Table of contents

2. Introduction

In this assignment, we will be looking at a manned space flight system. We shall refer to this system as **NUSpace**. This system sends data periodically to a server known as **NUSCloud**.

NUSpace has three modes of operation: Stationary, Launch and Return modes, and will be transmitting to NUSCloud if certain conditions are met. Additionally, other device interfacing is required for NUSpace, which shall be described in the following pages. The stationary mode is the mode in which the rocket is still in its launchpad. Lauch mode is the mode in which the rocket is launched towards space. Return mode is the mode in which the space shuttle mounted on the rocket is detached and returns to land on earth.

It is assumed that the base board is a smaller prototype version of **NUSpace**, with several output devices to help in the debugging during development. The XBee RF module is assumed to a be a low powered wireless communication device that sends collected data to **NUSCloud**.

We are required to interface with various devices on the Base Board for data capture and data transmission. The reasons behind the use of the three main sensors are indicated in the table below.

Sensor	**Usage and Potential Implementations**
Accelerometer	Accelerometer has an extremely wide range of applications. For NUSpace, the accelerometer is assumed to be mounted on the launch system such that the Z direction faces the launch. Furthermore, the trajectory is assumed to be perfectly vertical. The accelerometer can be used to detect orientation of the system.
Light Sensor	Light sensor is used to simulate a radar, where the intensity increases as the obstacle is closer when the space shuttle comes back to land.
Temperature Sensor	The temperature module on NUSpace is used to monitor the temperature of its fuel tank, as a safety/precautionary measure to contain any possible fires quickly and effectively.

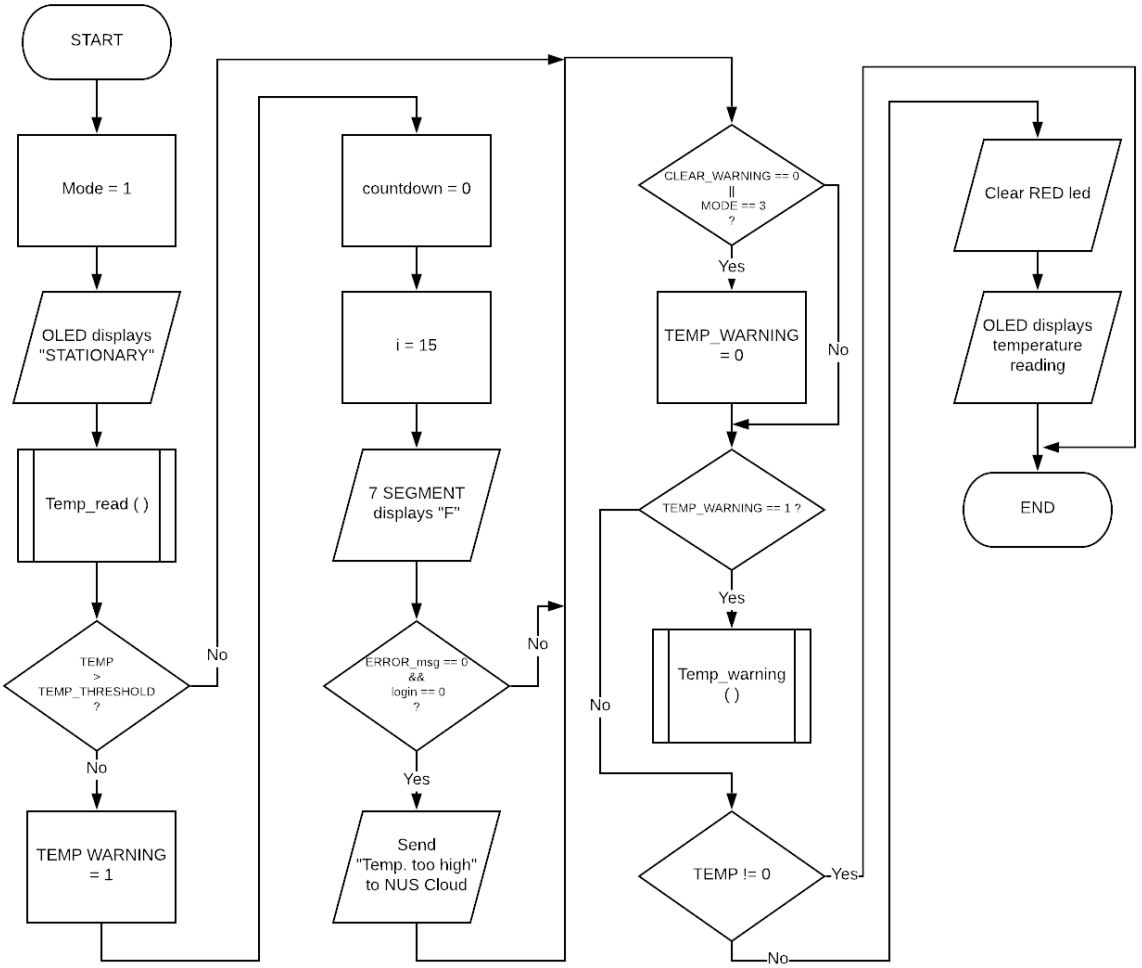
3. Objectives

After completing assignment 2, we will:

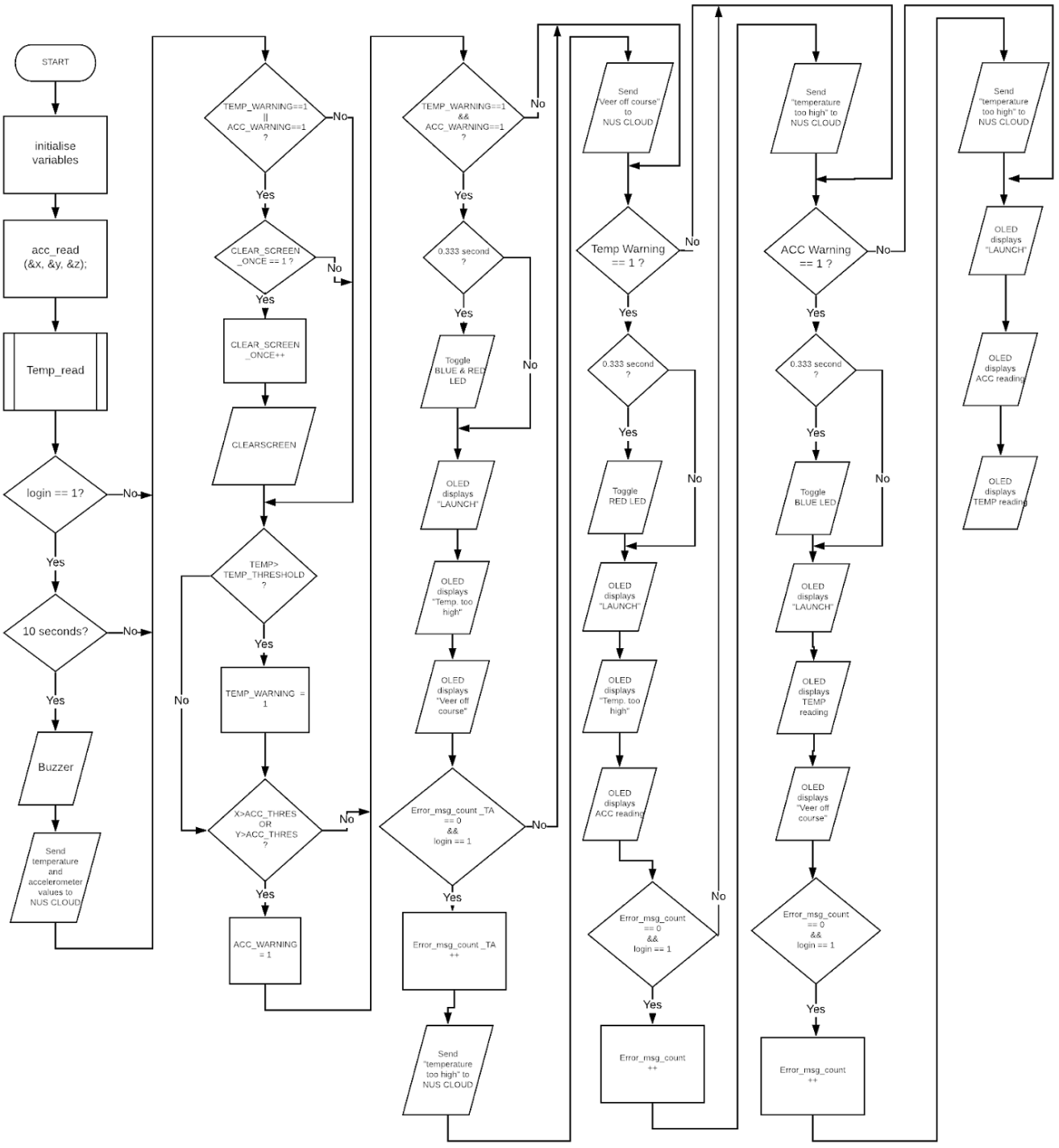
- be able to apply system design approaches, such as using flowcharts, to design embedded applications
- understand the interfaces between microcontrollers and peripherals
- have the ability to develop C embedded programming controller based applications

