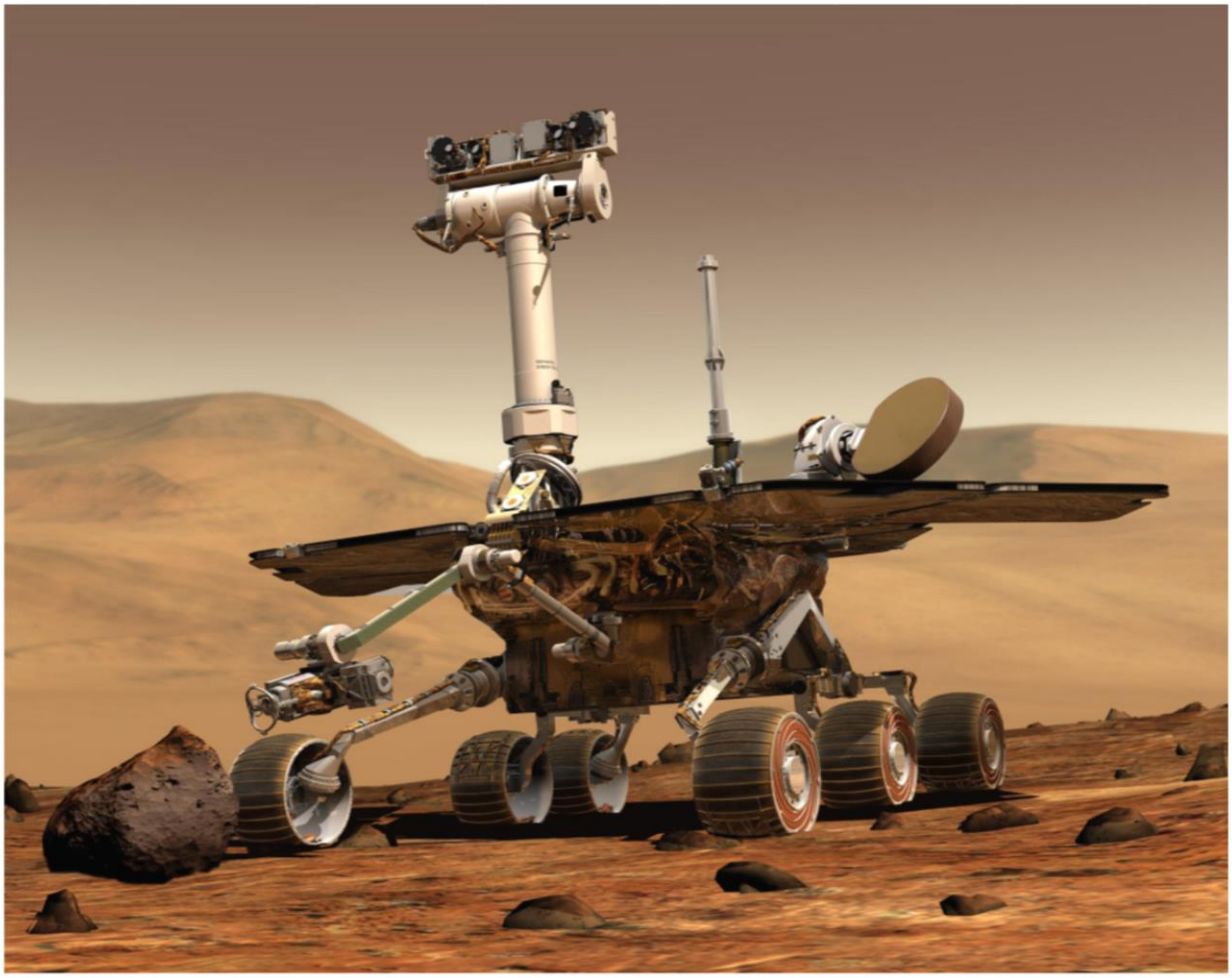

EE2024 Assignment 2: HOPE

HABITABILITY OBSERVER AND PRECURSORY EXPLORER

Naman Sharma | Pan Jieming
A0133877U | A0123918A
Assessing G.A.: Tu Fangwen

Table of Contents

Background / Introduction

In this assessment, we assume that we are dealing with a future robotics mission to Mars, and which will pave the way for safe and sustainable human colonization. The robotic mission shall be called Mars HOPE, short for Habitability Observer and Precursory Explorer. In addition to the scientific instruments present on the robotic rover itself, there are multiple other sensors and systems that deal with monitoring, control and telemetry.

In this assignment, it is assumed that the base board is a set of scientific instruments and sensors on HOPE, the LPC1769 development board is a radiation resistant microcontroller for the scientific instruments and sensors, and the XBee RF module is a set of low gain antennas on HOPE pointed to an intermediate message transmission relay called HOME, short for Hovering Orbiter and Messenger Extension.

Modes of HOPE

On start-up, the HOPE system does a self-diagnostic test before going into the functioning modes. There are two mutually exclusive modes of operation of HOPE:

1. EXPLORER Mode: In this mode, the HOPE system continuously takes in reading from the following three sensors on it and relays it to the HOME:
    a. Accelerometer: This is used as an inclinometer, to prevent HOPE from getting into an unrecoverable position while surveying the potential landing areas for human beings.
    b. Light Sensor: Dust storms can create lightning on Mars. The presence of such dust storms would require HOPE to immediately go into survival mode. The light sensor is used to detect such lightning.
    c. Temperature Sensor: It is used as a part of HOPE's temperature control system. Instruments and power systems on HOPE need to be within the safe temperature range. This is critical since the mean temperature on Mars is below freezing point.
2. SURVIVAL Mode: This mode is to protect the HOPE system from the dust storms that are common on Mars. Under this mode, HOPE stops reading of sensor data and the transmission of data to HOME. It remains in this mode till the time it detects lightning. If it stops detecting lightning, it waits for a certain amount of time before switching back to EXPLORER mode.

The switching from the EXPLORER mode to the SURVIVAL mode depends on a set of conditions. The lightning sensor must sense at least three LIGHTNING_THRESHOLD within 3 seconds. LIGHTNING_THRESHOLD is defined as a light intensity greater than 3000 lux for less than 500 milliseconds.

Detailed System Design

The overall system of HOPE is shown in the flowchart below. This report will consequently look at the respective parts individually.

