EE2024 ASSIGNMENT-2 REPORT

Care Unit for The Elderly (CUTE)

Students : PANKAJ BHOOTRA A0144919W

ZHAO HONGWEI A0131838A

Lab Group : WEDNESDAY

We understand what plagiarism is and have ensured we did not plagiarise for this assignment. This assignment is in partial fulfilment of the requirements for the module EE2024 Programming for Computer Interfaces.

Table of Contents

1. Introduction and Objectives

In this assignment, we have implemented a Care Unit for The Elderly system (CUTE). The main purpose of CUTE is to ensure that the elderly live in a safe and secure environment. The system achieves its purpose by monitoring the luminosity and temperature to send an alert warning signal to the Centralized Elderly Monitoring System (CEMS) if a fire has occurred or the elderly attempts to move around in darkness.

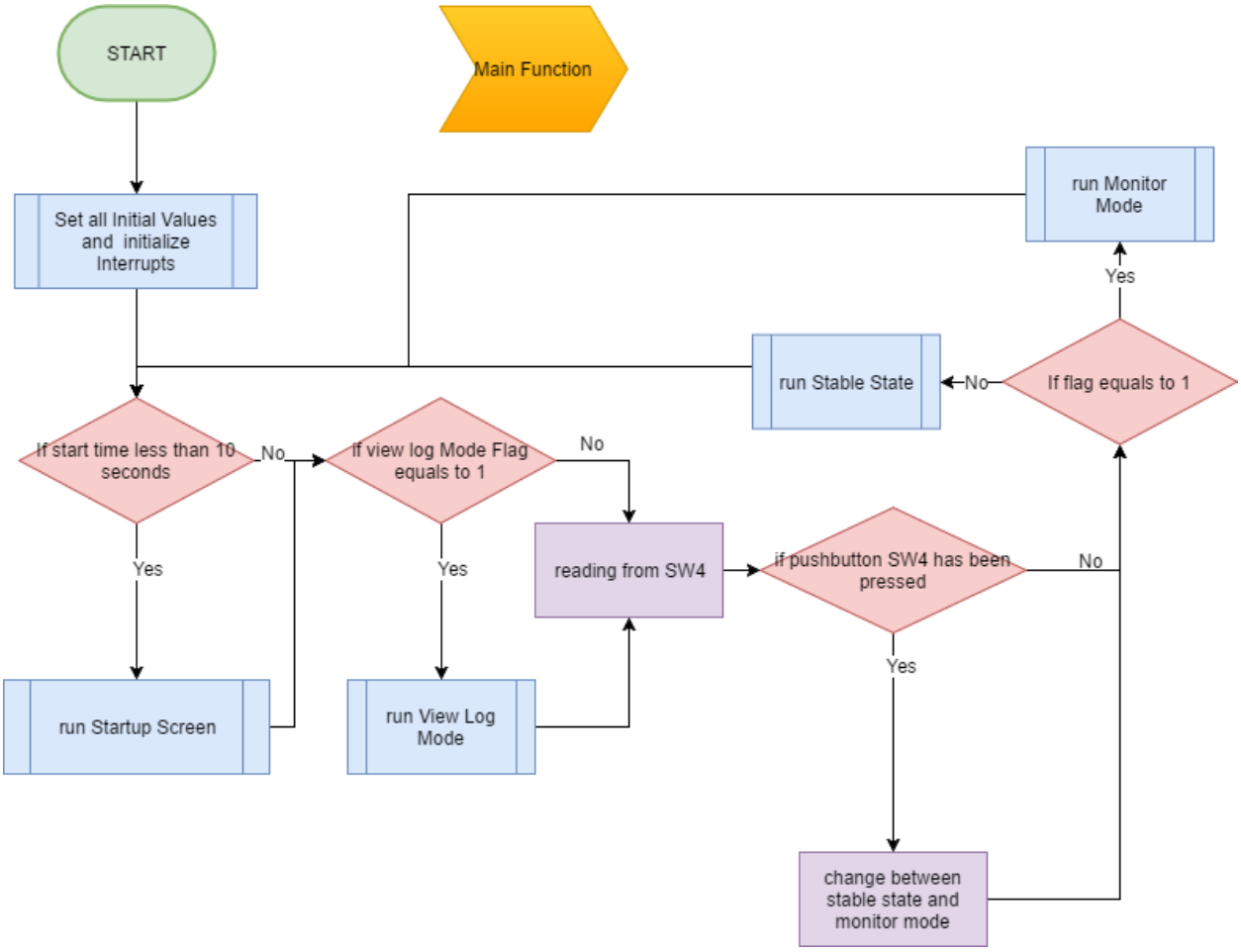
There are two main modes in which this system operates, namely **Stable State** and **Monitor Mode**:

- Stable State is the first mode entered when CUTE is started. In this mode, all devices are OFF and temperature, luminosity and acceleration values are not read. There is no message sampled to the UART either. We use the set of 8 Red LEDs to indicate we are in Stable State by turning them ON. When pushbutton SW4 is pressed, the system goes into Monitor Mode.
- Monitor Mode for CUTE represents the situation when the device is being deployed in a real environment for monitoring the elderly person's surroundings. When in MONITOR mode, we turn on the 8 Green LEDs to indicate the same. As soon as CUTE enters MONITOR mode, the temperature sensor, light sensor and accelerometer values are sampled. The moment danger is detected in the form of fire or movement in darkness, a warning message is sent to CEMS and red/blue LEDs blink to indicate fire or movement in darkness.