4. Flowcharts

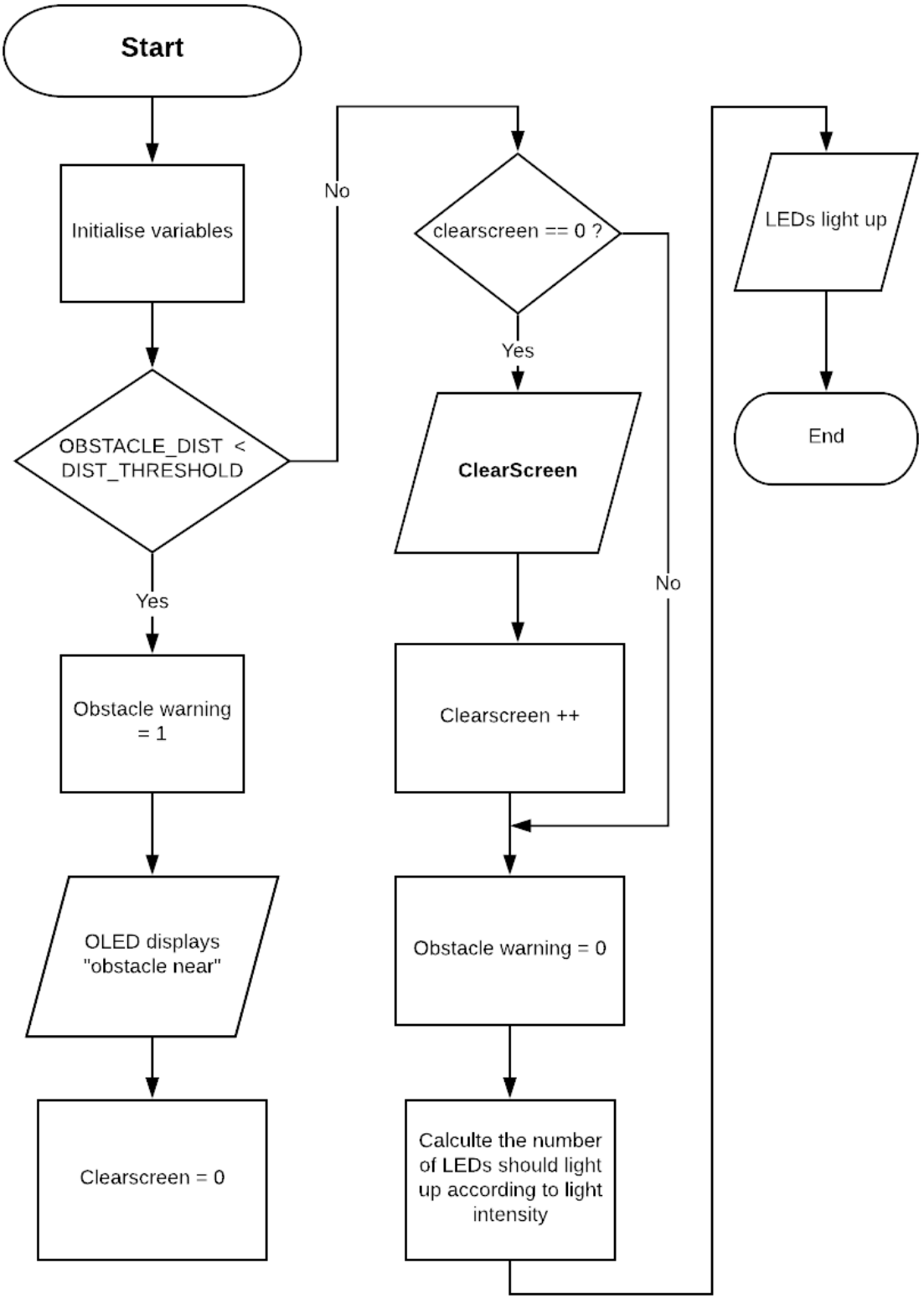
STATIONARY function



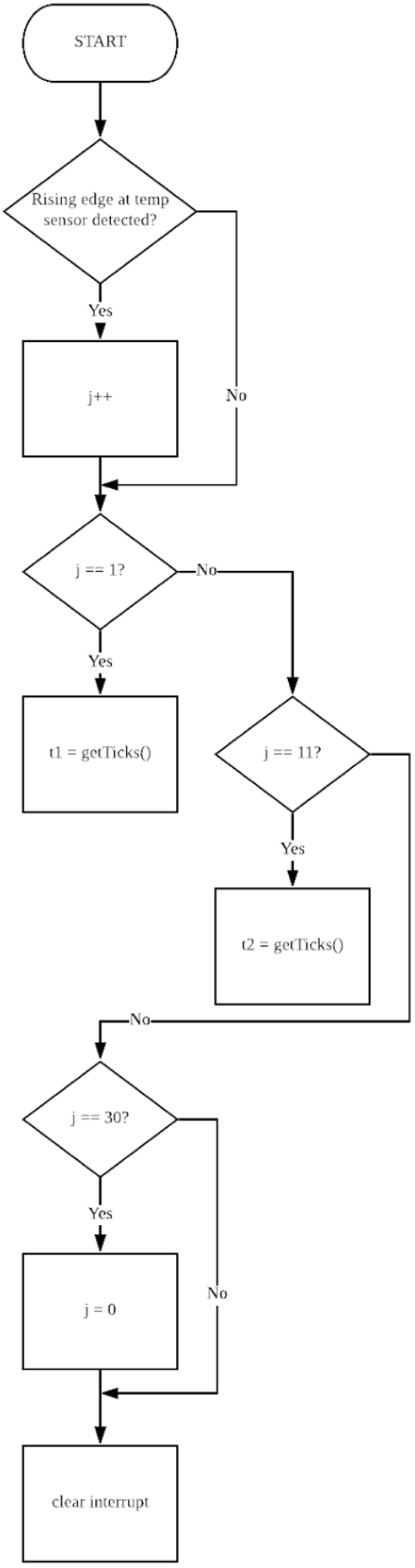
LAUNCH function



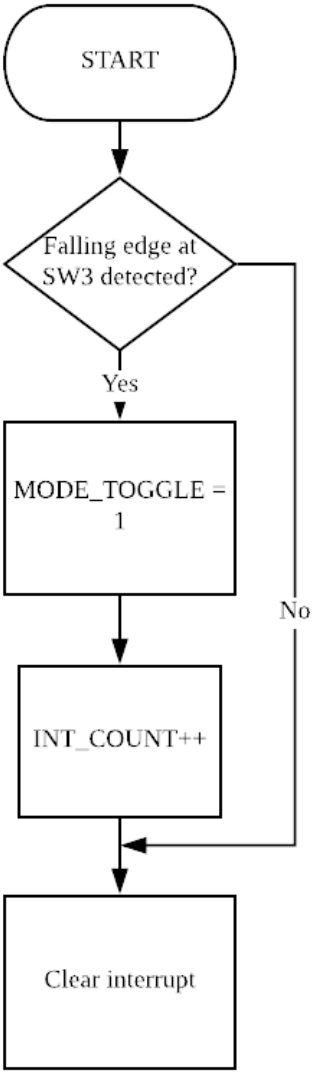
RETURN function



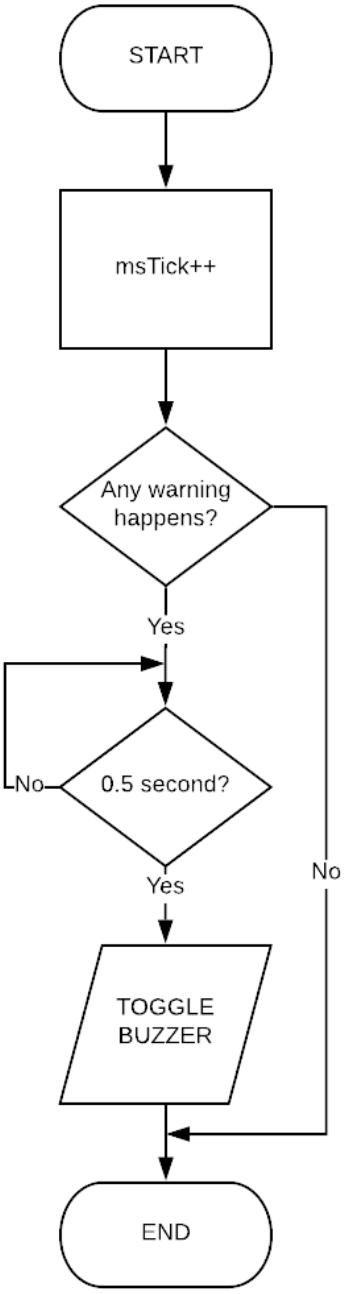
EINT3_IRQHandler



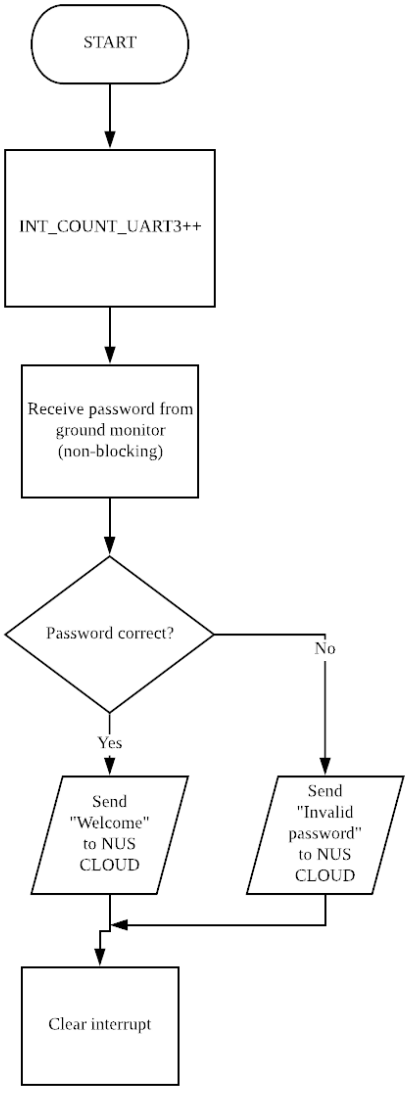
EINT0_IRQHandler



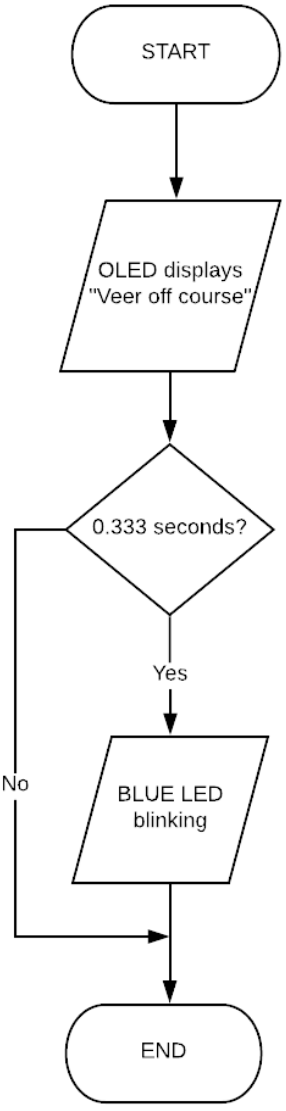
Systick_Handler



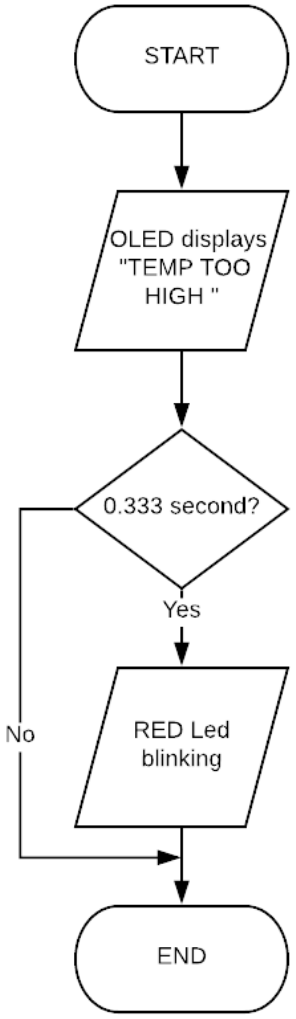
UART3_IRQHandler