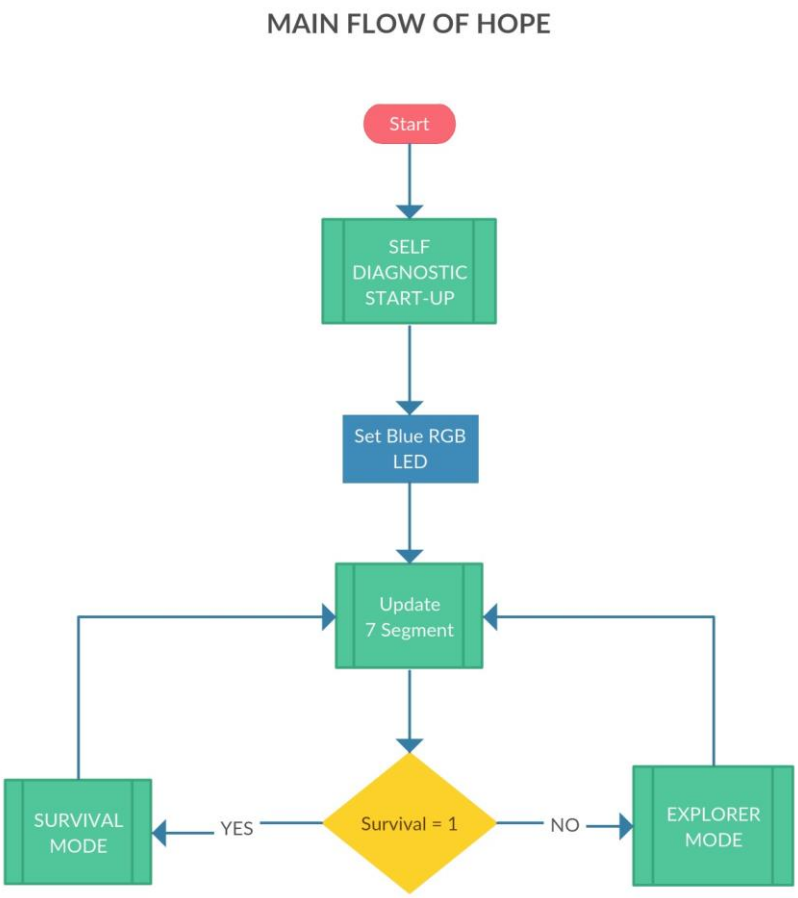

Figure 1: Flowchart of Entire program

I. SELF DIAGNOSTIC START

The baseboard must be on a horizontal surface when powered on for the first time. The self-diagnostic flowchart is presented below. Firstly, we have initiated I2C, SSP, GPIO and UART. We then set priority for NVIC, and setup Systick Timer, EINT3 interrupt, Timero, and Enable interrupts respectively. Following, we initiated all sensors and other related hardware. To offset value for the gravitational force of the planetary body on Mars, we stored the initial value of the z-axis of the accelerometer. The SEGMENT_DISPLAY must start with the number '0' and increment to '6' every second, and continues from letter 'A' to 'F'. The numbers and letters are displayed using led7seg_setChar function in a loop with a systick_delay of 1000 ms. During the 'A' to 'F' 7 segment countdown, an animation will be displayed. After displaying 'F' for 1 second, SEGMENT_DISPLAY turns off and HOPE enters EXPLORER mode.

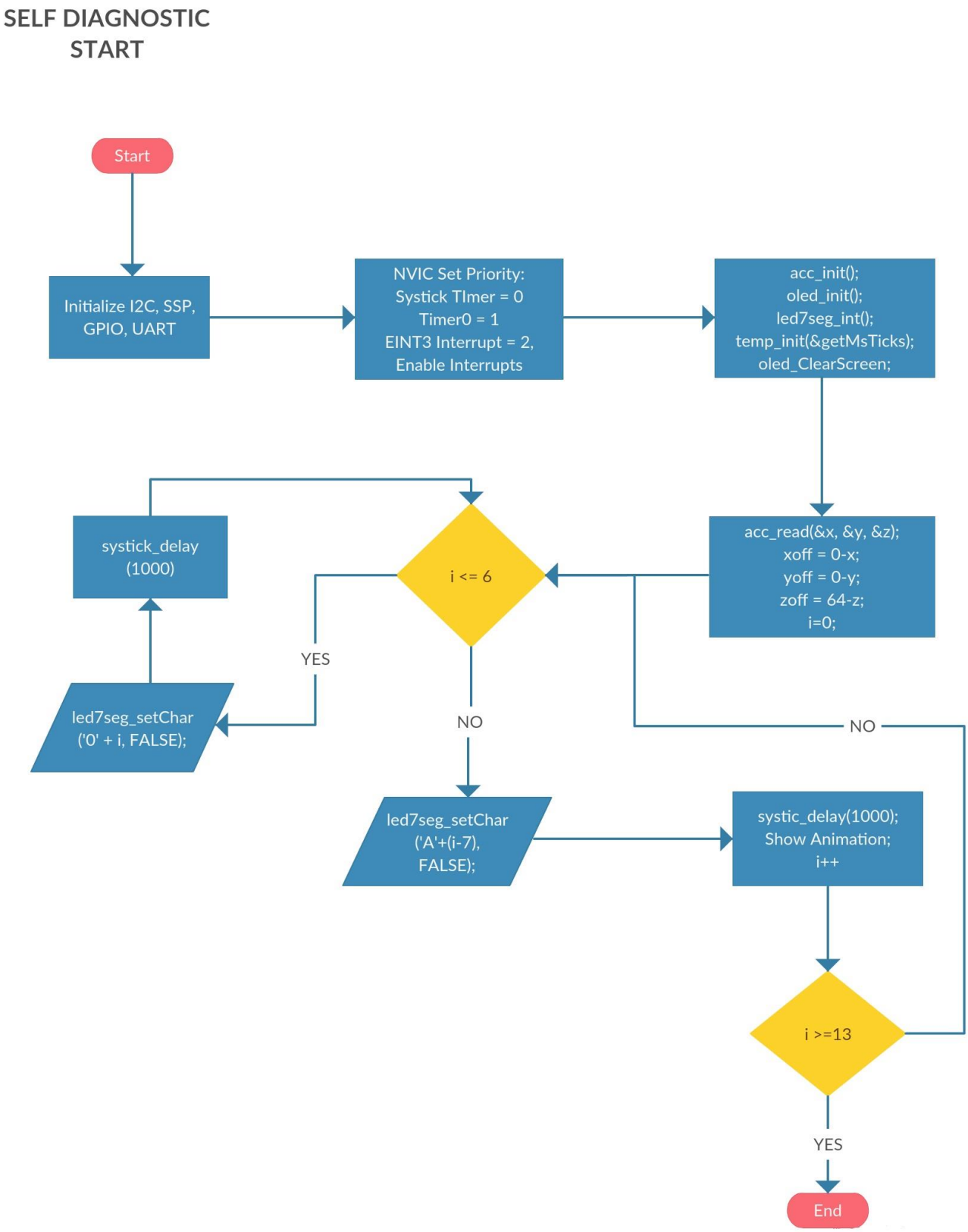

Figure 2: Flowchart of Self Diagnostic

II. EXPLORER MODE

In this mode, HOPE will first check if SW3 interrupt is triggered. If triggered, InstantData will be set to 1. The program will enter a subroutine to read, display and send data on OLED. After which InstantData will be reset to 0. If SW3 interrupt is not triggered, HOPE will enter subroutine to read, display and send data every 2 second. Lastly, if survival = 1, the program will return to survival mode. This is because this cycle of explorer mode function is triggered by SW3 interrupt, hence it should only be executed once. Else the currentTicks will be updated to the current time.