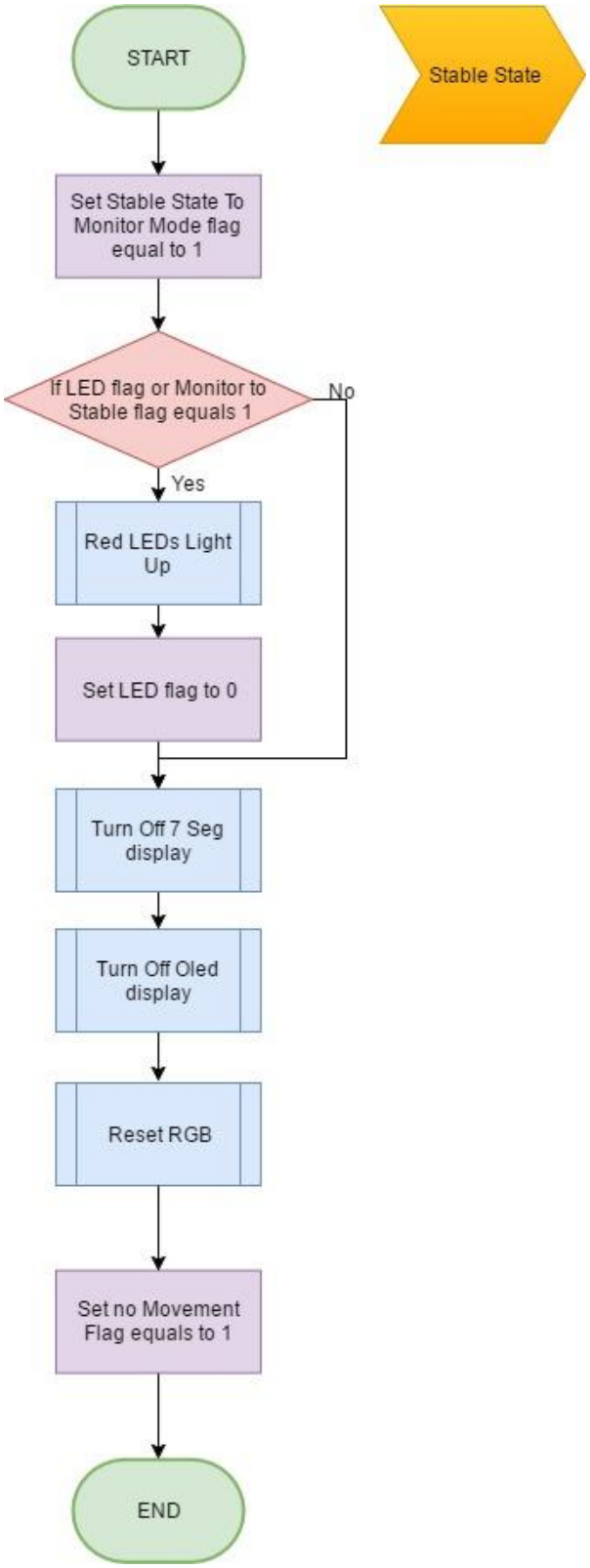
Lastly, we have extended the functionality of the system to include a View Log Mode. This mode is used to browse through previous values of temperature, luminosity and accelerometer values that were displayed on the OLED. We also have other additional features such as our own mechanism for measuring temperature using interrupts instead of the given polling/blocking mechanism in the library code, displaying values on the 7-segment display in the direction the baseboard is tilted (i.e. invert the value displayed if needed) and we have also used interrupts for pushbutton SW3, light sensor and temperature sensor, to make our system efficient in response.

