED_State were used to detect the current LED state, so that the control flow logic control using waiting times stay consistent through the transition.

Refer to flow diagrams for logic. Importantly, if the flag Warning_Mode = WARNING_ON is set, **change_duty_cycle** will set waiting time parameters to that of Warning_Mode. On the other hand, mode can only be set to STATIONARY, LAUNCH, or RETURN as states, which is used when exiting Warning_Mode.

Entering / Clearing Warning Mode and Changing Modes (with USERBUTTON Interrupt)

Warning Mode can be entered through any mode by the same sequence of three instructions, called through **void update_Warning_Mode**(**void**): (1) setting Warning_Mode = WARNING_ON, (2) changing the LED duty cycle accordingly and (3) resetting the In_Count_Down flag and count_down timer. The setting of Warning_Mode = WARNING_ON is picked up in the main loop, which then transmits the appropriate warning message to TermiNUS.

Resetting the Warning_Mode flag begins with **void HAL_GPIO_EXTI_Callback**(uint16_t GPIO_Pin) which is triggered by the USERBUTTON input. It sets a Warning_Mode_Timer = TIMER_START flag, and starts a warning_timer_countdown which increments with each SysTick Interrupt (1ms). If the Warning_Mode_Timer flag stays set when warning_timer_countdown reaches 1000(ms), the "Clear Warning" instructions are executed: Warning_Mode is reset so that the main loop no longer transmits warnings; Warning_Mode_Timer flag & warning_timer_countdown are reset for reuse. Executing **change_duty_cycle**(mode) recovers the waiting time parameters of the previous mode!

A second press of USERBUTTON input before warning_timer_countdown reaches 1000(ms) would reset the flag Warning_Mode_Timer <- TIMER_STOP, which would prematurely terminate the process of clearing Warning Mode.

On the other hand, if Warning_Mode_Timer = TIMER_STOP, the Push Button Interrupt would instead compare if the previous interrupt (represented by previous_time) occurred 1000ms before current_time, causing a change of modes if the condition is true. Because mode represents only the internal states STATIONARY, LAUNCH, & RETURN, and determines the system behavior alongside the Warning_Mode flag, updating of mode suffice to cause an eventual change in system parameters during the execution of main_loop. Indeed, main_loop compares mode with another internal state variable Check_Mode to determine if the instructions for updating system behavior (such as changing duty cycle and transmitting the change in modes) via **void init_new_mode**(Mode_TypeDef new_mode) needs to be executed.

However, as the mode transition from STATIONARY to LAUNCH requires an additional countdown, a In_Count_Down flag is set instead of updating mode. The setting of In_Count_Down (= TRUE) is detected in main_loop, which triggers the decrement of counter count_down from 9 to 0 each time. Finally, when count_down = 0, both count_down (to 9) and In_Count_Down (to FALSE) get reset, and mode finally gets updated to LAUNCH. We repeat again that entering Warning Mode would involve resetting both count_down and In_Count_Down, which stops the countdown process.

Accelerometer: Implementation of Free Fall External Interrupt

The Free Fall interrupt is transmitted directly from accelerometer to the microcontroller when it detects (based on threshold and timing parameters) that the device is in free-fall. This occurs when acceleration is between ±threshold value for a time more than the timing parameter chosen. The integration of the free fall interrupt into the main program is straightforward: when an interrupt is sent from the accelerometer, the interrupt handler executes the callback function **void HAL_GPIO_EXTI_Callback**(uint16_t GPIO_Pin) which then brings the system into Warning Mode via **void update_Warning_Mode**(**void**). See Above. This theoretically allows us to detect when the spacecraft loses power during its trajectory.

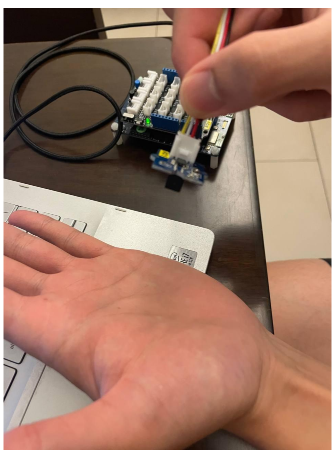
However, some initial effort was needed to set up the interrupt. We enabled it by setting its priority to 1 (which is just lower than Systick interrupt), cleared any pending status and enabled the IRQ handler on NVIC. GPIO D pin 11 was configured to pull down and rising mode to detect high voltage from the accelerometer during free fall. Next, we enabled the free fall interrupt by setting the Free Fall Bit in the Tap_CFG register, followed by the bit registers for timing and threshold values. Finally we set the route free fall interrupt bit in the MD1_CFG register to route the interrupt to the int1 pin of the accelerometer as it is connected to GPIOD P1. The corresponding code can be found in **void Enable_AWT_Accelerometer**(**void**).