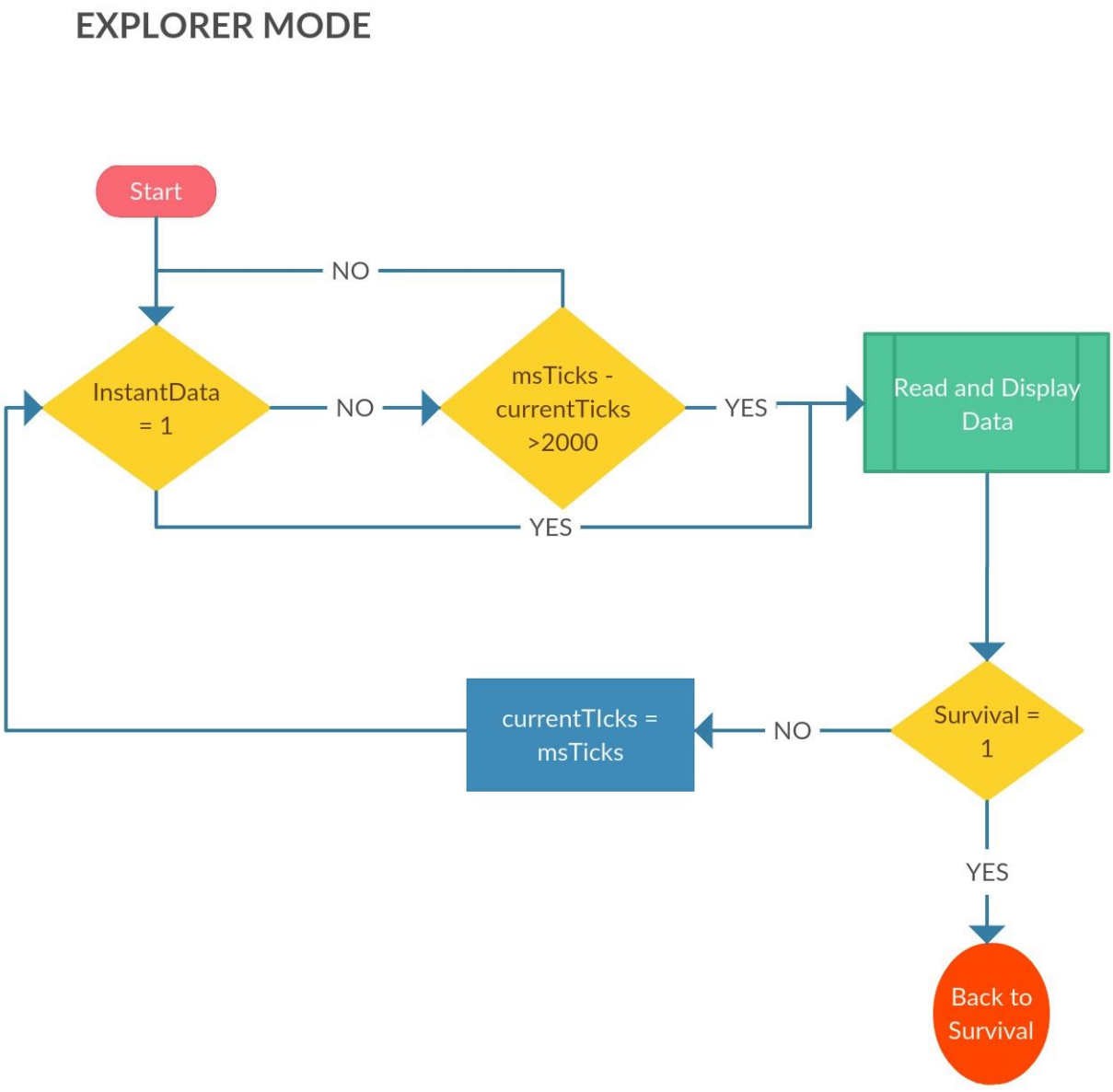

Figure 3: Flowchart for Explorer Mode

The Read and Display Data subroutine reads the value of temperature, light_Val and acc_read and insert them into TempPrint, LightPrint, AccX, AccY, and AccZ respectively using sprintf. The program then prints out all the values on the OLED and send them to HOME using UART Xbee.