2. Flowcharts describing the system design and approaches

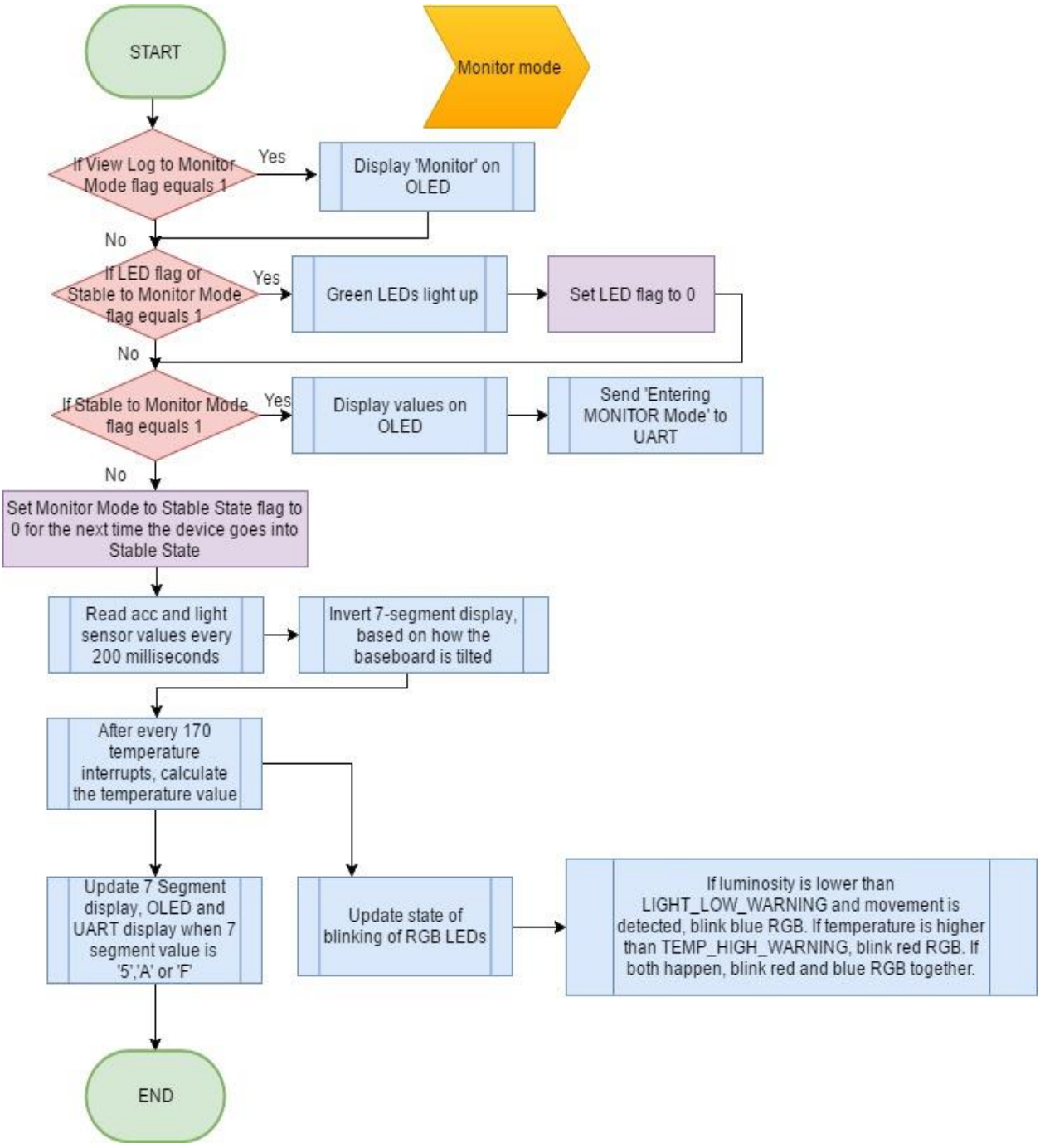
2.1 Main Function



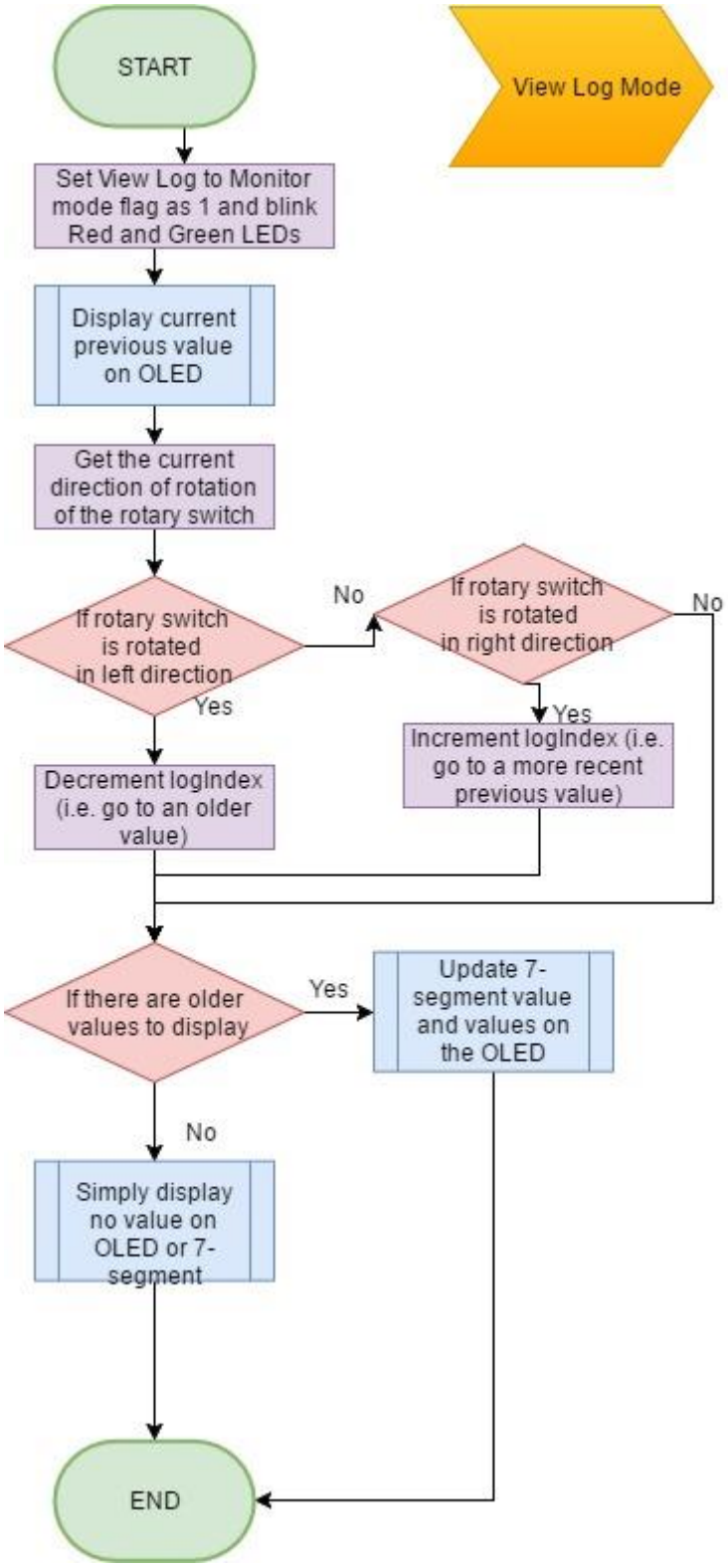
2.2 Stable State



2.3 Monitor Mode



2.4 View Log Mode

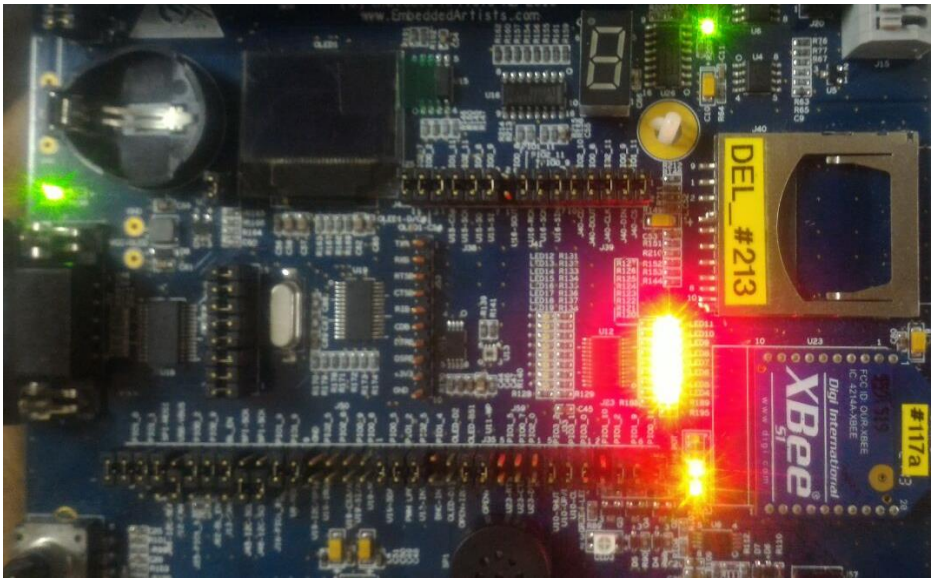


3. Detailed implementation

The CUTE operates in three modes: Stable, Monitor and View Log.

3.1 Stable State

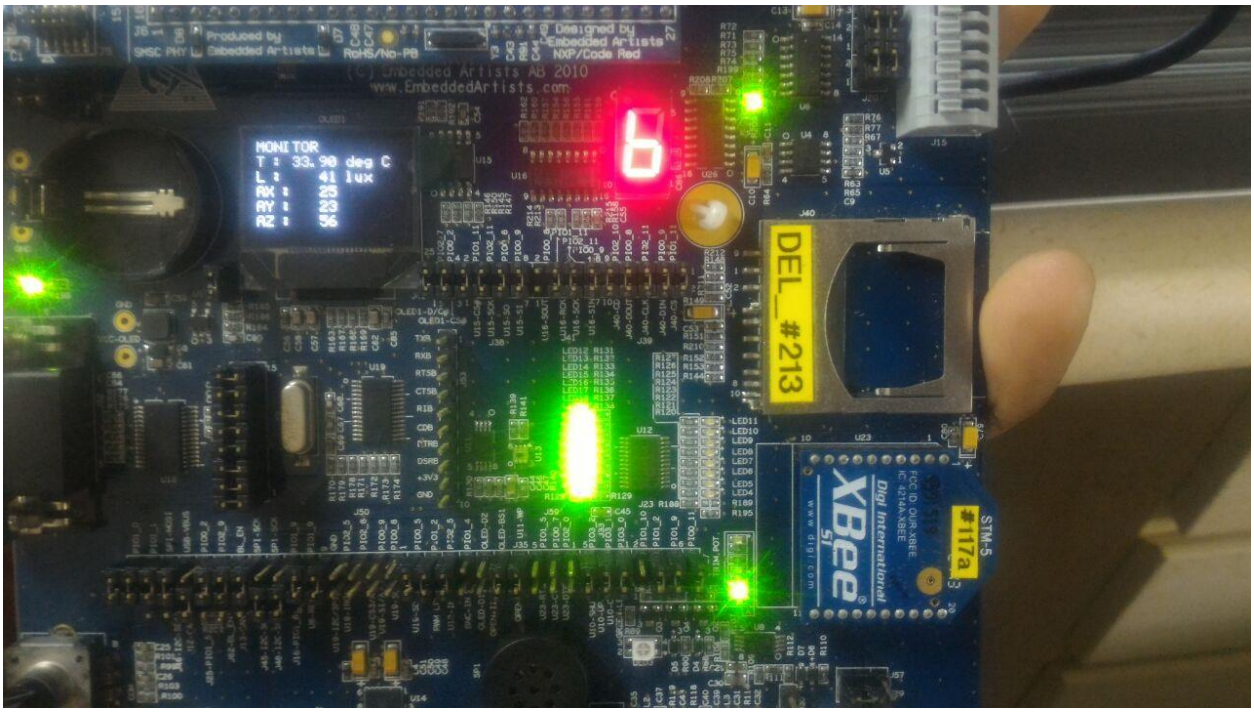
Stable State is the first mode entered when CUTE is started. In this state, all devices are OFF, no values are read and no message is transmitted to the UART. Only the Red LEDs light up, indicating Stable State. When SW4 is pressed, the device switches to Monitor Mode.




```
void runStableState() {
    //the 3 sensors should not be reading values here
    //UART should not be getting any message
    sToMFlag = 1;
    if (ledFlag == 1 || mToSFlag == 1) {
        pca9532_setLeds(0x00FF, 0xFFFF);
        ledFlag = 0;
    }
    oled_clearScreen(OLED_COLOR_BLACK);
    led7seg_setChar('@', 0); // turn off the display
    blink_blue_flag = 0;
    blink_red_flag = 0;
    rgbFlag = 0;
    setRGB(RGB_RESET);
    noMovementFlag = 1;
    tTicks = msTicks;
}
```

3.2 Monitor Mode

Monitor Mode is when the temperature sensor, light sensor and accelerometer values are sampled and the message “Entering MONITOR Mode” is sent to the UART. The temperature is measured at every 170 interrupts by the temperature sensor whereas the light sensor and accelerometer values are read every 200 milliseconds. We are also displaying the hex values 0-F on the 7-segment at intervals of 1 second and at display value equal to 5 or A or F, we update the temperature, light sensor and accelerometer values displayed on the OLED. At F, we also send the same data to the CEMS aka UART Terminal. In order to send data to the UART Terminal, we use Wireless Xbee modules that have been configured to process data at the baud rate of 115200 bytes/second. We use the RGB LED and the UART to indicate danger in the form of fire or movement in darkness. Hence, when the temperature crosses a threshold set at 45℃, the Red LED of RGB blinks and if the luminosity goes below a threshold set at 50 lux and movement is detected, then the Blue LED of RGB blinks, both alternating between ON and OFF every 333 milliseconds. They remain that way until the system switches back to Stable State. The relevant messages indicating danger are also sent to the UART.

