CARE UNIT FOR THE ELEDERLY

EE2024 Assignment Report

WEDNESDAY MORNING LAB

ANNIYA BASKARAN A0141812R

BAGHABRA DANA A0144902L

Table of Contents

1: Introduction

In this assignment, we will be implementing a Care Unit for The Elderly, known as CUTE. The main purpose of the CUTE is to proactively take action to ensure safe ageing. The system achieves its purpose by monitoring the light intensity, movement and temperature of the surroundings regularly and send an alert warning signal to the system when movement in darkness and fire are detected.

In CUTE, there are two main modes called the STABLE mode and MONITOR mode.

- STABLE mode is the first mode that will be entered upon CUTE being powered on. Upon powering on, CUTE displays the following behaviours.
    - The SEGMENT_DISPLAY is off
    - The GRAPHICS_DISPLAY is off
    - BLINK_RED is off
    - BLINK_BLUE is off
    - The TEMPERATURE_SENSOR, LIGHT_SENSOR and ACCELEROMETER are idle and not reading any data.
    - There are no transmissions to CEMS

- MONITOR mode is the second mode that will be entered upon pressing MODE_TOGGLE during STABLE state. MONITOR mode for CUTE depicts the situation when CUTE is being used in a real environment for monitoring the elderly person's surroundings. As soon as CUTE enters MONITOR mode, the TEMPERATURE_SENSOR, LIGHT_SENSOR and ACCELEROMETER are sampled.

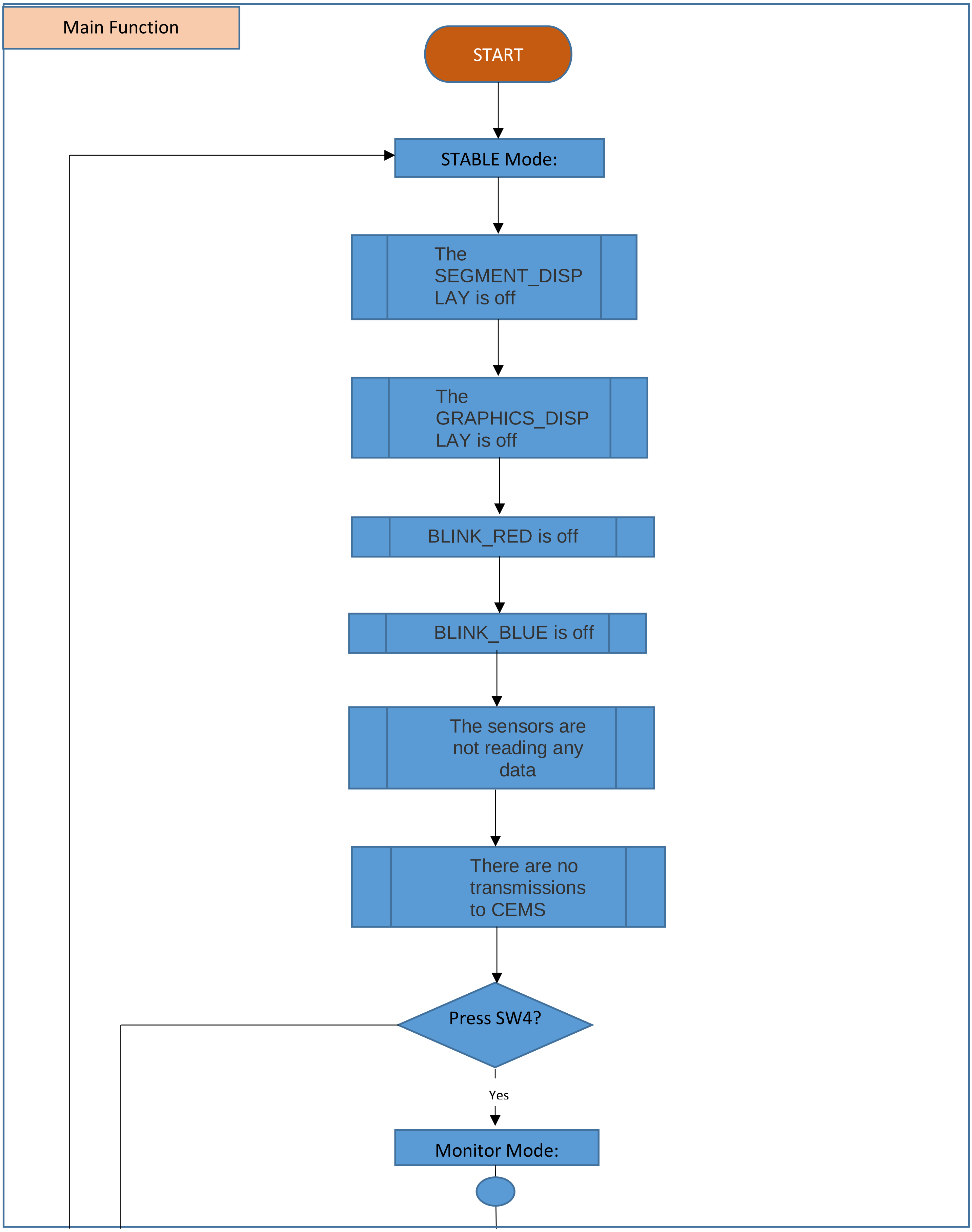
Furthermore, we have extended the CUTE system to incorporate a GAME mode. This mode is entered upon turning on GAME_SWITCH from STABLE or MONITOR mode. Further details about the mode will be provided in the following sections.

The following is a list of devices used for CUTE.