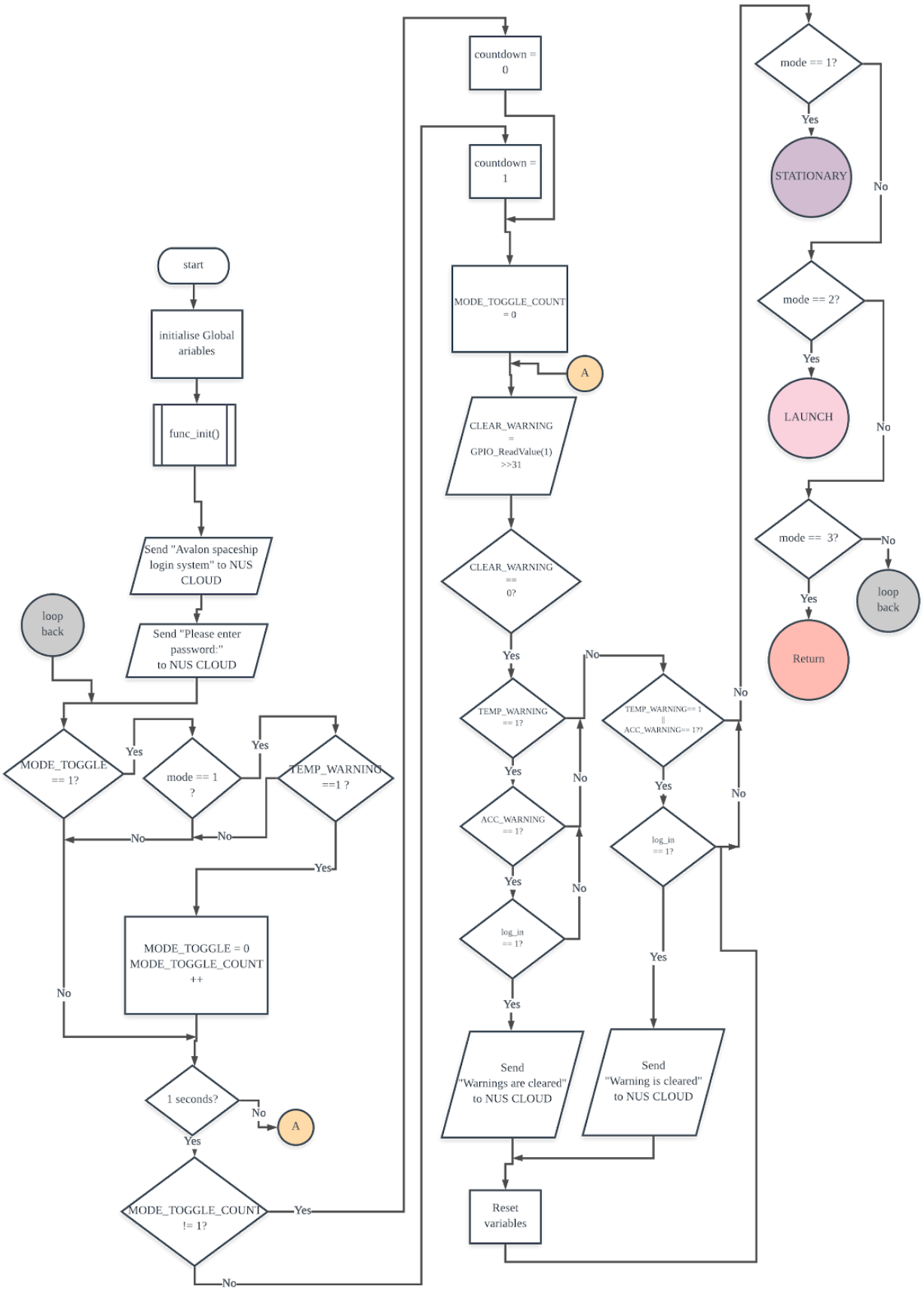
ACC_WARNING_



TEMP_WARNING_

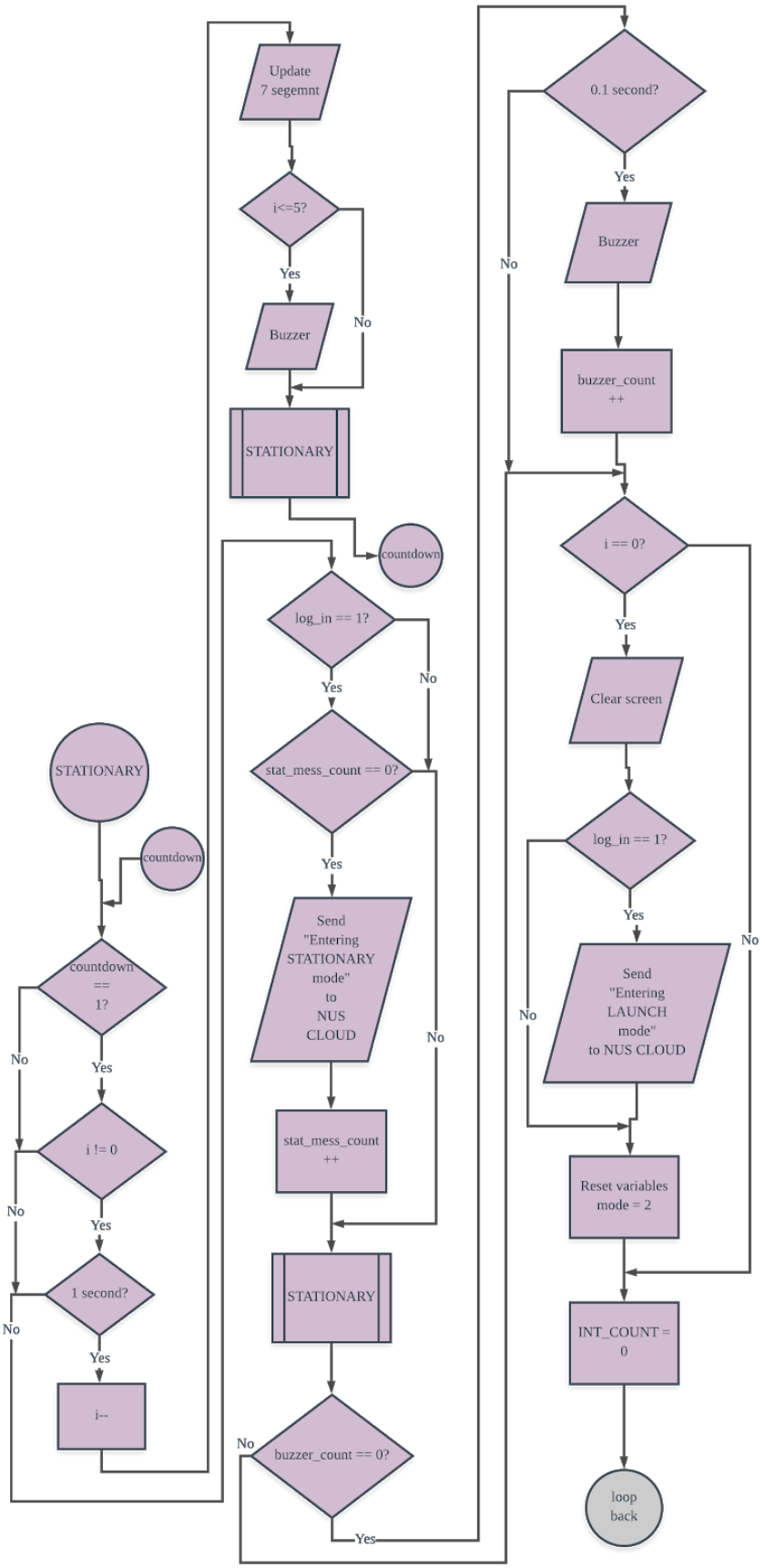


int main


start
initialise Global ariables
func_init()
Send "Avalon spaceship login system" to NUS CLOUD
Send "Please enter password:" to NUS CLOUD
loop back
MODE_TOGGLE == 1?
Yes
mode == 1 ?
Yes
TEMP_WARNING ==1 ?
No
No
Yes
MODE_TOGGLE = 0
MODE_TOGGLE_COUNT ++
No
1 seconds?
No
A
Yes
MODE_TOGGLE_COUNT != 1?
Yes
No
countdown = 0
countdown = 1
MODE_TOGGLE_COUNT = 0
A
CLEAR_WARNING = GPIO_ReadValue(1) >>31
CLEAR_WARNING == 0?
Yes
No
TEMP_WARNING == 1?
Yes
No
ACC_WARNING == 1?
Yes
No
log_in == 1?
Yes
No
TEMP_WARNING== 1 || ACC_WARNING== 1??
Yes
No
log_in == 1?
Yes
No
Send "Warnings are cleared" to NUS CLOUD
Send "Warning is cleared" to NUS CLOUD
Reset variables
mode == 1?
Yes
STATIONARY
No
mode == 2?
Yes
LAUNCH
No
mode == 3?
No
loop back
Yes
Return