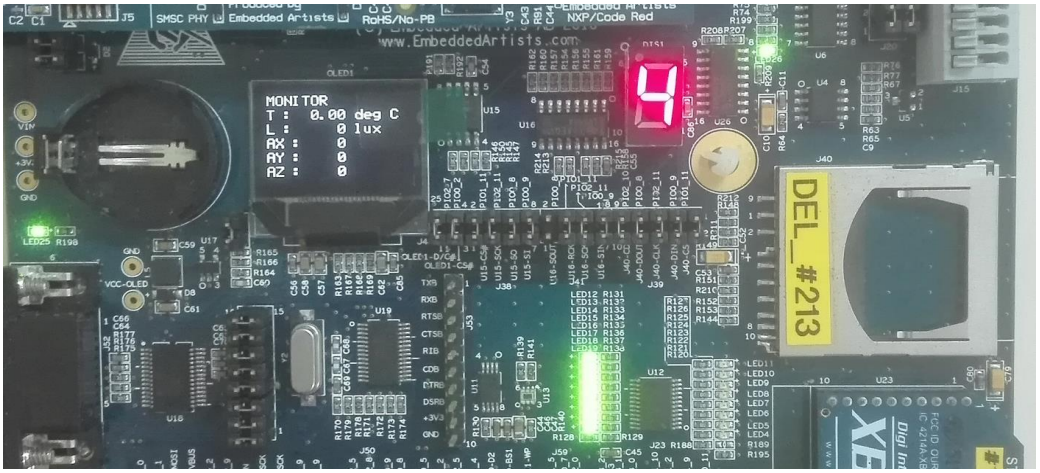


Monitor Mode

3.3 Transition from Stable State to Monitor Mode

```
if (sToMFlag == 1) {
    oled_clearScreen(OLED_COLOR_BLACK);
    oled_putString(0, 0, (uint8_t *) "MONITOR", OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    unsigned char TempPrint1[40] = "";
    unsigned char LightPrint1[40] = "";
    strcat(TempPrint1, "T :  0.00 deg C");
    oled_putString(0, 10, (uint8_t *) TempPrint1, OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    strcat(LightPrint1, "L :     0 lux");
    oled_putString(0, 20, (uint8_t *) LightPrint1, OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    char XPrint1[20] = "", YPrint1[20] = "", ZPrint1[20] = "";

    strcat(XPrint1, "AX :    0 ");
    strcat(YPrint1, "AY :    0 ");
    strcat(ZPrint1, "AZ :    0 ");
    oled_putString(0, 30, (uint8_t *) XPrint1, OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    oled_putString(0, 40, (uint8_t *) YPrint1, OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    oled_putString(0, 50, (uint8_t *) ZPrint1, OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    //displaying entering monitor mode on UART
```

We set a flag indicating change of state from Stable to Monitor Mode. During this change, we clear the OLED screen and display necessary information that is to be displayed during Monitor Mode. Also, we send the message “Entering MONITOR Mode” to the UART terminal.



Transition from Stable State to Monitor Mode

3.4 Log Mode

This feature is explained in details under Section 4.7 of this report as part of “Application Logic Enhancements”.

3.5 Transition from Log Mode to Monitor Mode

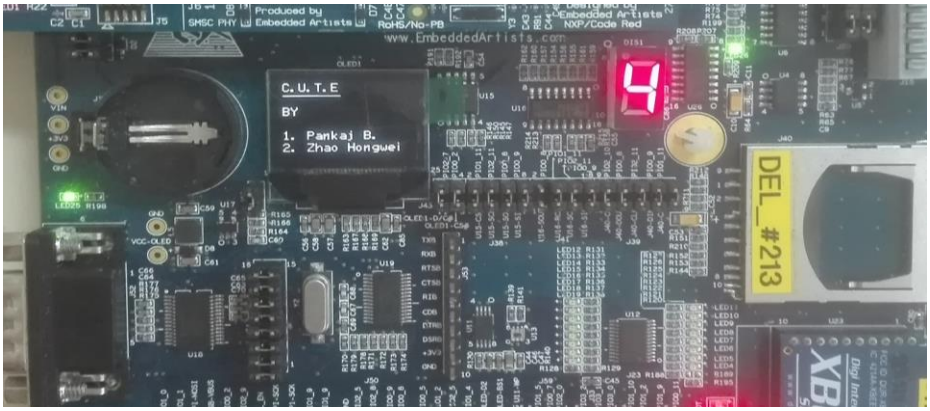
```
if (lToMFlag == 1) {
    lToMFlag = 0;
    logIndex = index;
    oled_putString(0, 0, (uint8_t *) "MONITOR       ",
            OLED_COLOR_WHITE, OLED_COLOR_BLACK);
}
```

We set a flag indicating change of state from View Log to Monitor Mode. During this change, we change the heading displayed on the OLED from “VIEW LOG” to “MONITOR” and also set the pointer to all values (logIndex) back to the current value (index). This is done so that when we switch to Log Mode again, then the values are once again displayed starting from the most recent value and going back to the older ones. Once the system is back in Monitor Mode, it resumes from where it left off while switching from Monitor to View Log Mode.