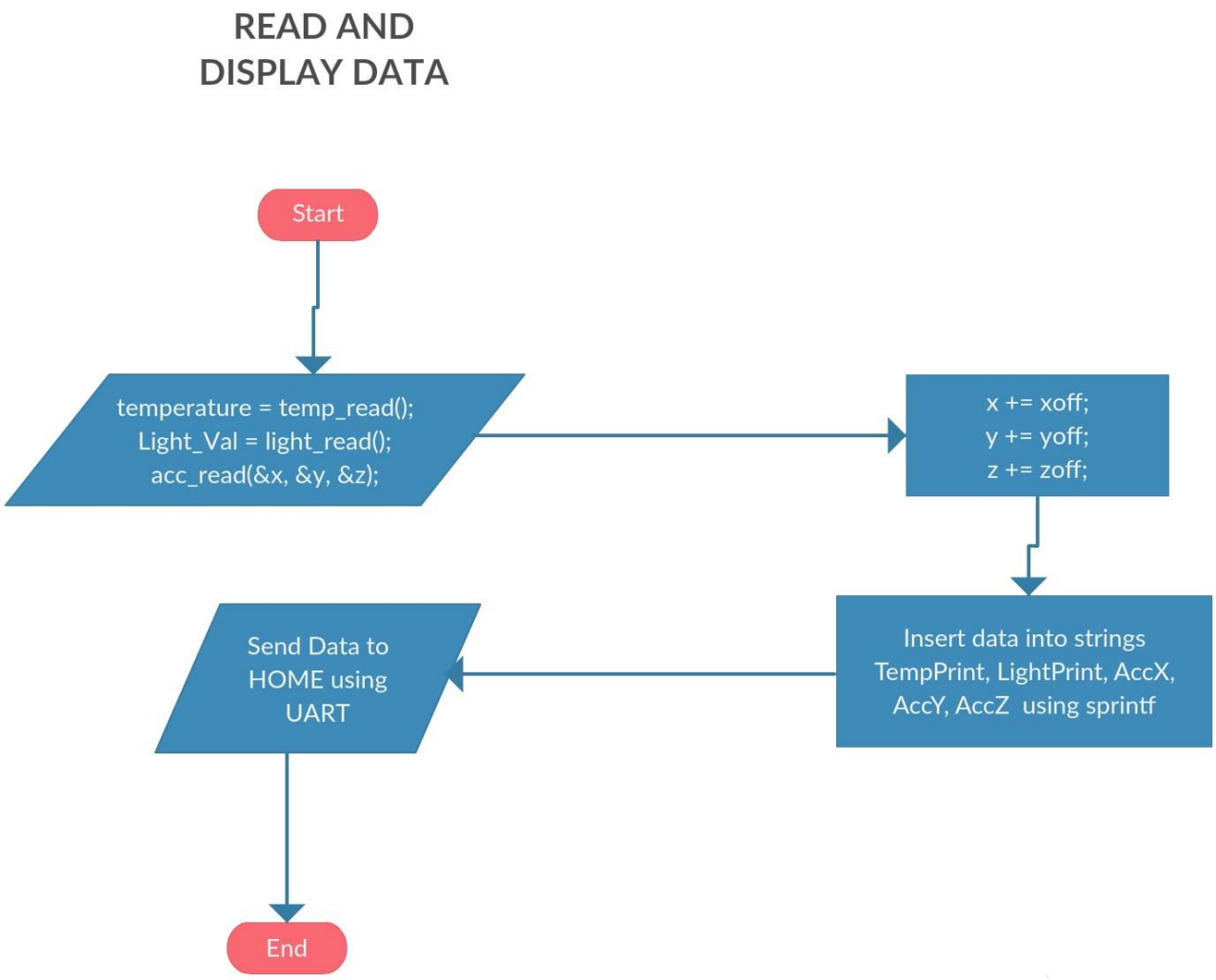

Figure 4: Read and Display Data Sub routine

Update7Seg function is used to update the number of lightning detected within 3 seconds. RecentLightning[i] is an array holding the time values of past lightning instances detected and msTicks is the current time. If the difference between msTicks and RecentLightning is less than 3 seconds, LightNumber will increase. The function will loop and check if the earlier lightning detected is also within 3 seconds, hence increase LightNumber accordingly. The updated LightNumber will then be display on the 7 segment LED display. If the value of RecentLightning[i] is more than msTicks by 3 seconds, the function will exit the loop and check if LightNumber = 0. If LightNumber is 0, 7 Segment LED will display '0'.

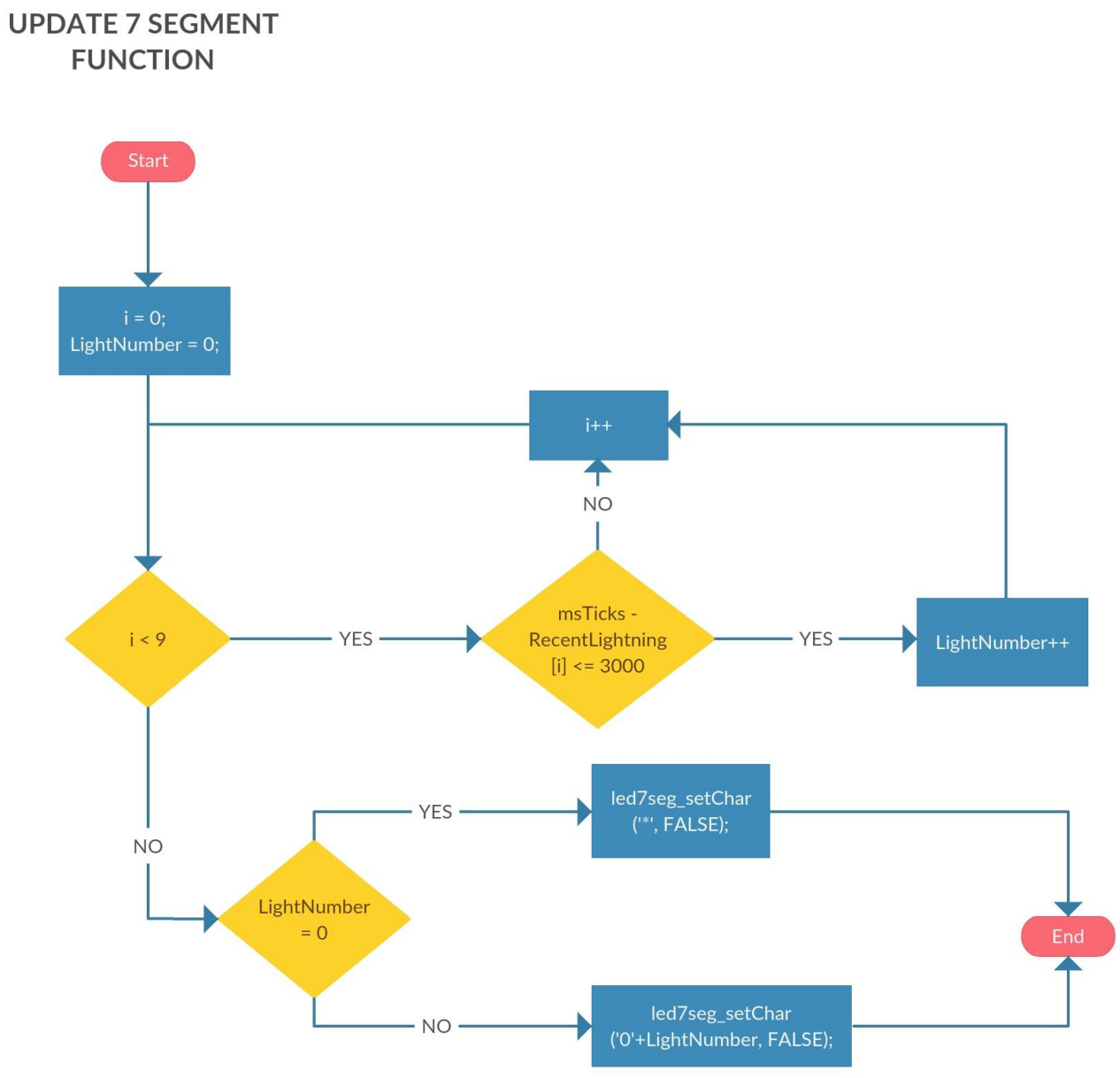

Figure 5: Update 7 segment Flowchart

III. CHANGE FROM EXPLORER TO SURVIVAL

The flowchart showing the EINT3_IRQ Handler Function is shown below. The EINT3_IRQ Handler is the function where the tasks for GPIO Interrupts are written. Therefore, this includes both the SW3 push button function to receive immediate values from HOPE and also the interrupt from the Light Sensor when either the High Threshold or the Low Threshold is passed.