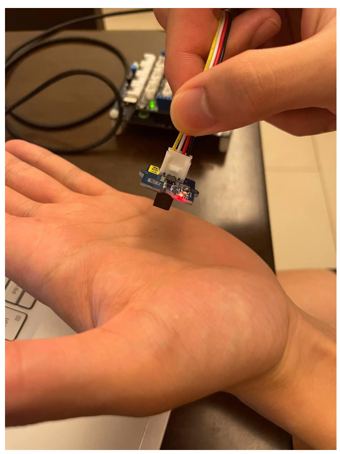
Enhancement: Implementation of Automatic Door Opening via IR Sensors

As enhancements, we incorporate an Infrared Reflective sensor into our system, which acts as a control for an automated door.The IR sensor is able to detect objects at a maximum of 4.5cm away. Automatic doors value-add to a spacecraft because under the conditions of zero-gravity, it can be difficult for astronauts to exert force for opening / closing doors, or pressing buttons. This is resolved by placing one's hand near the sensor. The sensor was integrated into the system so that the door stays open when an object is detected, which happens if a person is holding the door for more than 1 person. The door would automatically close if the sensor no longer detects any object for 3 seconds. In this project, the opening and closing of the door will be represented by a message sent to TermiNUS, given by: "Door Opening"/"Door Closing".

When the IR sensor detects an object within 4.5cm in front of it, it sends a low voltage through the digital pin. Hence, we connected the IR sensor to ARD D2 (GPIOD P14) and enabled it as an EXTI interrupt with a NVIC priority of 2. The pin mode was set to detect both rising and falling and the pin was pulled up since the interrupt would be a low voltage from the IR sensor. When an interrupt due to the IR sensor occurs, this enabsles the interrupt handler, which then changes the Door_Open flag to TRUE.

The main loop constantly compares the Door_Open flag and prev_Door_Open flag. When the former is TRUE and the latter FALSE, the message "Doors Opening" will be sent to TermiNUS, representing the opening of doors. The prev_Door_Open Flag gets updated to TRUE. This allows "Door Opening" to be printed only once to indicate the door opening and that it stays open. When the user's hand leaves the IR sensor range, the sensor generates another interrupt that sets the Start_Door_Timer flag to TRUE, and starts the countdown of 3 seconds, after which the Systick Interrupt will change the Door_Open Flag to FALSE. This will trigger a "Door Closing" to be sent to TermiNUS in the main loop. If users activate the sensor anytime between the opening of the doors and the and the closing, it will restart the door timer such that the door will remain open, mimicking the functional use of automatic doors. Refer to flow diagrams for details.





Reflections

Problems and Solutions: During the project, one significant problem encountered was in the enabling of the free fall interrupt in the accelerometer registers. As the datasheet does not explicitly explain what the register bits are required to enable the interrupt, we had to look through all the registers that mentioned freefall to see if they have to be enabled. This created a problem as we initially did not route the free fall interrupt to the int1 pin of the accelerometer, due to the misconception that the interrupt would enable simply by "enabling the interrupt" and setting the thresholds. However we realized that the interrupts do not go to int1 by default and the set-up worked after making that modification.

Issues & Suggestions: For the further extensions, we were given different sensors and peripherals to try and incorporate into our system. However most of these most did not have datasheets online (eg, 5 way switch). Hence those that needed I2C communication were very hard to implement as we did not know what the addresses were for I2C communication and how one could enable the different functions.

Conclusion

In conclusion, the B-L475E-IOT01A board was successfully utilised to model the controller program for a spacecraft. Through the effective use of global state variables, we achieved the basic design. Through the proper configurations of external interrupts, we were able to add a free-fall detection scheme and automatic door function to the system.