4. Application Logic Enhancements

4.1: We display our names on the OLED for 10 seconds with a 10-second countdown on the 7-segment display (from 9 to 0) when the system is first powered ON. *Devices involved*: OLED, 7-segment display.

```
void runStartupScreen() {
    oled_putString(0, 0, (uint8_t*) "C.U.T.E", OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    oled_line(0, 10, 42, 10, OLED_COLOR_WHITE);
    oled_putString(0, 20, (uint8_t*) "BY", OLED_COLOR_WHITE, OLED_COLOR_BLACK);
    oled_putString(0, 40, (uint8_t*) "1. Pankaj B.", OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    oled_putString(0, 50, (uint8_t*) "2. Zhao Hongwei", OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
    if (firstTime) {
        led7seg_setChar(val, 1);
        firstTime = 0;
    }
    if (msTicks - sevenSegIntroTime >= 1000) {
        led7seg_setChar(--val, 1);
        sevenSegIntroTime = msTicks;
    }
}
```



Startup screen on OLED (above)

4.2: When displaying values on the 7-segment, we display it according to how the baseboard is tilted. We invert the value displayed in the direction that favors the reader (similar to screen rotation property of a smartphone). *Devices involved*: 7-segment display.

To meet this enhancement, we edited the library 7-segment code that was provided to us to include the following array. Essentially, we added the same characters 0-F, but now in inverted position.

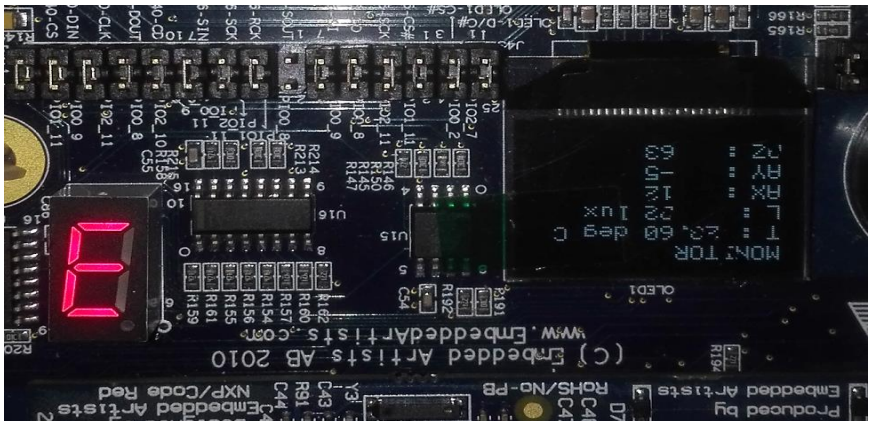
```
/* inverted character mapping */
static uint8_t inverted_chars[] = {
        /* '-', '.' */
        0xFB, 0xDF, 0xFF,
        /* digits 0 - 9 */
        0x24, 0x7D, 0xE0, 0x70, 0x39, 0x32, 0x22, 0x7C, 0x20, 0x30,
        /* ':' to '@' are invalid */
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        /* digits A - F */
        0x28, 0x23, 0xA6, 0x61, 0xA2, 0xAA,
};
```

Set of inverted characters (new hex codes) added (above)

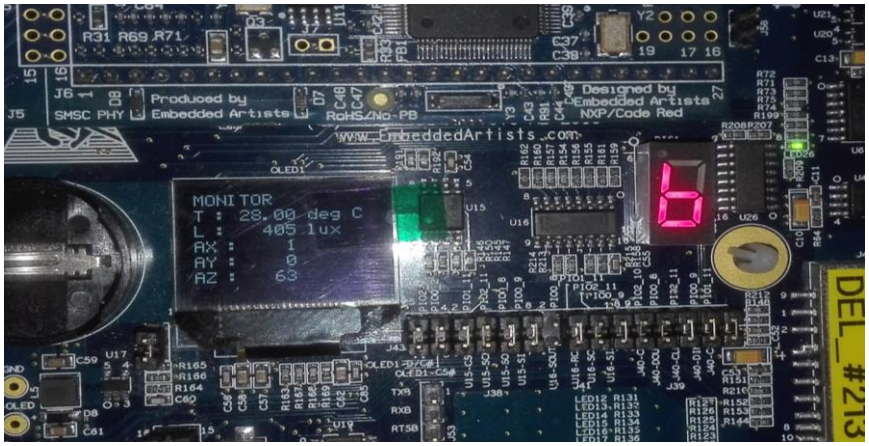
We also removed the concept of ‘rawMode’ provided in the library code (since our project requirements don’t involve using characters outside 0-9 and A-F) and replaced it with ‘invertedMode’ to achieve inverted character display, as shown below.

```
void led7seg_setChar(uint8_t ch, uint32_t invertedMode)
{
    uint8_t val = 0xff;
    SSP_DATA_SETUP_Type xferConfig;

   // LED7_CS_ON();

    if (ch >= '-' && ch <= '|') {
        val = chars[ch-'-'];
    }

    if (invertedMode) {
        val = inverted_chars[ch - '-'];
    }
```

Displaying inverted value ‘E’ (below)



Displaying normal value ‘B’ (below)



4.3: The Green and Red LEDs indicate which mode we are currently in. If we are in ‘Stable’ State, the Red LEDs will light up. If we are in ‘Monitor’ Mode, the Green LEDs will light up. If we are in ‘View Log’ Mode, both the Green and the Red LEDs will blink at the same rate as the

RGB does under the basic requirements (i.e., toggles every 333 milliseconds synchronously). *Devices involved*: Green and Red LEDs (pca9532 LEDs).