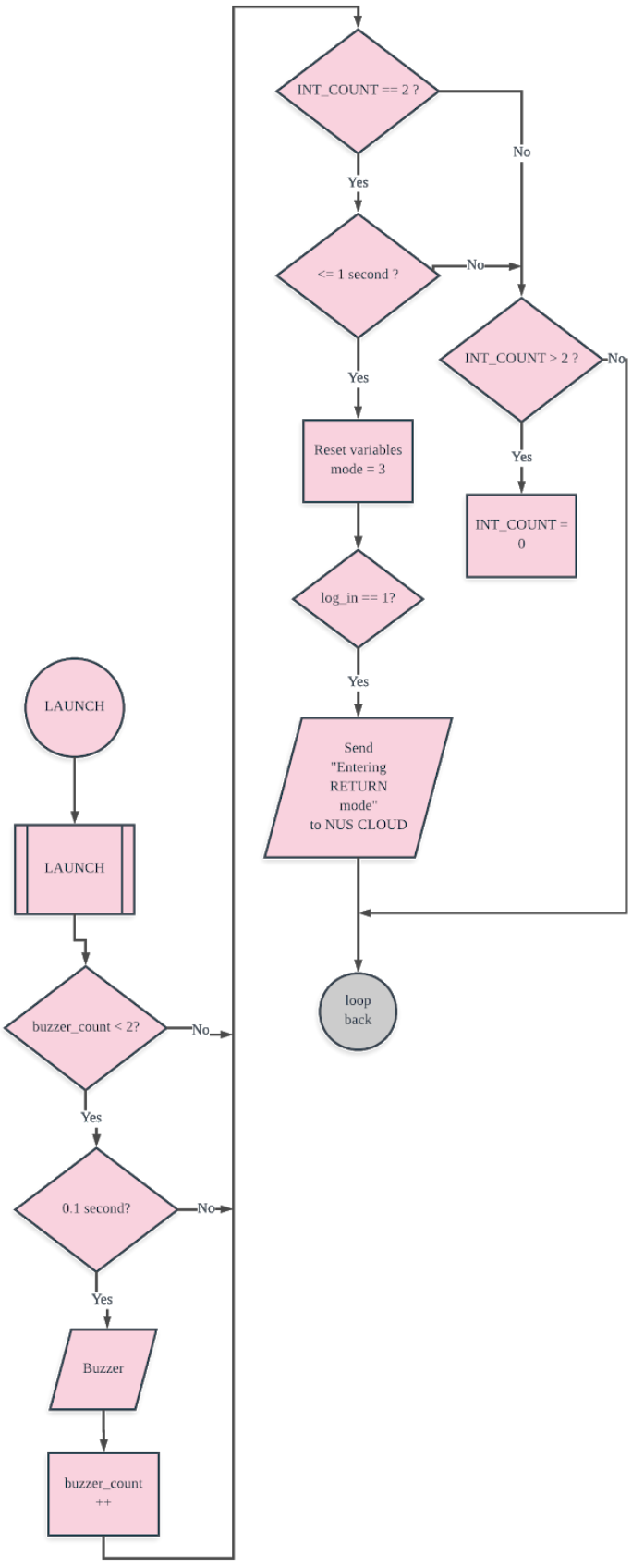
STATIONARY MODE

(Please refer to int main)



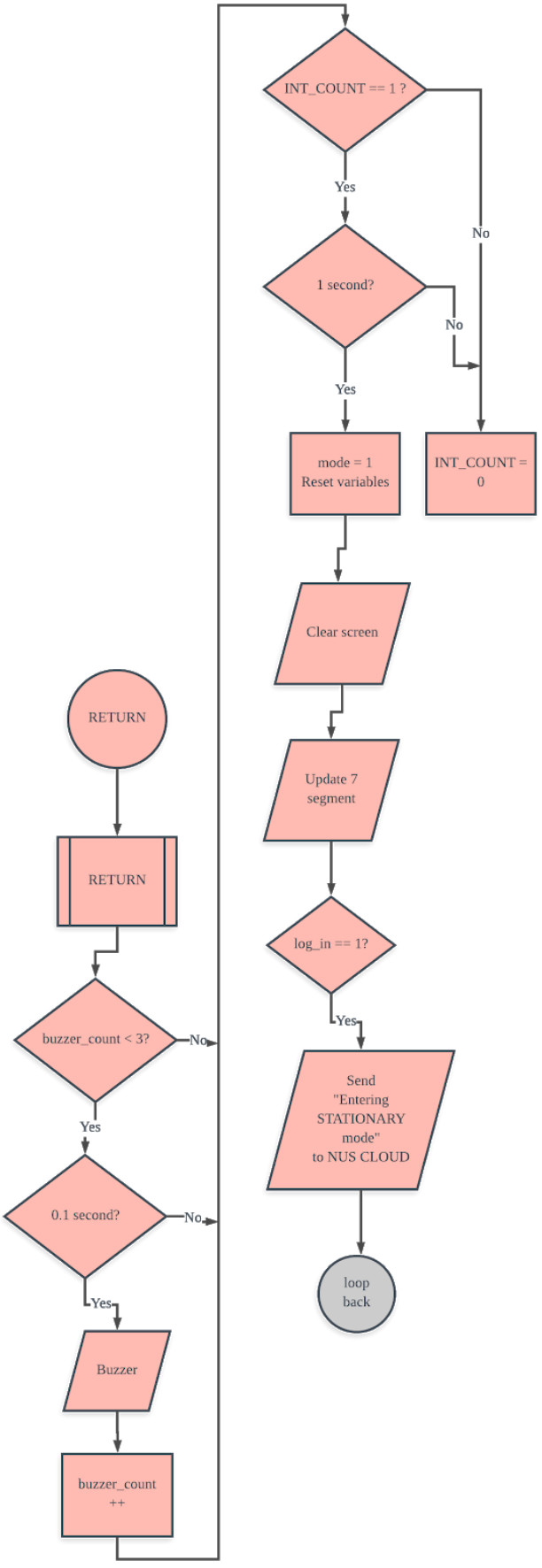
LAUNCH mode

(Please refer to int main)



RETURN mode

(Please refer to int main)



5. Detailed implementation steps

5.1 Program structure

To reduce the complexity and readability of the program, the codes for STATIONARY, LAUNCH and RETURN modes are segmented into several functions, and are integrated with conditional loop in the main function.

```
int main(){

while(1){


STATIONARY();

---------Button conditions----
------------countdown----------
-----TEMP_Warning condition-----

LAUNCH();

---------Button conditions----
---------Reset conditions-----

RETURN();

---------Reset conditions----


}
}
```

5.2 OLED

Owing to the need for clearing the oled screen in between every mode transition and in the event of warnings, different approaches had been implemented.

Clearn screen during mode transition

Conditional loops have been added to ensure the screen clearing process is only done once at the beginning of every modes.

```
if(INT_COUNT == 2){
        ini_t_launch_2 = getTicks();
        if(ini_t_launch_2-ini_t_launch<=1000){

                INT_COUNT = 0;
                Return = 1;
                Launch = 0;
                getTick_once = 0;

                TEMP_WARNING = 0;
                ACC_WARNING = 0;
                f_WARNING = 0;
                oled_clearScreen(OLED_COLOR_BLACK);
                GPIO_ClearValue(2,1);
                GPIO_ClearValue(0,1<<26);
```

```
if(i == 0){
    oled_clearScreen(OLED_COLOR_BLACK);
    if(log_in == 1){
    msg = "Entering LAUNCH Mode \r\n";
    UART_Send(LPC_UART3, (uint8_t *)msg , strlen(msg), BLOCKING);
    }
    t_mode_sound = getTicks();
    sig_count = 0;
    ini_t_TA = getTicks();//

}
```