To switch to SURVIVAL Mode, we need to check if there have been at least three LIGHTNING_THRESHOLD's passed in the last 3 seconds. This means that we must be able to measure the period over which the light intensity was over 3000 lux. To achieve this, we take the msTicks value when the light intensity first increases to more than 3000 lux and store it in the variable LightStart. We then store the value of msTicks into LightStop when the intensity reduces below 3000 lux. Therefore, the difference between these two values allows us to judge if the high intensity light lasted for less than 500 milliseconds.

The value of the light_HiThreshold is set at 3000 at start up. This means that whenever the light intensity reaches above 3000 lux, the sensor sends an interrupt request to the processor. Since at this point, the Light_Threshold = 3000, we take the msTicks value and store it into LightStart. We further change Light_Threshold = 62271 and then change the light_HiThreshold = Light_Threshold. This number is the maximum value that the light sensor can sense. Therefore, the light sensor cannot interrupt again due to the light_HiThreshold. On setting the light_LoThreshold = 3000, we ensure that the next interrupt occurs when the light intensity goes below 3000.

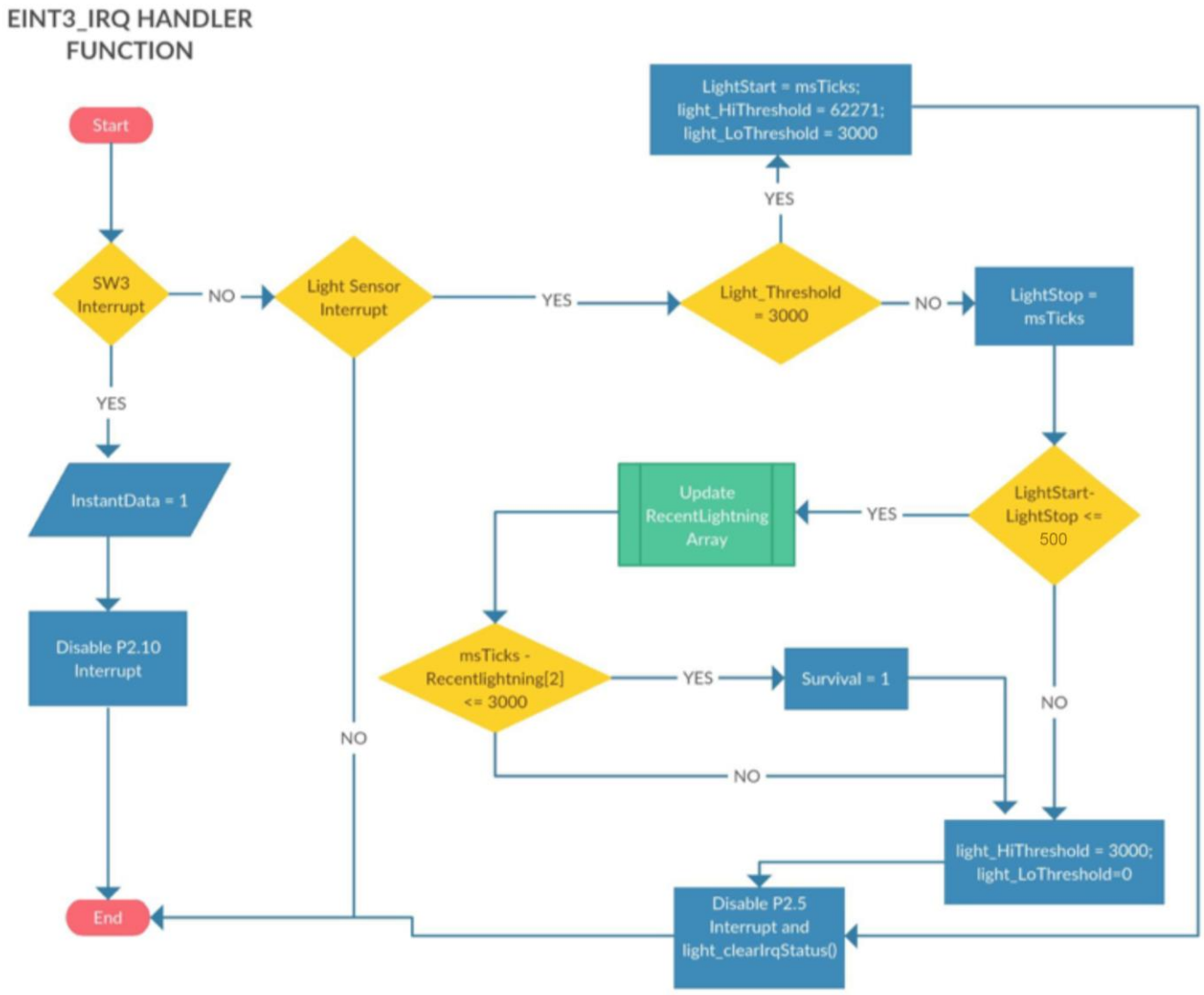

Figure 6: Flowchart for EINT3 Interrupt Handler

When the interrupt due to light_LoThreshold occurs, the msTicks value is saved into LightStop. If LightStop-LightStart >= 500, a lightning is said to have occurred. At this point, we need to update the array RecentLightning[] which holds the msticks values of when the last 9 lightning instances, with the most recent time being stored in RecentLightning[0] and the least recent being in RecentLightning[8]. The array is updated by the Update_Array function, for which the flowchart is given below.