4.4: The pushbutton SW3 is used to switch to 'View Log' Mode any time from any of the other modes (Stable or Monitor). SW3 is handled using an interrupt, at every falling edge of pushbutton SW3, to make it more efficient in response. So the current mode switches to View Log the exact moment the falling edge is encountered and without any delay. *Devices involved*: Pushbutton SW3.

4.5: We have used interrupts for light sensor. For light sensor, we have set a low threshold equal to the LIGHT_LOW_WARNING value defined in Assignment 2 guide (50 lux). So, the light sensor will trigger an interrupt every time the value goes below LIGHT_LOW_WARNING (this helps make RGB respond much quicker as now the blink blue RGB happens almost instantaneously as the value goes below LIGHT_LOW_WARNING). *Devices involved*: Light sensor.

4.6: We have also used interrupts for temperature sensor. Note that we haven't used *temp_read()* to read in temperature value because *temp_read()* has a while loop which causes polling for about half a second, thus delaying all the other devices on the baseboard by the same time. Instead, we use temperature sensor interrupts to measure the period of one PWM generated by the temperature sensor at every interrupt. This is done by taking the average of period over 170 interrupts. With this value of period, the temperature is then calculated using the formula:

```
(((1000.0 * period over 170 interrupts) / 170) - 2731) / 10.0
```

This approach removes the polling problem and ensures that all devices respond timely as the issue of lag because of temp_read() is now eliminated. *Devices involved*: Temperature sensor.

```
if (tempIntFlag == 1) {
    tempIntFlag = 0;
    tempEndTime = tempEndTime - tempStartTime;
    tempSensorVal = ((1000.0 * tempEndTime) / TEMP_HALF_PERIODS) - 2731;
    tempIntCount = 0;
    tempStartTime = 0;
    tempEndTime = 0;
    if ((msTicks - tTicks >= 400) && (blink_red_flag == 0)
            && (movementDetected() == 1)
            && (tempSensorVal >= TEMP_HIGH_WARNING)) {
        blink_red_flag = 1;
    }
}
```

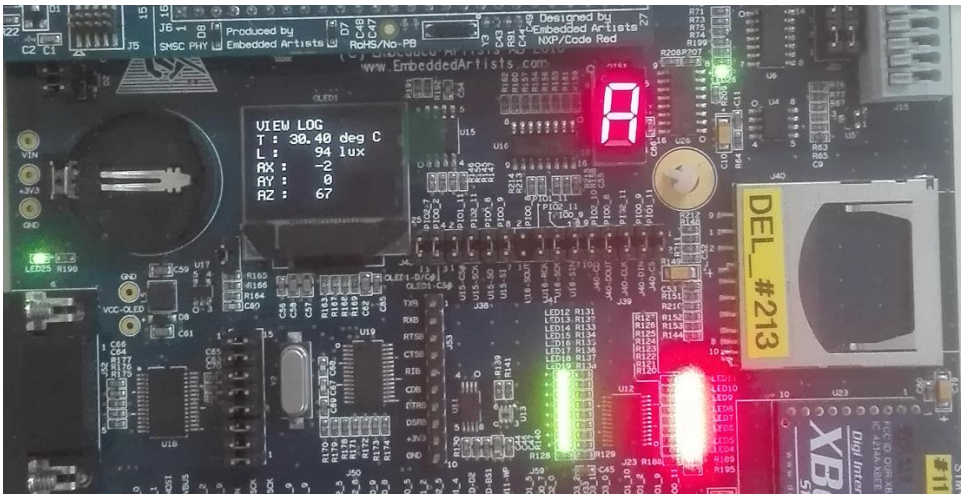
Calculating temperature (above)

```
// EINT3 Interrupt Handler
void EINT3_IRQHandler(void) {
    // Determine whether GPIO Interrupt P2.10 has occurred - Pushbutton SW3
    if ((LPC_GPIOINT ->IO2IntStatF >> 10) & 0x1) {
        // Clear GPIO Interrupt P2.10
        LPC_GPIOINT ->IO2IntClr = 1 << 10;
        logModeFlag = !logModeFlag;
    }

    // Determine whether GPIO Interrupt P2.5 has occurred - Light Sensor
    if ((LPC_GPIOINT ->IO2IntStatF >> 5) & 0x1) {
        LPC_GPIOINT ->IO2IntClr = 1 << 5;
        light_clearIrqStatus();
        lightIntFlag = 1;
    }

    // Determine whether GPIO Interrupt P0.2 has occurred - Temperature Sensor
    if ((LPC_GPIOINT ->IO0IntStatF >> 2) & 0x1) {
        if (tempStartTime == 0 && tempEndTime == 0)
            tempStartTime = msTicks;
        else if (tempStartTime != 0 && tempEndTime == 0) {
            tempIntCount++;
            if (tempIntCount == TEMP_HALF_PERIODS) {
                tempEndTime = msTicks;
                tempIntFlag = 1;
            }
        }
        LPC_GPIOINT ->IO0IntClr = 1 << 2;
    }
}
```

Interrupts for Pushbutton SW3, Light Sensor and Temperature Sensor

4.7: We have a new mode called View Log, as mentioned under sections 3.4 and 4.4. This mode is used to browse through previous values of temperature, luminosity and accelerometer values that were displayed on the OLED, i.e., the values displayed when 7-segment showed 5, A or F. Browsing through previous values is done using the rotary switch (SW5). The user can browse through as many old values as desired, with the limit set at 999. It is assumed that 999 will never be reached. "Browsing through previous values" means that the previous values will be displayed on the OLED when the rotary switch is rotated in the appropriate direction. Also, the 7-segment value will be changed to show the value (5, A or F) at which these values were previously

displayed in MONITOR mode. This is an extremely useful feature for the user as it can help him/her in situations such as when the RGB LED starts blinking (either of blue or red or both) but the current values displayed on the OLED are within the acceptable threshold and the user would like to know what values of temperature or luminosity had caused the RGB to blink. *Devices involved*: Rotary switch, OLED display and 7-segment display.