- ACCELEROMETER: MMA7455L
- LIGHT_SENSOR: ISL29003 with LIGHT_RANGE_4000
  TEMPERATURE_SENSOR: MAX6576
- SEGMENT_DISPLAY: 7-segment LED display
- GRAPHICS_DISPLAY: 96x64 White OLED
- BLINK_BLUE: Blue Light for RGB LED, alternating between ON and OFF every 333 milliseconds
- BLINK_RED: Red Light for RGB LED, alternating between ON and OFF every 333 milliseconds
- MODE_TOGGLE: SW4
- CEMS: UART terminal program (Tera Term) on a personal computer
- GAME_SWITCH: SW5

The following is a list of libraries used for CUTE.

- Lib_CMSIS: helper functions for core peripherals (NVIC/interrupts, SysTick etc.)
- Lib_MCU: helper functions for on-chip peripherals (PINSEL, GPIO, I2C, SPI, UART etc.)
- Lib_EaBaseboard: helper functions for external peripherals on the baseboard (light sensor, temperature sensor, OLED display etc.)


Main Function
START
STABLE Mode:
The SEGMENT_DISPLAY is off
The GRAPHICS_DISPLAY is off
BLINK_RED is off
BLINK_BLUE is off
The sensors are not reading any data
There are no transmissions to CEMS
Press SW4?
Yes
Monitor Mode:

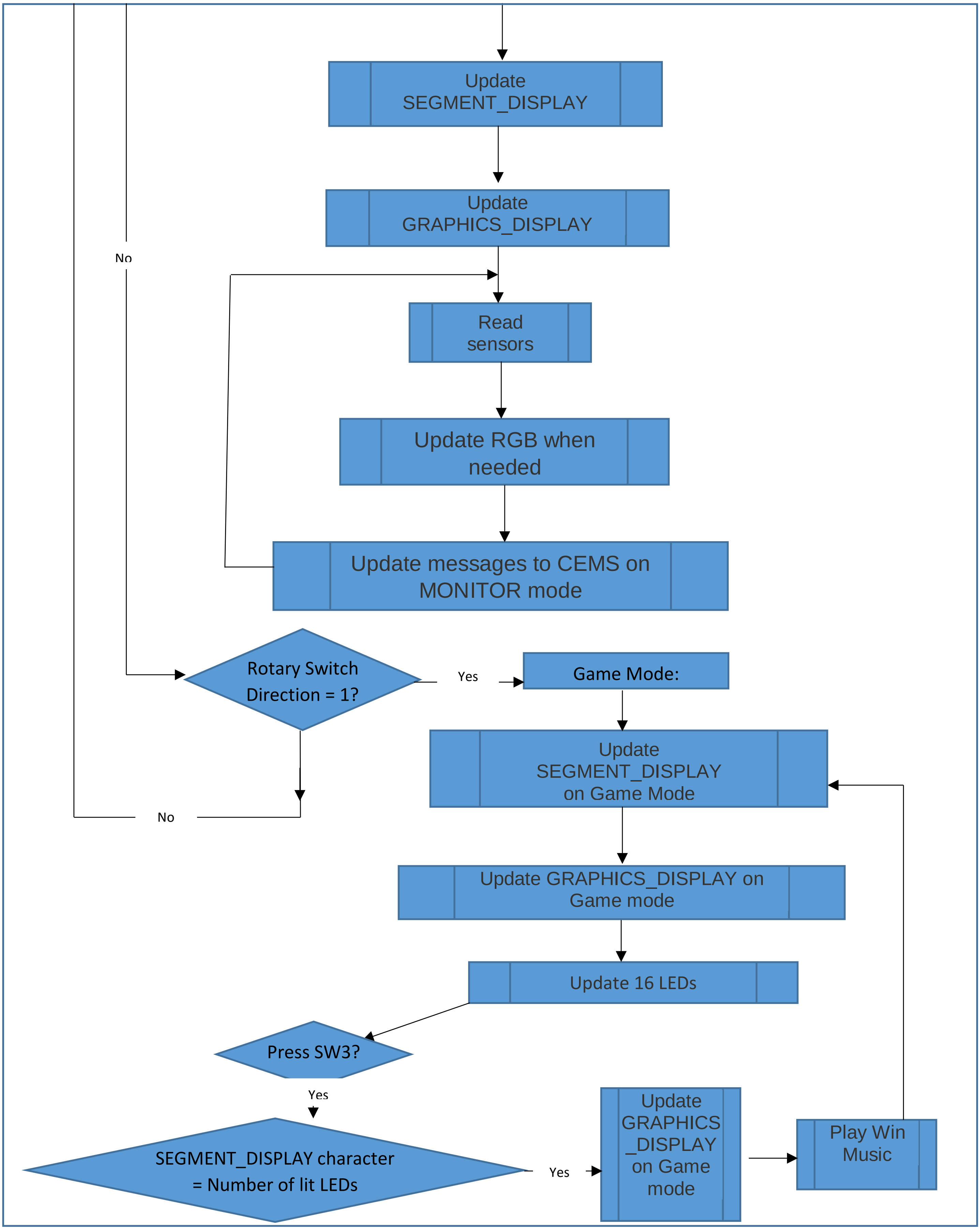


Figure 1: Main System Flowchart

2: MODE_TOGGLE

As mentioned in Section 1, MODE_TOGGLE is used for switching between STABLE and MONITOR modes and it is the pushbutton SW4. It makes use of the GPIO interface. To initialise this SW4 button, we first make use of the void init_GPIO () function to include the following code.

```
static void init_GPIO(void) {

	//Initialize button sw4
	PINSEL_CFG_Type PinCfg;
	PinCfg.Funcnum = 0;
	PinCfg.OpenDrain = 0;
	PinCfg.Pinmode = 0;
	PinCfg.Portnum = 1;
	PinCfg.Pinnum = 31;
	PINSEL_ConfigPin(&PinCfg);
	GPIO_SetDir(1, 1 << 31, 0);
```