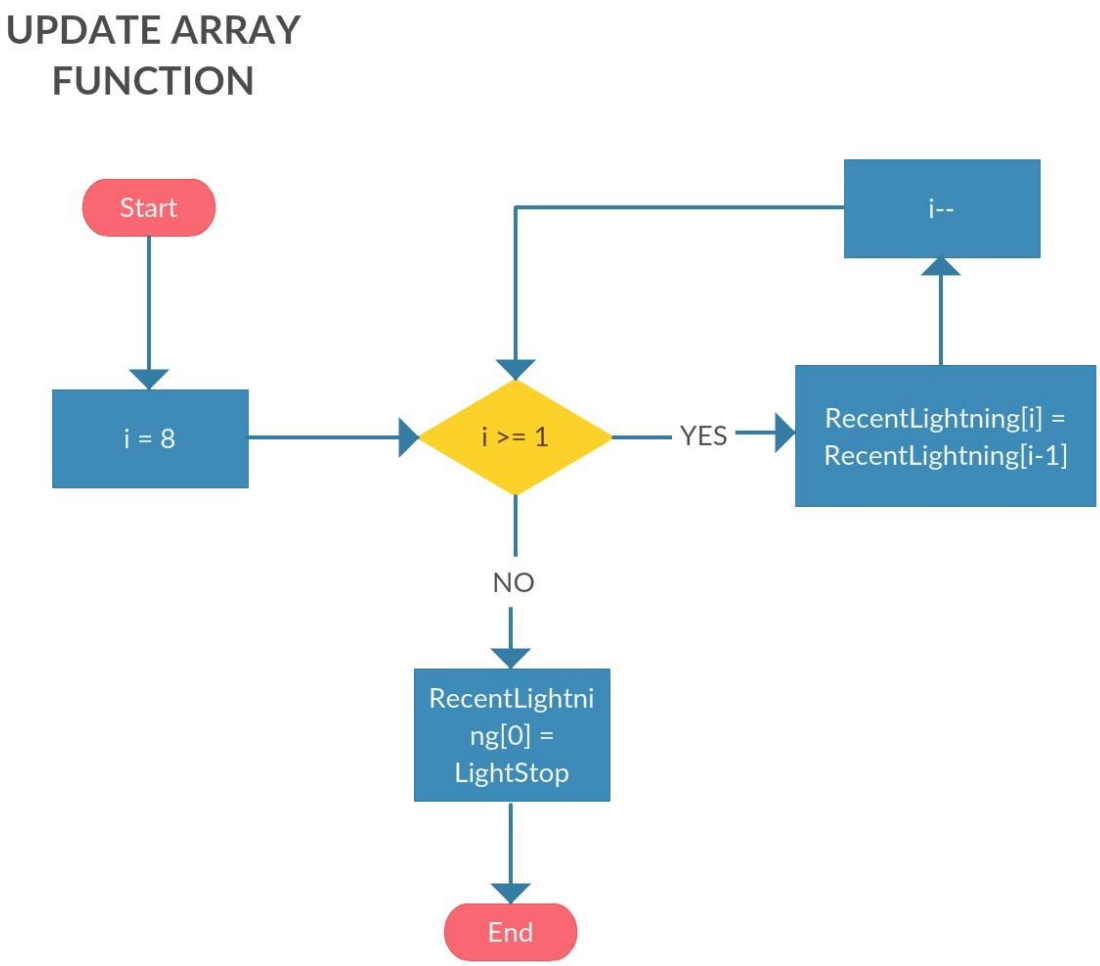

Figure 7: Flowchart for Update Array Function

If msTicks - RecentLighting[2] <= 3000, this means that at least 3 lightning instances have happened in the last 3 seconds. Hence, we set the variable survival = 1, allowing HOPE to enter SURVIVAL mode. Before we exit the interrupt handler, we need to set the light_HiThreshold = 3000 and light_LoThreshold = 0, so that the next interrupt occurs when the light intensity goes above 3000 lux.

IV. SURVIVAL MODE

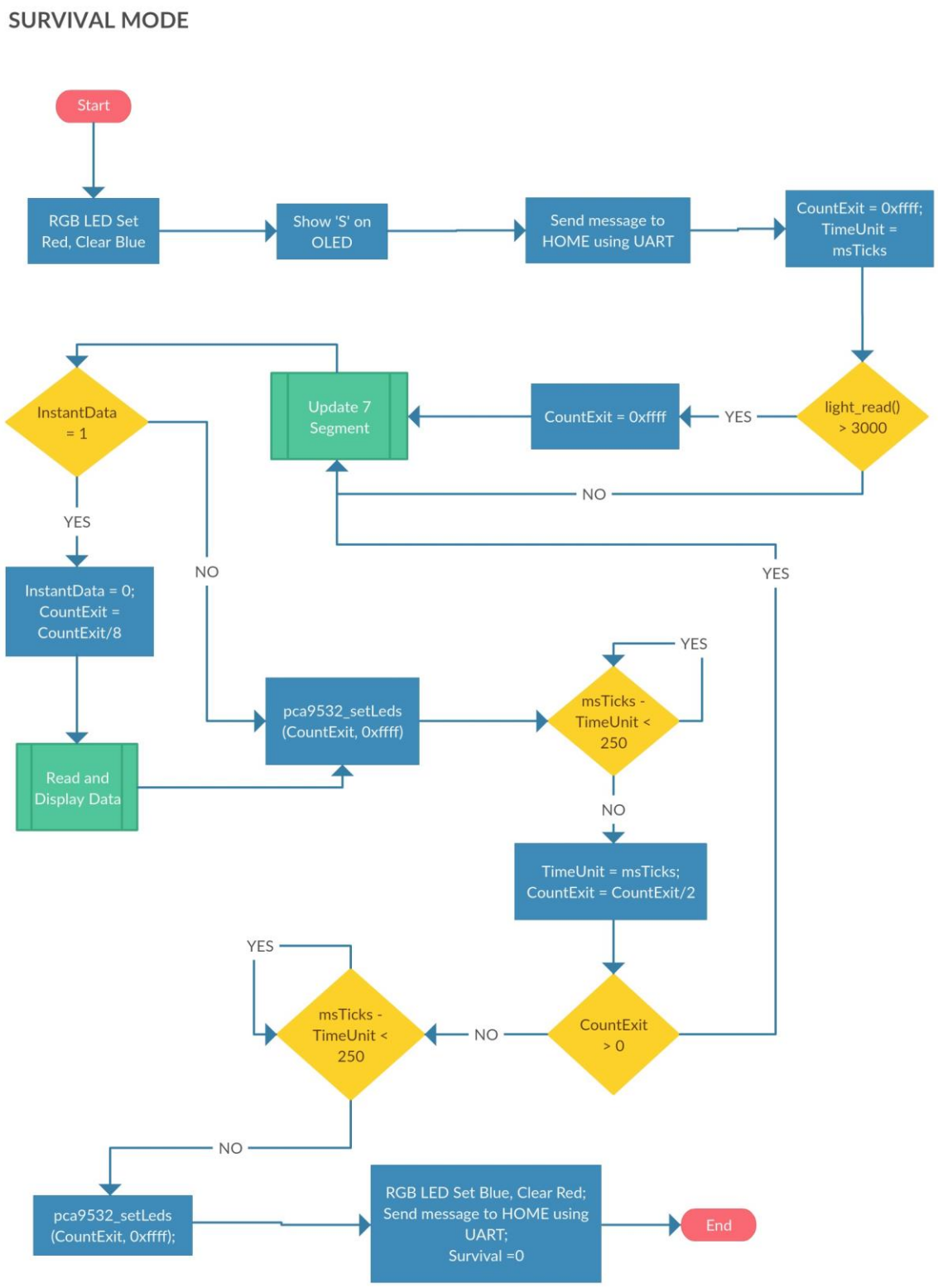

Figure 8: Survival Mode Flowchart

In survival mode, RGB blue LED is cleared and red LED is set. All components value on OLED display will display 'S', while a message will be send to HOME using UART. Next, CountExit will be set to oxffff and TimeUnit will be set to current time. If the light intensity detected by light sensor is more than 3000 lux, CountExit will be set to oxffff and 7 segment LED will continue to display the number of lightning detected in the past 3 seconds. The program will then check if SW3 is triggered and setting InstantData to 1. If so, the program will enter explorer mode for one cycle and read, display and send collected data to HOME. Since the process will take a few hundred milliseconds, hence a 3 16Led will be turned off using CountExit = CountExit/8 and pca9532_setLeds peripheral. If SW3 is not triggered, pca9532 16Led will update directly based on CountExit value. After 250 milliseconds, new current time will be updated to TimeUnit and one more 16Led will be turned off using CountExit = CountExit/2 calculation. This process will continue until CountExit = 0, and after another 250 milliseconds, all 16Led will be turned off and RGB red LED will be Cleared and blue LED is set. A resume message will be send to HOME using UART and the system will return to explorer mode.