Log Mode in action

4.8: We have also *enabled* the standard interrupt handler for UART send and receive. However, due to time constraints, we haven't *used* it in the main code. Our target was to trigger an interrupt every time any message is sent to the UART or received from the UART, so as to make it more efficient in response (similar to how we handled interrupt for pushbutton SW3). *Devices involved*: UART peripherals, XBee modules and Teraterm terminal.

5. Significant Problems Encountered and Solutions Proposed

5.1: *Problem with temp_read() function*: We discovered that temp_read() method defined in temp.c of the baseboard library code takes up around 0.5 seconds because of a polling/blocking mechanism it involves to compute temperature. Hence, every time temp_read() was called, all other devices on the baseboard would be blocked for around 0.5 seconds. In order to tackle this issue, we decided to write our own code for measuring temperature that doesn't involve polling, but involves use of interrupts. The idea is that every time temperature value is read by the sensor,

it returns an analog PWM whose period is directly proportional to the temperature value. Hence, in our code, we get this PWM at every falling edge, i.e. at every interrupt, and in order to get an average value for the period, we get the period for 170 PWM's, i.e., period after 170 interrupts were raised. This period is then multiplied by 1000 to get period in microseconds, which then divided by 170 gives the average period that is roughly equal to 10 times the temperature in kelvin. Then, we subtract 2731 from this number and divide this by 10 to get the temperature reading in degree celsius. This way, we improved the response time of our code and eliminated polling from temperature reading process, ensuring that other devices on the baseboard are no longer blocked every time temperature is measured.

5.2: *Lagging*: We noticed that sometimes, all devices on the baseboard would suddenly lag and stop working. On debugging, we found out that there are quite a few different reasons/possibilities which could have led to this problem. We discovered that the accelerometer reading and clearing the light sensor interrupt using *light_clearIrqStatus()* is done via the same I2C bus. As a result, if both of these processes request the I2C bus at the same instant, then it causes a breakdown of the whole system. Hence, we set appropriate flags so that both the processes didn't request the I2C bus at the same time. Another issue was that there were certain variables, mostly flag variables, whose values were being updated periodically (for example, every 1 second) but were being accessed all the time (for say, conditional checking). This is a possible reason for a lag to happen because in this situation, the variable could possibly be accessed at the very instant it is being updated, which would cause a breakdown of the system. To tackle this issue, we made sure all variables were both updated and accessed at fixed times only (say, every 1 second only) so that this problem was eliminated.

Another reason for the lag was regarding an array that we used in our code to store previous values that were displayed on the OLED (as part of one of our enhancements that involves using the rotary switch to display previous values on the OLED). The array had a fixed size of 999, but we were adding values to the array whenever the 7 segment value was 5 or A or F. Hence, for the entire 1000 milliseconds during which the 7 segment value is any one of 5 or A or F, the array will keep getting a new value and so in no time, we would cross the number of values that the array can store (999), causing an index out of bounds error which crashes the baseboard. Hence,

we changed our code so that values were added to the array *only once* when the 7 segment value is 5 or A or F.

We also found that reading accelerometer and light sensor values at every single iteration of the infinite while loop slows down the system. Hence, now we read these values only every 200 milliseconds, which has improved response time.

6. Issues or Suggestions

It was truly an amazing experience working on this project as we got to learn so much about programming embedded systems, using existing libraries for various devices, learning how to read datasheets, etc. We also learnt about concepts like interrupts, which are so useful and are being used everywhere around us. We would like to emphasize that this project should be given more weightage towards grading than what it is given now because this module should be more about hands-on learning than writing exams. 30% weightage for this assignment and 50% for the exam doesn't seem fair, given that we spent a larger share of time and effort working on this assignment. We would like to bring to attention how the prerequisite for this module, EE2020, was graded for us. The module had 20% weightage for two midterms each and had 60% weightage for the CA component which included an FPGA Design Project, with no final exam. This was a much more fun and enjoyable experience as the module emphasized so much on hands-on learning. Even our capstone project that we take after EE2024, namely CG3002, is an 100% CA module graded solely on the project aka embedded system we come up with. Since EE2024 involves building embedded systems just like EE2020 and CG3002 does, emphasis in EE2024 should be given on hands-on learning in a manner similar to these modules. In the long run, we believe that our hands-on learning experience in this module will matter more to us in our profession. We conclude with the hope that this change will be brought into effect when this module is offered to students in future semesters as they will benefit and enjoy learning more from a project-focused experience in this module.

7. Conclusion

For this assignment, we have successfully implemented a Care Unit for The Elderly system (CUTE). The system designed is able to measure the temperature, light and acceleration continuously which can help ensure that the elderly can live in a safe and secure environment. We have also implemented additional features to make the life of the elderly more comfortable, such as the View Log Mode, which can browse through previous values of temperature, luminosity and accelerometer values that were displayed on the OLED and help the elderly even look at previous conditions he may have faced. This is especially useful in situations where the blue and red LEDs are blinking but the current values are within the acceptable threshold and the user would like to see which of the previous values caused the LEDs to blink.

The entire system has been successfully developed to model reality. A message is sent automatically to the Centralized Elderly Monitoring System (CEMS) at regular intervals and in the event of a danger (fire/movement in darkness), the appropriate message is also delivered. We have also used interrupts for light sensor and temperature sensor to make sure the system is more efficient in response. Therefore, we have successfully completed all the basic requirements of this assignment and have also enhanced the CUTE system with useful, advanced features.

---END OF REPORT---