…

The Funcnum here represents the function number and since GPIO interface is used, the function number is 0. SW4 is PI01_4 which makes use of P1.31. In order to use SW4, the jumper, J28, should not be inserted. In order to get rid of the debouncing effect upon pressing SW4, `(msTicks - sw4PressedTicks >= 500)` was implemented in the main function.

```
while (1) {
		btnSW4 = (GPIO_ReadValue(1) >> 31) & 0x01;

		if ((btnSW4 == 0) && (msTicks - sw4PressedTicks >= 500)) {
			sw4PressedTicks = msTicks;
```

…

The following code was then implemented to switch between STABLE/GAME mode and MONITOR mode.

```
if ((mode == STABLE) | (mode == GAME)) {
		mode = MONITOR;
		monitorMsg = "Entering MONITOR Mode.\r\n";
		UART_Send(LPC_UART3, (uint8_t *) monitorMsg,
strlen(monitorMsg),BLOCKING);

		 sevenseg_Count = 0;
} else {
		 mode = STABLE;
}
```

2.1: STABLE mode

STABLE mode is the mode CUTE enters upon being powered on. A caretaker of the elderly can use MODE_TOGGLE to switch from MONITOR mode to STABLE mode to indicate his/her presence to take care of the elderly.

During STABLE mode, the SEGMENT_DISPLAY will be off, meaning now showing any character on the screen. The GRAPHICS_DISPLAY will be off, meaning it shows a black screen with no characters on it. The BLINK_BLUE and BLINK_RED will be off, meaning the RGB will not be displaying red and blue light. This also means that the three sensors – TEMPERATURE_SENSOR, LIGHT_SENSOR and ACCELEROMETER will be idle and not read any data. Lastly, there will be no data transmissions to CEMS.

```
case STABLE:
                    led7seg_setChar(NULL, FALSE);
                    oled_clearScreen(OLED_COLOR_BLACK);
                    isFire = 0;
                    isWalkingDark = 0;

                    break;
```

isFire is a boolean like integer variable that becomes 1 when temp_read() is more than 450 (45°C).

- temp_read() is a function that is used in temp.c file to read the temperature read by the temperature sensor

Once isFire is set to 0, the temperature sensor will be idle and the RGB will not blink red.

isWalkingDark is a boolean like integer variable that becomes 1 when light_read() is less than the light intensity of 50 lux and the difference in magnitude of the previous and current accelerometer values is more than 7 which means there is movement.

- light_read() is a function that is used in light.c file to read the light intensity read by the llight sensor

Once isWalkingDark is set to 0, the light sensor and accelerometer will be idle and the RGB will not blink blue.