Extension

The following features have been added to the HOPE system to improve its functioning and allow more accuracy in its working.

I. NOISE REJECTION FOR LIGHTNING DETECTION

To improve the efficiency of the lightning detection and to make sure that HOPE does not go into SURVIVAL mode even in the absence of a dust storm, we included the feature that allows the HOPE system to better reject random noise during light detection. Now, the HOPE system needs to detect at least three LIGHTNING_THRESHOLDs within a span of 3 seconds for it to go into SURVIVAL Mode. This is achieved by using the RecentLightning Array which stores the time values of the last 9 lightning instances. If msTicks-RecentLightning[2] <= 3000, at least 3 instance of lightning have occurred in the past 3 seconds, and subsequently HOPE switches to SURIVIVAL mode.

II. TIMER0 INTERRUPT FOR MODE INDICATOR

While programming the HOPE System, we realized that if we used a simple delay function to toggle the RGB LED every 1 second (Blue for EXPLORER and Red of SURVIVAL), it leads to unnecessary delay in the system and also irregular timing between the LED toggle states. To counter this, we made use of interrupt handler to maintain a very precise 1 second timing between the toggle of LED state and also remove the unnecessary delay from the function. The interrupt used is the Timero interrupt.

The Timero interrupt is set-up in the Self-Diagnostic Start-up using the following code:

```
//Timer0 initialization
    LPC_SC->PCONP |= 1 << 1; //Power on Timer0
    LPC_SC->PCLKSEL0 |= 1 << 2; //Timer0 frequency = CCLK
    LPC_TIM0->MR0 = 100000000; //MR0 is achieved in 1 second
    LPC_TIM0->MCR |= 1 << 0; //Interrupt when TIM0=MR0
    LPC_TIM0->MCR |= 1 << 1; //Reset Timer0 when TIM0=MR0
    NVIC_EnableIRQ(TIMER0_IRQn);
    LPC_TIM0->TCR |= 1 << 0; //Start Timer0
```

The timer initialization requires the Timero to be powered on first, since by default it is not given power by the LCP1769. After that, we set the frequency of the Timero to be equal to the frequency of the LCP1769. The Timero interrupt occurs when the value in Timero reaches the value stored in one of the MR registers. Here, we use the MR0 register to hold the value of 1 second. Once the interrupt occurs, the Timero is automatically set back to zero, and the process starts again. The Timero Interrupt is handled by the following Handler code:

```
void TIMER0_IRQHandler (void)
{
       //Determine if TIMER0 Interrupt has occurred
       if((LPC_TIM0->IR & 0x01) == 0x01)
       {
              LPC_TIM0->IR |= 1<<0; //Disable Interrupt
              //Toggle the RGB LED
              if(survival)
              {
                     int ledstate = GPIO_ReadValue(2);
                     GPIO_ClearValue(2,(ledstate & (1<<0))); //Turn off if on
                     GPIO_SetValue(2, ((~ledstate) & (1<<0)));
              }
              else
              {
                     int ledstate = GPIO_ReadValue(0);
                     GPIO_ClearValue(0,(ledstate & (1<<26))); //Turn off if on
                     GPIO_SetValue(0, ((~ledstate) & (1<<26)));
              }
       }
}
```

This Handler simply read the current value of the corresponding port and inverts it accordingly. Hence, the RGB LED toggles at a very precise time period of 1 second. The flowchart of the above code is given as follows.