Clearn screen in the event of warnings

For temperature and accelerometer warning, the warning status on oled are cleared only when CLEAR_WARNING(SW4) is 0.

```
if(CLEAR_WARNING == 0 ){
            if(TEMP_WARNING == 1 && ACC_WARNING == 1 && log_in == 1){
                msg = "Warnings are cleared \r\n";
                UART_Send(LPC_UART3, (uint8_t *)msg , strlen(msg), BLOCKING);
            }else if((TEMP_WARNING == 1 || ACC_WARNING == 1) && log_in == 1){
                msg = "Warning is cleared \r\n";
                UART_Send(LPC_UART3, (uint8_t *)msg , strlen(msg), BLOCKING);
            }

            TEMP_WARNING = 0;
            ACC_WARNING = 0;
            MODE_TOGGLE = 0;
            countdown = 0;
            clear_screen = 0;
            error_message = 0;
            error_message_TA = 0;
            GPIO_ClearValue(2,1);
            GPIO_ClearValue(2,1<<6);
            oled_clearScreen(OLED_COLOR_BLACK);

        }
```

As for obstacle warning status, it is cleared only when the counter condition, “clear_screen == 0” is satisfied. Again, this is to ensure the screen is only cleared once.

```
if(clear_screen==0){
    oled_clearScreen(OLED_COLOR_BLACK);
    clear_screen++;
}
OBST_WARNING = 0;
```

5.3 7-segment

Countdown

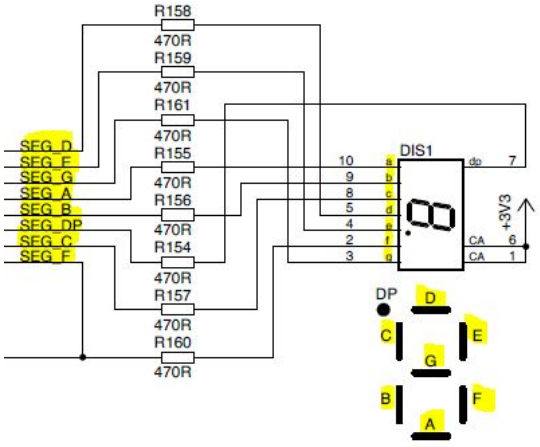
When SW3 is pressed within 1s, the value displayed on 7-segment is updated approximately every 1s when i value is decreasing in the while(1) main loop.

```
if(getTicks()-ini_t_7seg >= 1000){
    ini_t_7seg = getTicks();
    if(i != 0){
        i--;
        led7seg_setChar(charaters[i],TRUE);
```

180 degree flipped 7 segment

To flip the 7 segment 180 degree for better readability, raw mode is enabled and the respective hexadecimal values for different characters from 0 to F are figured out through pin mapping.

```
uint8_t charaters[16] ={0x24,0x7D,0xE0,0x70,0x39,0x32,0x22,0x7C,0x20,0x30,0x28,0x23,0xA6,0x61,0xA2,0xAA};
                        0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```



5.4 RGB

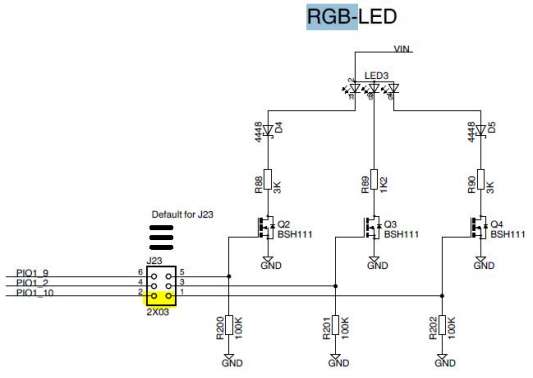
Blinking in an interval of 333ms

To achieve the RGB blinking effect when warning occurs, a conditional loop is used to check whether the time reaches 333ms and update the time once the condition is satisfied. In addition, a variable RRGB is used to toggle the RGB.

```
if(getTicks()-ini_t_R>=333){
    ini_t_R = getTicks();
    RRGB = ~RRGB;
    GPIO_ClearValue(2,(~RRGB&1)<<0);
    GPIO_SetValue(2,(RRGB&1)<<0);
}
```

Physical connection

Due to the internal connection of OLED and Green LED, Green LED will be affected when OLED is in use. Therefore the highlighted connection is removed in order to avoid the interference between the green LED and OLED.



5.4 Audio speaker

Indication of successful message delivery

To indicate the different modes and completion of message delivery through sound, Playnote function is used to beep distinct times of sound at the beginning of different modes and once the messages is sent to NUS CLOUD.

```
if(getTicks()-ini_t_TA>=10000){
    ini_t_TA = getTicks();
    playNote(1912,250);
    sprintf(display_T,"Temp : %2.2f degrees; ACC X : %2.2f, Y : %2.2f \r\n",TEMP,x_g,y_g);
    UART_Send(LPC_UART3, display_T , strlen(display_T), BLOCKING);
}
```

indication of modes

Once the system enters STATIONARY mode, the speaker will beep once.

```
STATIONARY();

if(buzzer_count==0){
    if(getTicks()-init_t>=100){
        init_t = getTicks();
        playNote(2272,250);
        buzzer_count++;
    }
```

In LAUNCH mode the speaker will beep twice at the beginning.

```
LAUNCH();

if(buzzer_count<2){

    if(getTicks()-init_t>=100){
        init_t = getTicks();
        playNote(2024,250);
        buzzer_count++;
       }
```

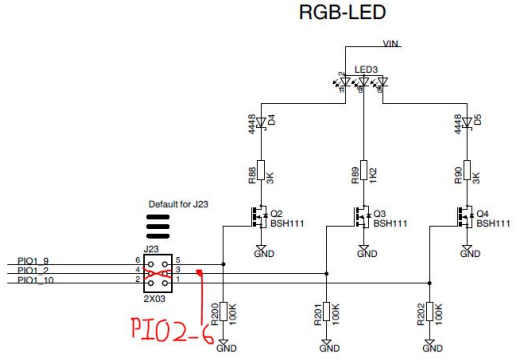
At the beginning of RETURN mode, the speaker will beep for three times

```
RETURN();

if(buzzer_count<3){
    if(getTicks()-init_t>=100){
                init_t = getTicks();
                playNote(1912,250);
                buzzer_count++;
        }
```

Physical connection

The connection is modified through shorting the input of blue LED and the PWM output, PIO2_6 with a physical jumper.



5.5 SysTick handler

Update msTick in every 1ms

This is used to increase a counter by 1 at every 1 ms so as to give a real time reference to the system. It is the most important one, hence it is prioritized at the highest priority.

Generate a frequency and duty cycle for the speaker

SysTick_Handler function is used to generate a frequency and duty cycle for the speaker .

```
void SysTick_Handler(void)
{
    msTicks++;
    if(TEMP_WARNING==1 || ACC_WARNING==1 && CLEAR_WARNING != 0 && mode != 3 || OBST_WARNING == 1){
    if (sound_flag >= 0 && sound_flag <= 500)
    {
        if (sound_toggle ==2)
        {
            GPIO_SetValue(0, 1<<26);
            sound_toggle = 0;
        }
        else
        {
            GPIO_ClearValue(0, 1<<26);
            sound_toggle++;
        }
    }
    else if (sound_flag >= 1000)
    {
        sound_flag = 0;
    }
    sound_flag++;
}
}
```

Buzzing frequency

Buzzing duration

5.6 SW3

EINT0 interrupt

To improve the sensitivity of SW3, it is used with EINT0. A counter is used in the interrupt function in order to keep track of the interrupt frequency, so that the transition between different modes is achievable in the program.