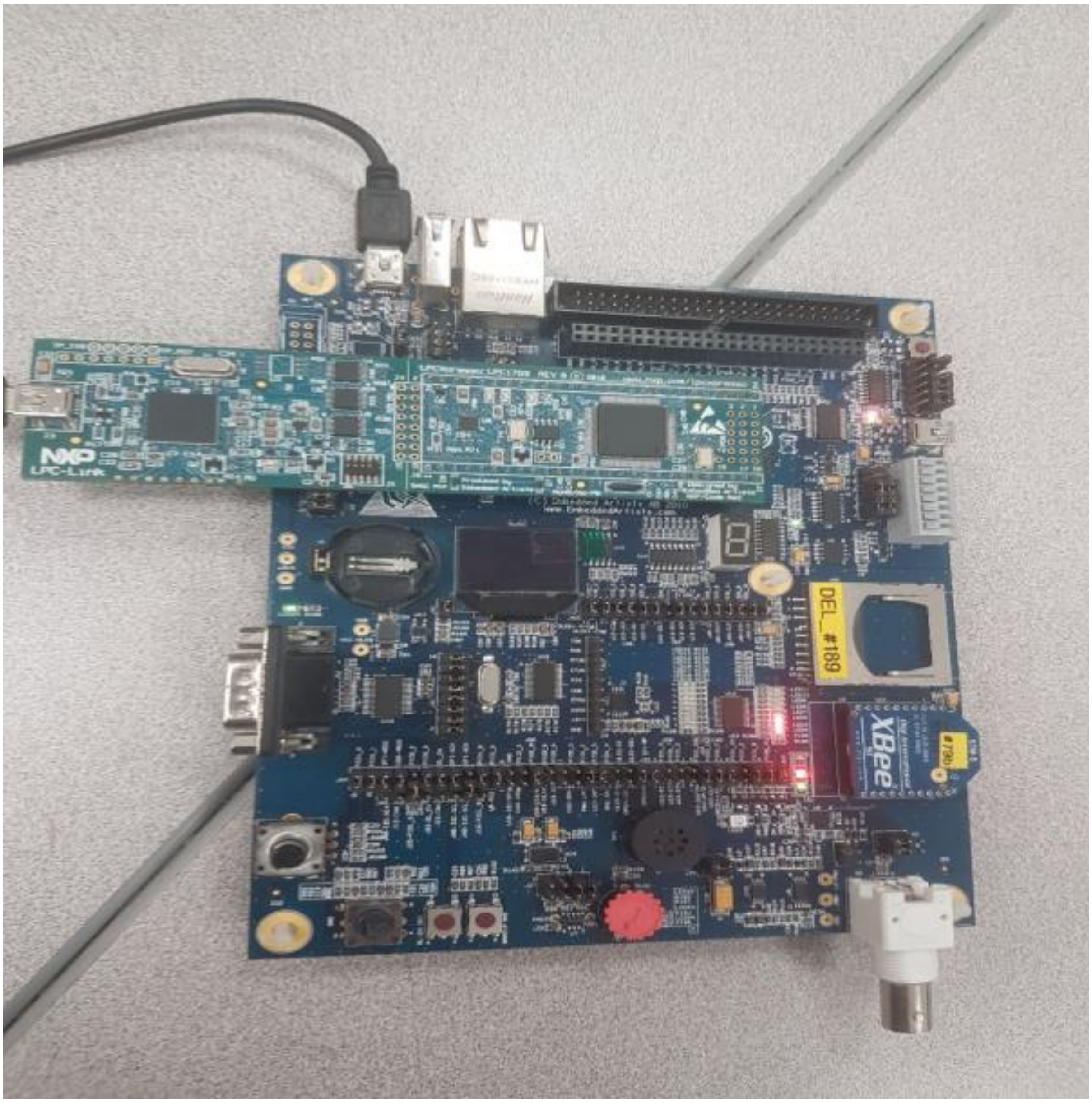


Figure 2.1: LPCXpresso Base Board during STABLE mode

2.2: MONITOR mode

MONITOR mode is the mode CUTE enters once MODE_TOGGLE is pressed during STABLE state. In this mode, all three sensors will be sampled.

Upon entering MONITOR mode, the message "Entering MONITOR Mode. \r\n" will be sent to CEMS. The GRAPHICS_DISPLAY will include the word 'MONITOR' throughout the MONITOR mode. THE SEGMENT_DISPLAY hexadecimal value increases by 1 every second. After the SEGMENT_DISPLAY shows the last value, it restarts with the first value of the hexadecimal system. The sequence is as shown below:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Further details about the updating of SEGMENT_DISPLAY during MONITOR mode will be provided in Section 3.

The TEMPERATURE_SENSOR, LIGHT_SENSOR and ACCELEROMETER are samples and the values are updated on the GRAPHICS_DISPLAY when the SEGMENT_DISPLAY shows:

5 A F

Further details about the updating of GRAPHICS_DISPLAY during MONITOR mode and the three sensors will be provided in Section 4.

Transmission of the currently sampled sensor values from the TEMPERATURE_SENSOR, LIGHT_SENSOR, and ACCELEROMETER, to CEMS, occurs once each time the SEGMENT_DISPLAY shows:

F

The format of the transmitted data should be as follows:

NNN_-_T*****_L*****_AX*****_AY*****_AZ*****\r\n

where the temperature value, light value, and x-axis value, y-axis value, z-axis value of the accelerometer are respectively preceded by T, L, AX, AY, and AZ. NNN represents a 3-digits value that starts from 000 and increments by 001 each time such set of sensor data is transmitted to CEMS from CUTE. NNN never resets itself to 000, unless CUTE itself is powered on from a power off state.

BLINK_BLUE and BLINK_RED can happen simultaneously. BLINK_BLUE and BLINK_RED should not be turned off, unless CUTE goes into stable state.

If BLINK_BLUE is happening at the instant a scheduled transmission to CEMS occurs, the message “Movement in darkness was Detected.\r\n” will also be sent before the usual sensor values from the TEMPERATURE_SENSOR, LIGHT_SENSOR, and ACCELEROMETER.

If BLINK_RED is happening at the instant a scheduled transmission to CEMS is happening, the message “Fire was Detected.\r\n” will also be sent before the usual sensor values from the TEMPERATURE_SENSOR, LIGHT_SENSOR, and ACCELEROMETER.

Messages due to BLINK_RED occurs before messages due to BLINK_BLUE.

More details about the updating of RGB and messages to CEMS will be provided in Sections 5 and 6 respectively.