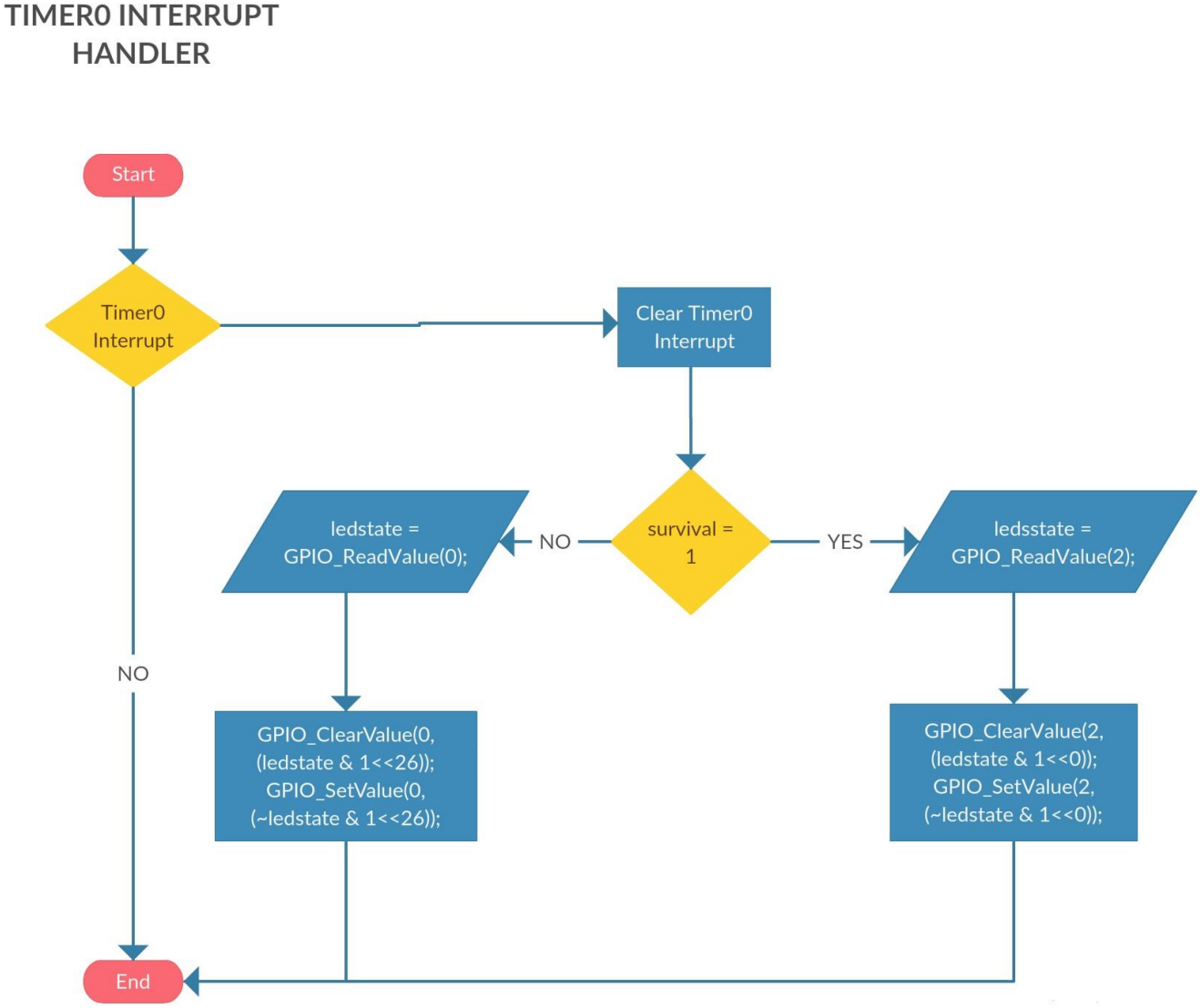

Figure 9: Timero Interrupt Flowchart

III. INVERTED 7 SEGMENT LED DISPLAY

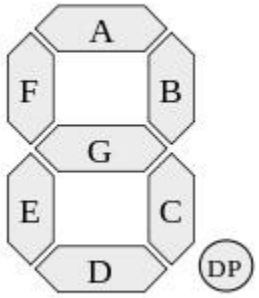

Figure 10: 7 Segment Display

This is done through changing the hexadecimal value in 7 segment LED display library. 7 segment LED display is active low, hence we put 'o' in area we need light up. The hexadecimal value is related to the display in FB[DP]C AGED sequence.

IV. ANIMATION SEQUENCE

Figure 11: Image sequence showing animation

Instead of simply keeping a static file on the start-up, we put an animation sequence into the code, to improve the look of the HOPE system.

V. SW3 INTERRUPT

On the day of the presentation, our GA has given us an added requirement which is to enable SW3 interrupt even during the Self diagnostic Start. This is slightly troublesome as the 7 segment display and OLED animations are shown during the Self diagnostic Start function, and SW3 interrupt will cause these display to clash. Hence a flash of various sensor readings will be shown before the next count down occurs and overwrite the displays. However, this added feature is functional as various sensor readings are successfully taken, displayed on OLED and sent to HOME using UART.

Problems Encountered and Solutions Proposed

I. ERRONEOUS LIGHTNING DETECTION

One of the major problems that we encountered was the erroneous detection of lightning. In our code, after detecting the time when the lightning goes above 3000 lux, we change the `light_HiThreshold` to a value high enough so that the light aimed at the sensor does not reach that intensity again. We earlier set this value to be 50000 lux. We were confident that a light source would not reach that limit. We completed our project with the same limit. However, during the final stages, when we were testing it for extreme cases, we realized that it is actually possible to hit the 60000 lux barrier if the light is kept very close to the sensor.

When such an erroneous detection occurs, the entire interrupt logic goes wrong and the HOPE system freezes. This is a very undesirable effect. In such a case, the only thing we can do is reset the HOPE system.

To overcome this problem, first tried higher values, but soon realized that this was not a fool proof solution and it was possible to hit very high values of light intensity. Therefore, we went back to the basic and looked at the data sheet of the sensor itself. In the data sheet, the range of the values that the sensor can take is clearly mentioned. For the range that we chose, 64000, the maximum sensor reading possible is 62,271. Hence, we set the `light_HiThreshold` to 62,271 when the first interrupt is requested. As a result, the problem was completely solved since the sensor will never send a reading of greater than 62,271.

II. SW3 PUSH BUTTON

In the earlier version of our code, The SW3 Interrupt Handler contained the entire code that would take the readings from the various sensor, store them in variable, put the data into a string using the sprint function, display the data on the OLED screen and send the data to HOME using UART. This is a lot of instructions inside an Interrupt Handler. We recognized this drawback in the system. Also, if you were to press the SW3 while the LPC1769 was running the middle portion of the `main()` function, it would work satisfactorily. However, if you were to hit the SW3 button at the end of the main function, it would jam the system.

Through the debugging process, we pin pointed this problem as being due to the function which writes to the OLED screen in the SW3 interrupt. Since the OLED uses SPI, the SPI data bus would jam because of interruption in the middle of transmission of data.

Recognizing this error, we entirely changed the SW3 Interrupt Handler. Now, the interrupt handler only consists of the following code:

```
// Determine whether GPIO Interrupt P2.10 has occurred (SW3)
	if ((LPC_GPIOINT->IO2IntStatF>>10)& 0x1)
	{
		LPC_GPIOINT->IO2IntClr = 1<<10; //Disable Interrupt
		InstantData = 1;
	}
```

Here, we define a global variable InstantData which is set to be 1 when instant data is requested using the SW3 push button. We subsequently made changes to the main() function of our program, such that if the InstantData = 0, then the program only takes values and prints data if msTicks – currentTicks >=2000, which is the normal mode of operation. However, if InstantData=1, then the processor skips the if msTicks – currentTicks >=2000, allowing immediate reading and display of values. To make sure that the InstantData also works in SURVIVAL mode, we check InstantData every instance one LED turns off. If InstantData=1, we link to the portion of the code in EXPLORER mode that involves the reading and display of values. We check the survival variable to make sure that we link back to survival mode if the SW3 was pushed while HOPE was in SURVIVAL mode.

This improvement completely removed the problem of jamming of the HOPE system, as we only change a variable in the interrupt handler.

Improvements Possible

I. UART INTERRUPT

UART interrupt for remote control of HOPE at HOME. This is an important feature as HOPE is engaged in an unmanned mission on MARS, it will be useful if engineers at HOME can have some level of control of HOPE as a contingency plan.

II. IMPROVED ANIMATION

Complex animation display on OLED can be useful, and it can be achieved through the use of bitmap draw function. We have hence designed more complex bitmap images as animation.

Figure 12: The animation we tried to implement

Conclusion

The system designed by us was able to work as a working model of the HOPE system. The sensors worked well in harmony to give the required effect. We completed all the defined requirements and added extra features to complement those requirements.

The development of the HOPE system allowed us to understand the complexity involved in even small projects of embedded programming. It also lead to a clearer understanding of how a computer is able to perform tasks seemingly in a parallel manner inspite of being inherently linear in its nature.

One of the most important lessons learnt through this assignment is the importance of knowing the hardware of the embedded system. Even though you can write a perfectly working code, it may not work because of simple things such as wrong jumper settings etc. In an embedded system, Software and Hardware go hand in hand to make the system work, and it is necessary to have a good understanding of both at the same time.