```
void EINT0_IRQHandler(void)
{

    if ((LPC_GPIOINT->IO2IntStatF>>10)& 0x1)
    {
        MODE_TOGGLE = 1;
        INT_COUNT = INT_COUNT +1;
    }

    LPC_GPIOINT->IO2IntClr = 1<<10;
    LPC_SC -> EXTINT = 0x1;
}
```

Ideal case implementation

1. STATIONARY to LAUNCH

MODE_TOGGLE is a variable used in main function, and it will be set to 1 when there is an interrupt detected due to the change of state in SW3. MODE_TOGGLE_COUNT is used to keep track of the frequency of SW3 is pressed, and it will be reset to zero if the yellow highlighted conditions are not met.

```
if(MODE_TOGGLE == 1 && mode == 1 && TEMP_WARNING != 1){
    MODE_TOGGLE_Count++;
    MODE_TOGGLE = 0;//To reset MODE_TOGGLE
}

if(getTicks()-PO_init_t >= 1000){//PO:press once
        PO_init_t = getTicks();
        if(MODE_TOGGLE_Count != 1){
            countdown = 0;
            MODE_TOGGLE_Count = 0;
        }else{
            countdown = 1;
            MODE_TOGGLE_Count = 0;
        }
}
```

2. RETURN to STATIONARY

INT_COUNT is a variable used in EINT0 function, which is used to count the times when there is an interrupt occurs due to SW3. In RETURN mode, the transition to STATIONARY mode will only be enabled if the yellow highlighted conditions are achieved.

```
if(INT_COUNT == 1){
        if(getTick_once == 0){
        ini_t_launch = getTicks();
        getTick_once++;
        }
}

if(INT_COUNT == 2){
                ini_t_launch_2 = getTicks();
                if(ini_t_launch_2-ini_t_launch<=1000){
                                        ---------
                                        ---------
                                        ---------

        }else{
                INT_COUNT = 0;
                getTick_once = 0;
        }
}

if(INT_COUNT>2){
        INT_COUNT = 0;
}
```

5.7 Temperature sensor

EINT3 interrupt

```
void EINT3_IRQHandler(void)
{
     if (LPC_GPIOINT->IO0IntStatR>>2)

    {
            j++;
            if(j == 1){
                t1 = getTicks();
            }
            else if(j==11) {
              t2=getTicks();
              }
            if(j==20)//give the program some computation time
            {
                j=0;
            }
            LPC_GPIOINT->IO0IntClr = 1<<2;
    }
}
```

The time taken for eleven rectangular wave periods is obtained using EINT3 interrupt. The j value is the number of times that temperature interrupt has occurred. It is only reset at the twentieth period as some computation time is needed.

Temperature calculation

The time differences t2-t1 is then used to calculate the temperature. The formula used is as highlighted in the below picture.

```
void Temp_int(void)
{
    float temp =0;
    temp=abs(t2-t1);
    TEMP =((temp*100.0)/160.0)-273.15;
}
```

Note: *The temperature, in °C, may be calculated as follows:*

$$T(°C) = \frac{\text{period}(\mu s)}{\text{scalar mulitplier}(\mu s/°K)} - 273.15°K$$

Physical connection

The physical connection is modified to connecting TS1 to VDD and TS0 to GND.

TS1	TS0	SCALAR MULTIPLIER (µs/°K)
GND	GND	10
GND	V_{DD}	40
V_{DD}	GND	160
V_{DD}	V_{DD}	640

5.8 UART

UART interrrupt

UART 3 is used for receiving password data from log-in system. The counter, INT_COUNT_UART3 is used in the UART3 interrupt function to count the number of times interrupt has occurred, which allows the password to be verified character by character.

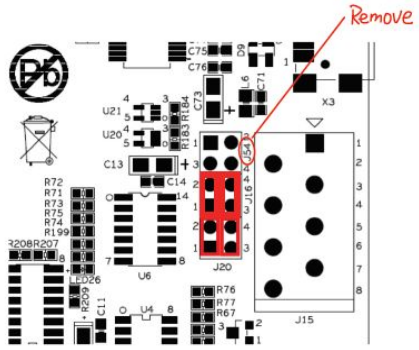
```
void UART3_IRQHandler(void){

    INT_COUNT_UART3++;
    UART_Receive(LPC_UART3, &data, 1, NONE_BLOCKING);

    if(INT_COUNT_UART3 == 1){
        if(data == '2'){
            count_log_in++;
        }
    }else if(INT_COUNT_UART3 == 2){
        if(data == '0'){
            count_log_in++;
        }
    }else if(INT_COUNT_UART3 == 3){
        if(data == '2'){
            count_log_in++;
        }
    }else if(INT_COUNT_UART3 == 4){
        if(data == '8'){
            count_log_in++;
        }
    }
```

Physical connection

To configure UART 3 to receive wireless data, the jumpers at j54 are removed and those jumpers that highlighted in red are connected accordingly.



5.9 Accelerometer

Value conversion

Due to the initial offset in the accelerometer, it has to be calibrated to get an accurate reading. The method used to get the initial reading of the accelerometer when the board is laid flat. To get the correct scale, the accelerometer reading should be subtracted with the offset value and divide by 64.0, as the default g-Range of accelerometer is 2g.

```
acc_read(&x, &y, &z);
x_g = (x-xoff)/64.0;
y_g = (y-yoff)/64.0;
```

Table 5. Configuring the g-Select for 8-bit output using Register $16 with GLVL[1:0] bits

GLVL [1:0]	g-Range	Sensitivity
00	8g	16 LSB/g
01	2g	64 LSB/g
10	4g	32 LSB/g

5.10 Light sensor

Algorithm for lighting up LED proportionally with light intensity

In order to calculate how far the obstacle is further away from the threshold and convert to the number of LEDs should light up, a formula is developed. To obtain the number of LEDs should light up, the value of light intensity and the maximum light intensity is subtracted and multiplied by 16. The float number is then converted to integer. By using a while loop, left shifting and “bitwise OR” operation, the

total number of LEDs is converted to the total number of logic ’1’ in the 16 bits at different bit position. The 16 bits value is then substituted in to the pca9532_setLeds function.

```
NUM_LED = (light_read()/MAX_LIGHT_INTENSITY)*16.0;
int TOT_NUM_LED = round(NUM_LED);//Convert float to decimel
i = 0;
LED_ARRAY = 0;

while(i<TOT_NUM_LED){
    LED_ARRAY |= 1<<i;
    i++;
}
pca9532_setLeds(LED_ARRAY,0xFFFF);
```

Light Range

To increase the detection range of the light sensor, the range is set to 4000.

```
light_setRange(LIGHT_RANGE_4000);
```

5.11 Priority setting

In order to overcome the contradictions between different interrupts, priority are set.