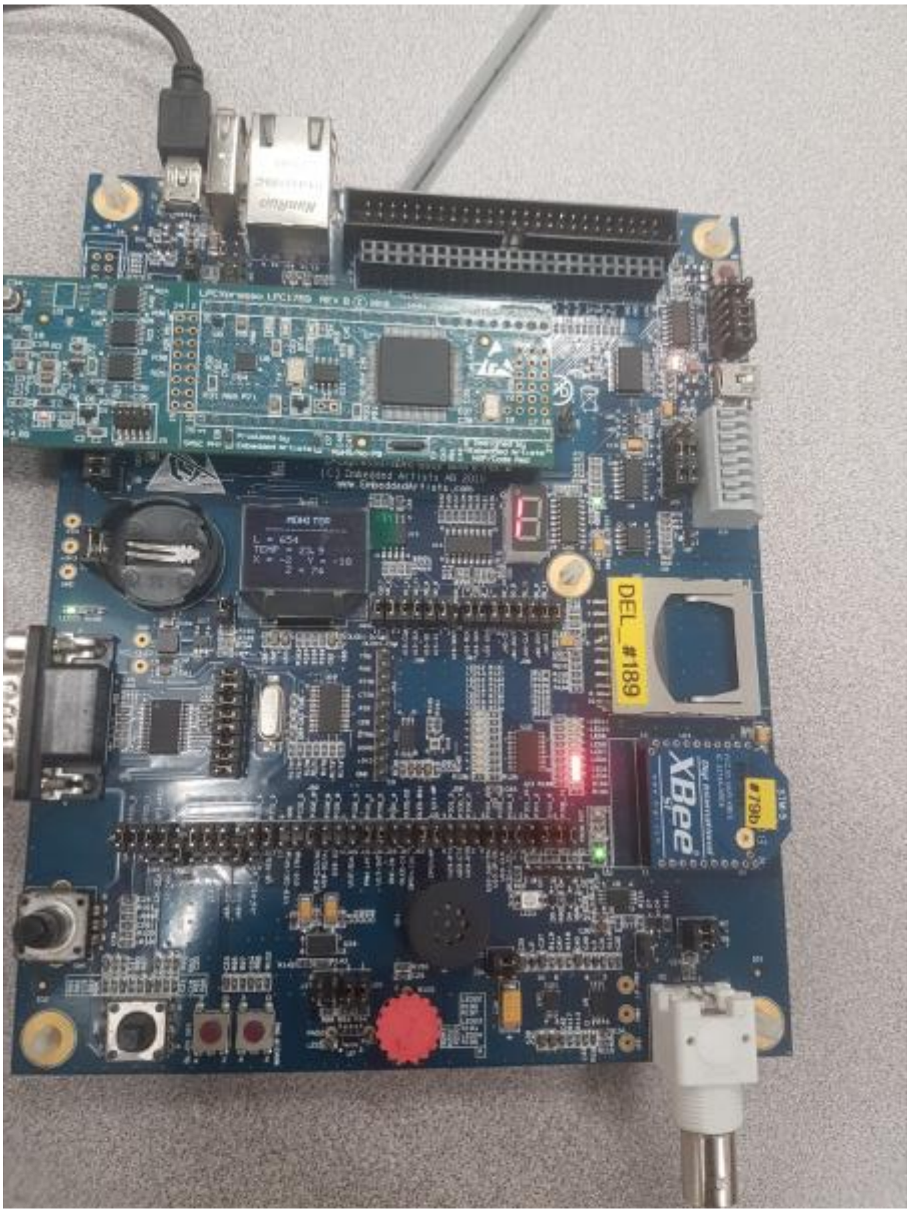


Figure 2.2: LPCXpresso Base Board during MONITOR mode idle on a horizontal surface at room temperature

3: Update SEGMENT_DISPLAY on MONITOR Mode

The 7 segment display is connected to the SPI bus. We made use of a led7seg.h helper file from Lib_EaBaseBoard. In order to increment the 7 segment from 0 to F every second, we used the msTicks from Systick handler. This Systick handler is accessed from the Lib_CMSISv1p30_LPC17xx. A static unsigned 32 bit integer, TICK_RATE_ONE_SEC, is given the value of 1000, meaning 1000 milliseconds (1 second). For code simplicity, a function void sevenSeg() was used to increment the seven segment from 0 to F every second.

```
void sevenSeg(){

      //increment 7 segment
      if ((msTicks - oneSecondTicks) >= TICK_RATE_ONE_SEC) {
            oneSecondTicks = msTicks;
            if (sevenseg_Count > 15) {     //If it is 'F' then rewind to 0
                  sevenseg_Count = 0;
            }

            if (sevenseg_Count >= 10) {
                  sevenseg_Count += 55;
                  sevenseg_Display = '0' + sevenseg_Count;
                  led7seg_setChar(sevenseg_Count, FALSE); //Uses character
65 and after
                  sevenseg_Count = (sevenseg_Count + 1 - 55);
            }

            else {
                  sevenseg_Display = '0' + sevenseg_Count;
                  led7seg_setChar(sevenseg_Display, FALSE); //Uses
characters '1' and after
                  sevenseg_Count = (sevenseg_Count + 1);
            }
      }

}
```

We used three if conditions in this case to make sure that it increases from 0 to F (hexadecimal value) and resets itself after F. sevenseg_Count is an integer value while sevenseg_Display is character. In this case, the sevenseg_Count regularly increases and resets after hitting a value of 15. Upon entering MONITOR mode, the sevenseg_Count value will be 0. After that it increases by 1 every second and sevenseg_Display adds with character ‘0’ and that is being displayed on the led7seg screen. After the sevenseg_Count reaches a value of 10, it needs to be displayed as the alphabet 'A' as A is a hexadecimal value that represents the decimal value 10. It then adds in a 55 as the ASCII value of A is 65. To increment this value, it minuses off 55 first and increments by 1. Once the sevenseg_Count is more than 15, it has to reset to a value of 0 and increment by 1 all over again.

Figure 3 on the next page summarises the cycle of the led7seg display during MONITOR mode.

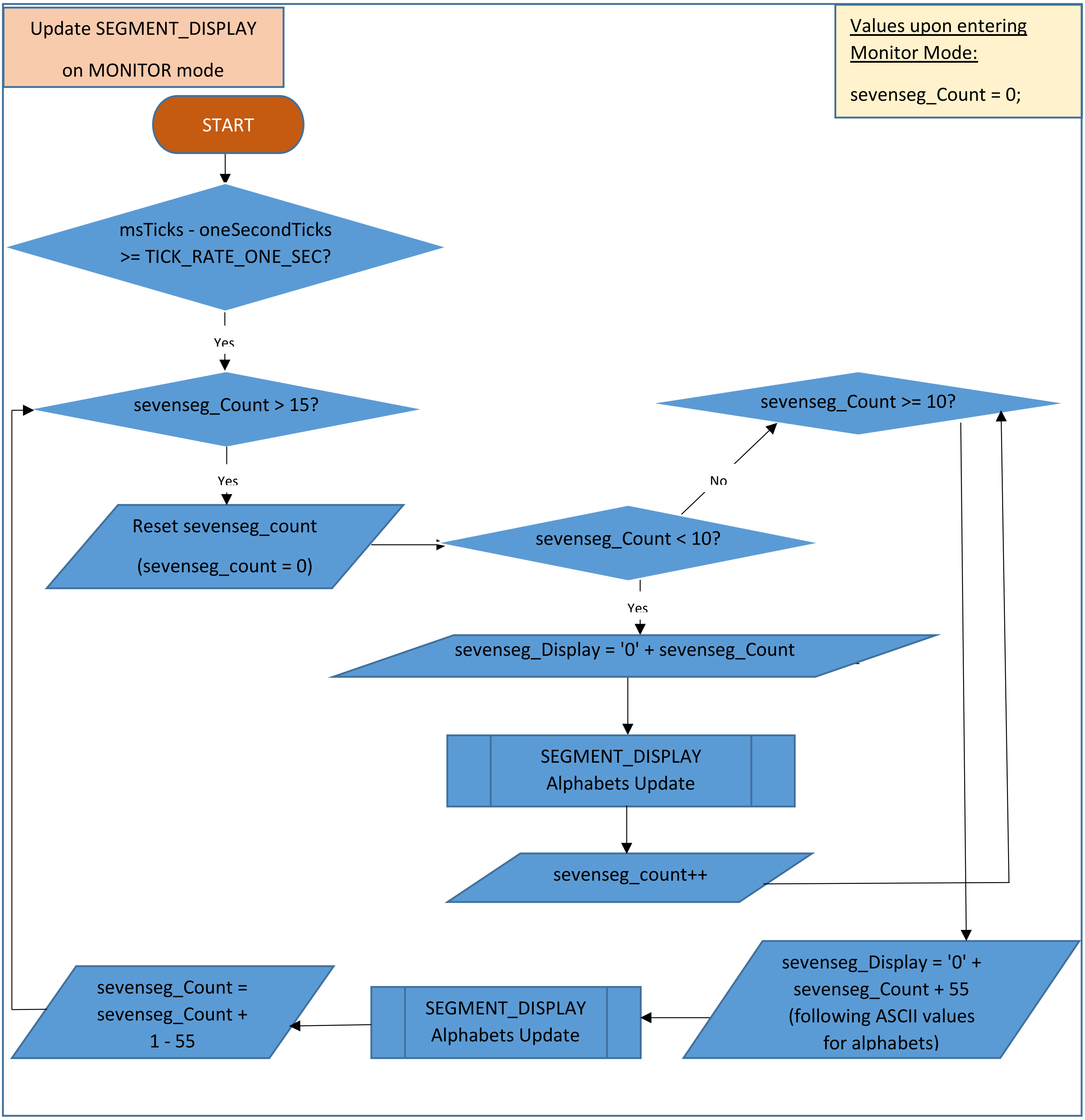


Figure 3: Update SEGMENT_DISPLAY on MONITOR mode Flowchart

4: Update GRAPHICS_DISPLAY on MONITOR mode

The OLED can be connected to the SPI-bus or the I2C-bus. For CUTE, OLED is connected to the SPI-bus. The jumper, J44, must always be inserted for PIO1_10 to control the OLED-voltage. For this interface, the jumpers in J42, J43, J45 pin1-2, J46 pin 1-2 and J58 are inserted for the SPI interface. We made use of a oled.h helper file from Lib_EaBaseBoard. The character strings for OLED during MONITOR mode are

```
//Strings for OLED
int8_t OLED_MODE[15];
int8_t OLED_X[15];
int8_t OLED_Y[15];
int8_t OLED_Z[15];
int8_t OLED_LIGHT[15];
int8_t OLED_TEMPERATURE[15];
```

As mentioned in Section 2.1, before entering the MONITOR mode from STABLE mode, the OLED screen will be cleared.

```
oled_clearScreen(OLED_COLOR_BLACK);
```

Upon entering MONITOR mode, ‘MONITOR’ will be displayed on the OLED screen.

```
sprintf(OLED_MODE, "MONITOR");
```

The light intensity, temperature and accelerometer values will also be displayed on the screen. If the cute enters the MONITOR for the first time after being powered on, the values by the three sensors will be shown still on the screen before the SEGMENT_DISPLAY shows a hexadecimal value of 5 for the first time as the sensors are read before being on STABLE mode. The light intensity, temperature and x, y and z of accelerometer values are updated on the OLED at every ‘5’, ‘A’ and ‘F’ displayed by the SEGMENT_DISPLAY. The light intensity value will be displayed as an integer value. Temperature value will be displayed as float value with one decimal place. The x, y and z values of the accelerometer will be displayed as integer values. All characters displayed on the OLED screen are white in colour with the OLED screen being black.

```
sprintf(OLED_LIGHT, "L = %d", light);
sprintf(OLED_TEMPERATURE, "TEMP = %.1f", temperature / 10.0);
sprintf(OLED_X, "X = %d", x);
sprintf(OLED_Y, "Y = %d", y);
sprintf(OLED_Z, "Z = %d", z);
oled_putString(30, 0, (uint8_t *) OLED_MODE, OLED_COLOR_WHITE,
                              OLED_COLOR_BLACK);
oled_putString(10, 10, "-------------", OLED_COLOR_WHITE,
                              OLED_COLOR_BLACK);
oled_putString(0, 20, (uint8_t *) OLED_LIGHT, OLED_COLOR_WHITE,
                              OLED_COLOR_BLACK);
oled_putString(0, 30, (uint8_t *) OLED_TEMPERATURE,
                              OLED_COLOR_WHITE, OLED_COLOR_BLACK);
oled_putString(0, 40, (uint8_t *) OLED_X, OLED_COLOR_WHITE,
                              OLED_COLOR_BLACK);
oled_putString(50, 40, (uint8_t *) OLED_Y, OLED_COLOR_WHITE,
                              OLED_COLOR_BLACK);
oled_putString(30, 50, (uint8_t *) OLED_Z, OLED_COLOR_WHITE,
                              OLED_COLOR_BLACK);
```