```
NVIC_SetPriorityGrouping(5);
NVIC_SetPriority(SysTick_IRQn,0x00);
NVIC_SetPriority(UART3_IRQn,0x40);
NVIC_SetPriority(EINT0_IRQn,0x80);//SW3
NVIC_SetPriority(EINT3_IRQn,0xC0);//Temp sensor
```

6. Application logic enhancements

If you have application logic enhancements, carefully **describe the application logic and the enhancements with significant amounts of detail**, especially technical details. You might consider **including several photos** of your working board at some special steps. This will help to distinguish your system and report from others

OLED

-System monitor is a part of widget engine. These monitoring systems are used to keep track of system information, such as temperature and position of the spacecraft. It is also used to display the current mode and any warning message.

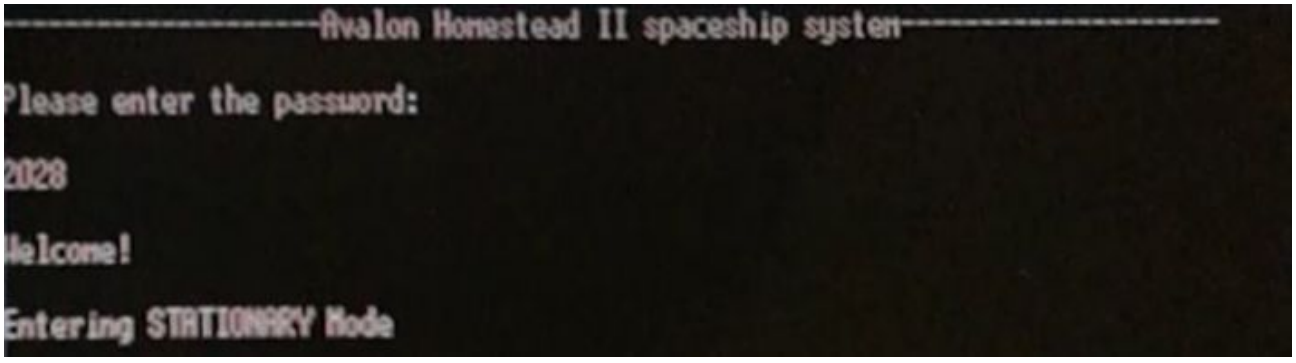
SPEAKER

-Automatic Warning System is part of the signalling system, the concept of AWS is to provide the astronauts with an audible indication of whether the flying path is clear or at dangerous situation and the current status of their spacecraft.

-Refer to page 12 for coding

UART (Zbee)

-Login system is used in the spacecraft we have built. This is a security measure designed to prevent unauthorized access to confidential data. When a login fails (i.e, the username and password combination does not match a user account), the user is disallowed to access.



-Messages and data travel through space as radio waves. The spacecraft has a transmitter and a receiver for sending the data to our spaceship station to keep track all the information such as temperature , position , any warning signals and current mode operation.

7. Improvement

-180 degree Inverted 7 segment
The 7-segment leds are now facing to the user instead of opposite to the user.
For technical details, please refer to page …

-SW3 with EINT0
Previously, we used EINT3 for temperature sensor and SW3. To enhance the responsiveness of our system, we further added in EINT0 function for SW3. This is to prevent the interrupt function from clashing when two inputs happen at the same time.

For technical details, please refer to page …

-Temp interrupt with EINT 3
The given temperature library will make the program slow. Therefore, instead of using temperature library we come out with this Interrupt method to get our temperature even precise and faster.

For technical details, please refer to page …

-UART with UART3
Making use of UART interrupt function, the message can be received immediately after the interrupt is triggered.

For technical details, please refer to page …

-Ideal case
We have designed our system such that it will not respond if the number of presses is more than one within a second in STATIONARY and RETURN modes, and more / less than two in LAUNCH mode.

For technical details, please refer to page …

8. Significant problems encountered and solutions proposed

1. Delay problem in the mainloop:

- Temperature sensor

The temp_read function in the temperature sensor's library consist of a while loop which will slow down the main loop severely and cause the clear warning process to be unresponsive. To overcome this problem, the temperature sensor is configured with EINT3 interrupt, and the number of periods taken to calculate the temperature is reduced to 11 instead of 340. This method significantly improve the smoothness in the main loop.

```
void EINT3_IRQHandler(void)
{

     if (LPC_GPIOINT->IO0IntStatR>>2)
    {
            j++;
            if(j == 1){
                t1 = getTicks();
            }
            else if(j==11) {

              t2=getTicks();

              }

            if(j==20)//give the program some computation time
            {
                j=0;
            }
            LPC_GPIOINT->IO0IntClr = 1<<2;

    }
}
```

- Speaker

There is a while loop and delay function in the Playnote function, which means calling the function in the main loop in the event of warning will delay the system significantly.

```
static void playNote(uint32_t note, uint32_t durationMs) {
    uint32_t t = 0;
    if(note>0) {
        while (t < (durationMs*1000)) {
            NOTE_PIN_HIGH();
            Timer0_us_Wait(note / 2);
            //delay32Us(0, note / 2);

            NOTE_PIN_LOW();
            Timer0_us_Wait(note / 2);
            //delay32Us(0, note / 2);
            t += note;
        }
    }
    else {
        //NOTE_PIN_LOW();
        Timer0_Wait(durationMs);
        //delay32Ms(0, durationMs);
    }
}
```

Solution:

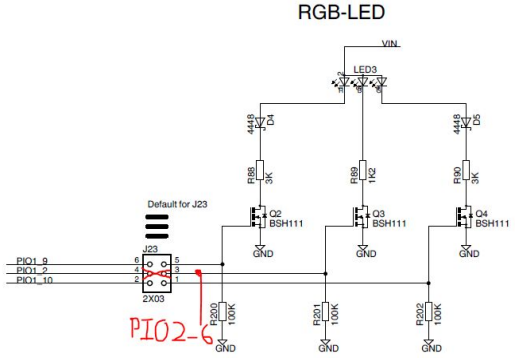
A solution that we figured out is making use systick function to produce a frequency and duty cycle for the speaker .In addition, even though if-else conditional loop and GPIO set and clear functions are used in the systick function, they will only lead to negligible amount of delay in the system. Besides, the

reason for inserting the speaker beeping function in the systick interrupt functions instead of the main function is because we want a faster and more stable speaker response.

Lesson learnt:
We have to be extra cautious with loop in other libraries such as ‘temp.h’, as it might affect the system responsiveness significantly. A possible solution can be make use of interrupt and if-else conditional loop to minimise the delay.

2. Problem involved unwanted internal connection between speaker and blue RGB

Solution:
We realized that the internal connection between audio speaker and blue LED will affect other LEDs in warning status. Therefore, the connection is further modified through shorting an the input of blue LED and the PWM output, PIO2_6 with and physical jumper. PIO2_6 is a good input alternative for blue LED, as it is not connected to any other devices, hence will not affect the system.



Lesson learnt:
It possible to board’s circuit with physical wire so as to eliminate the interference between device.

3. Problem regarding reseting the system

At the very early stage, we were having difficulties in resetting the system. We realised our system can only go from STATIONARY to LAUNCH mode, and follow by RETURN mode. It will not return to STATIONARY mode from RETURN mode after that.

Solution:
We figured out several ways to reset the variables(set to zero) during the transition between different modes.

STATIONARY to LAUNCH

```
if(i == 0){
    oled_clearScreen(OLED_COLOR_BLACK);
    if(log_in == 1){
    msg = "Entering LAUNCH Mode \r\n";
    UART_Send(LPC_UART3, (uint8_t *)msg , strlen(msg), BLOCKING);
    }
    init_t = getTicks();
    buzzer_count = 0;
    countdown = 0;
    mode = 2;
    ini_t_TA = getTicks();
}
INT_COUNT=0;
```

Lesson learnt:
During the design process, we not only need to think of the forward algorithm, but also the backward algorithm to make the system more holistic in any scenarios.

8. Issues or suggestions

Overall this assignment is very challenging yet enjoying. It allows us to explore endlessly in the aspect of c language, libraries as well as the features of LPC1769. We think troubleshooting is the toughest part in the assignment, especially when we are dealing with totally unfamiliar knowledge. But, once we break through the dilemma, our knowledge improve exponentially. We felt that we indeed improved a lot throughout this assignments. However, if there is one thing that we would like to suggest for an improvement, which is the uncertainty of task assignment, as it caused us some inconvenience to keep editing our codes.

9. Conclusion

All in all, we came to know that how important is embedded system, embedded systems are also called real time systems, where the response to external event has hard boundary for execution. Thus, they are better for applications where response to an external event is critical.
Besides, we got to know how to use some keyword to store data such as volatile, static and uint_t, how the device communicate with processor, how interrupt function works and etc.
We learned designing a good program architecture and converting real life situations into an efficient code, and how to write an good looking easily readable and understandable as well as time and memory efficient code.
While making of the project on “LPC1769 board” we made our progress by solving a number of problems. Solution to each problem by us was the most important part of the project because we certainly believe whatever we learnt in this module will prepare us to face our future career.