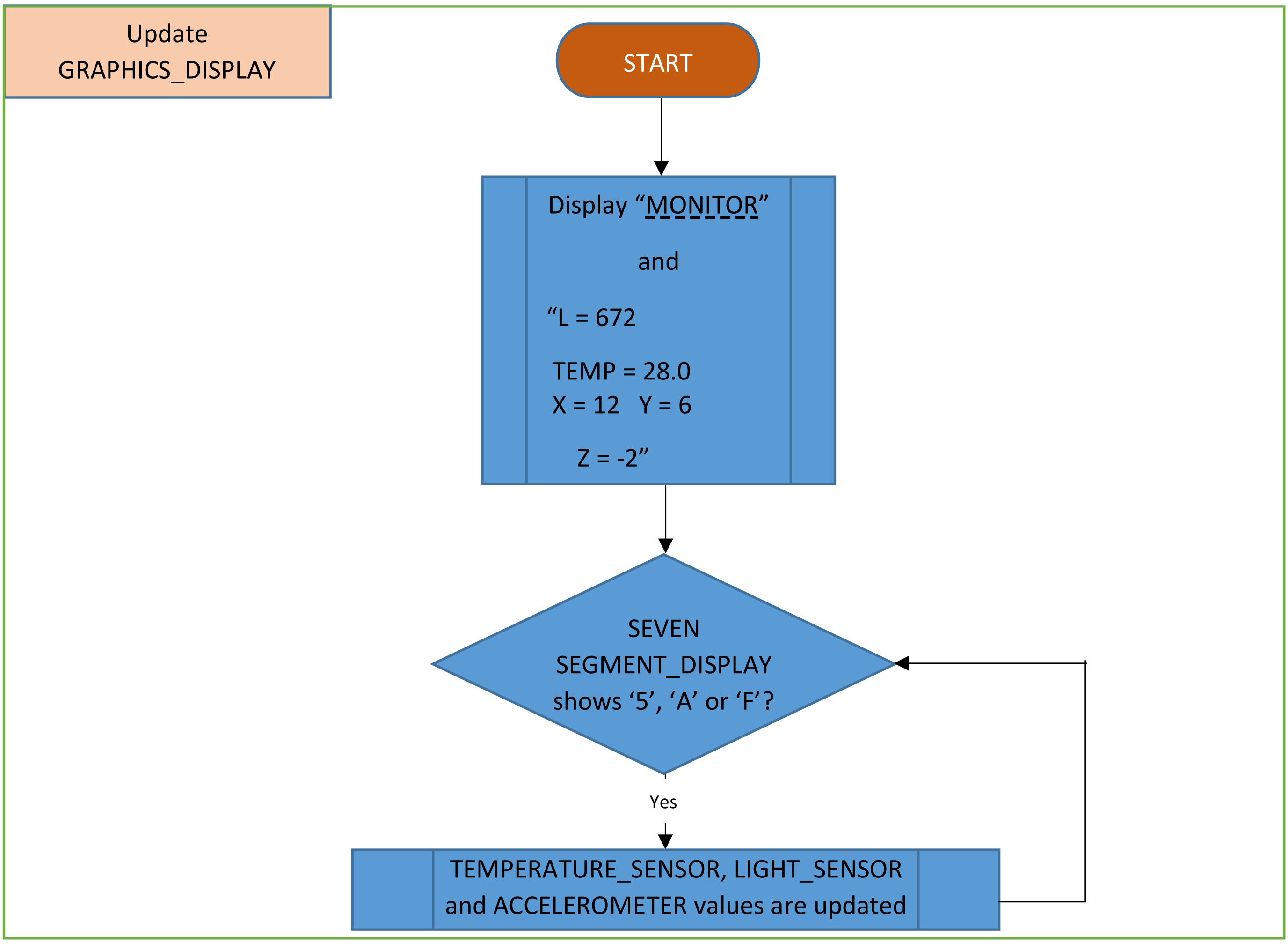


Figure 4: Update GRAPHICS_DISPLAY on MONITOR mode Flowchart

5: Update RGB on MONITOR mode

RGB is used mainly on MONITOR mode to indicate the presence of fire and the elderly's movement in darkness. The colours displayed by RGB are BLUE and RED. As such, the ledMask is set accordingly to the GPIO pin configurations of the colours.

```
void setRGB(uint8_t ledMask) {
	if (ledMask == RED) {
		GPIO_SetValue(2, (1 << 0));
	} else {
		GPIO_ClearValue(2, (1 << 0));
	}
	if (ledMask == BLUE) {
		GPIO_SetValue(0, (1 << 26));
	} else {
		GPIO_ClearValue(0, (1 << 26));
	}
}
```

For CUTE, it does BLINK_RED when fire is detected. The detection of fire is using temperature sensor which will be explained in Section 5.1. Since BLINK_RED is the red light for RGB LED that alternates between ON and OFF every 333 milliseconds. To alternate between ON and OFF every 333 milliseconds, Timer0_Wait, a function from the Lib_MCU, is used to delay the state of the RGB LED for 333 milliseconds.

```
void blink_RED(){
	setRGB(RED);
	Timer0_Wait(333);
	setRGB(0);
	Timer0_Wait(333);
}
```

The CUTE does BLINK_BLUE when movement in darkness is detected. The detection of movement in darkness is using light sensor and accelerometer which will be explained in Sections 5.2 and 5.3. Since BLINK_BLUE is the blue light that alternates between ON and OFF every 333 milliseconds, the following code is implemented.

```
void blink_BLUE(){
	setRGB(BLUE);
	Timer0_Wait(333);
	setRGB(0);
	Timer0_Wait(333);
}
```

Moreover, the CUTE does BLINK_BLUE and BLINK_RED at the same time when both fire and movement in darkness are detected. During such cases, BLINK_PURPLE happens since PURPLE is the combination of BLUE and RED.

```
void blink_PURPLE(){
	GPIO_SetValue(2, (1 << 0));
	GPIO_SetValue(0, (1 << 26));
	Timer0_Wait(333);
	GPIO_ClearValue(2, (1 << 0));
	GPIO_ClearValue(0, (1 << 26));
	Timer0_Wait(333);
}
```

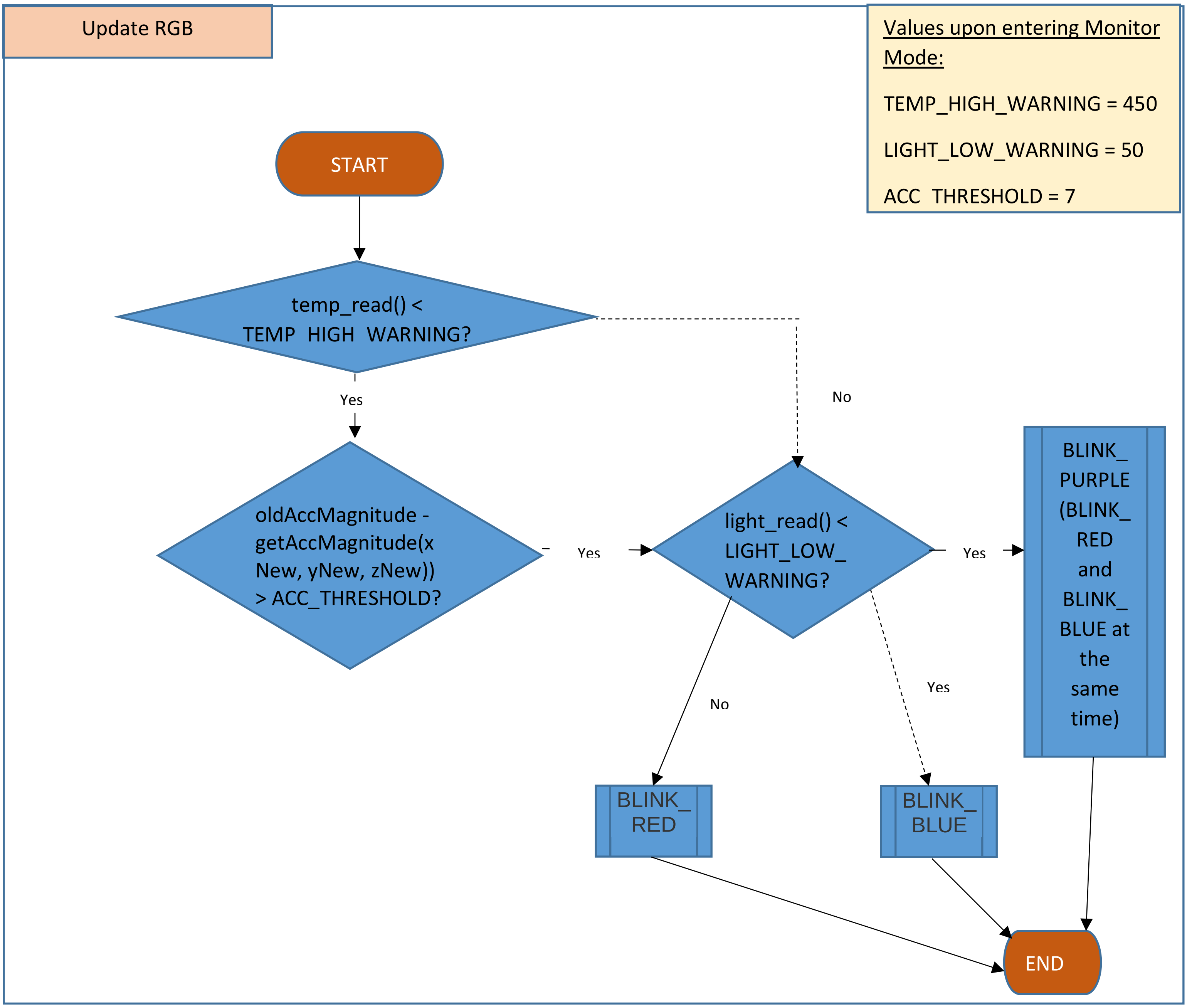


Figure 5: Update RGB on MONITOR mode flowchart

5.1: TEMPERATURE_SENSOR

To enable TEMPERATURE_SENSOR, we used temp_read() from the temp.c file. This gets the temperature read by TEMPERATURE_SENSOR. The value read by the temperature is 10 times the actual temperature of the surroundings in °C. For CUTE, when the TEMPERATURE_SENSOR reads a value more than 450, which is 45°C, the RGB will BLINK_RED to indicate detection of fire.

```
//Indicate Fire
                  if (temp_read() > TEMP_HIGH_WARNING) {
                        isFire = 1;
                  }
…

else if(isFire){
                        blink_RED();
                  }
```

In addition, for simplicity of code, readSensors() function was created to read all the three sensors at once.

```
//Reads sensors
void readSensors(uint32_t* temperature, uint32_t* light, int8_t* x, int8_t*
y,
            int8_t* z) {
      *temperature = temp_read();
      *light = light_read();
      acc_read(&*x, &*y, &*z);
}
```

For the purpose of showing that the fire detection works, we reduced the TEMP_HIGH_WARNING that was defined to be 450 to 50 (5°C). This means that when this change is made, the baseboard will BLINK_RED even at room temperature.

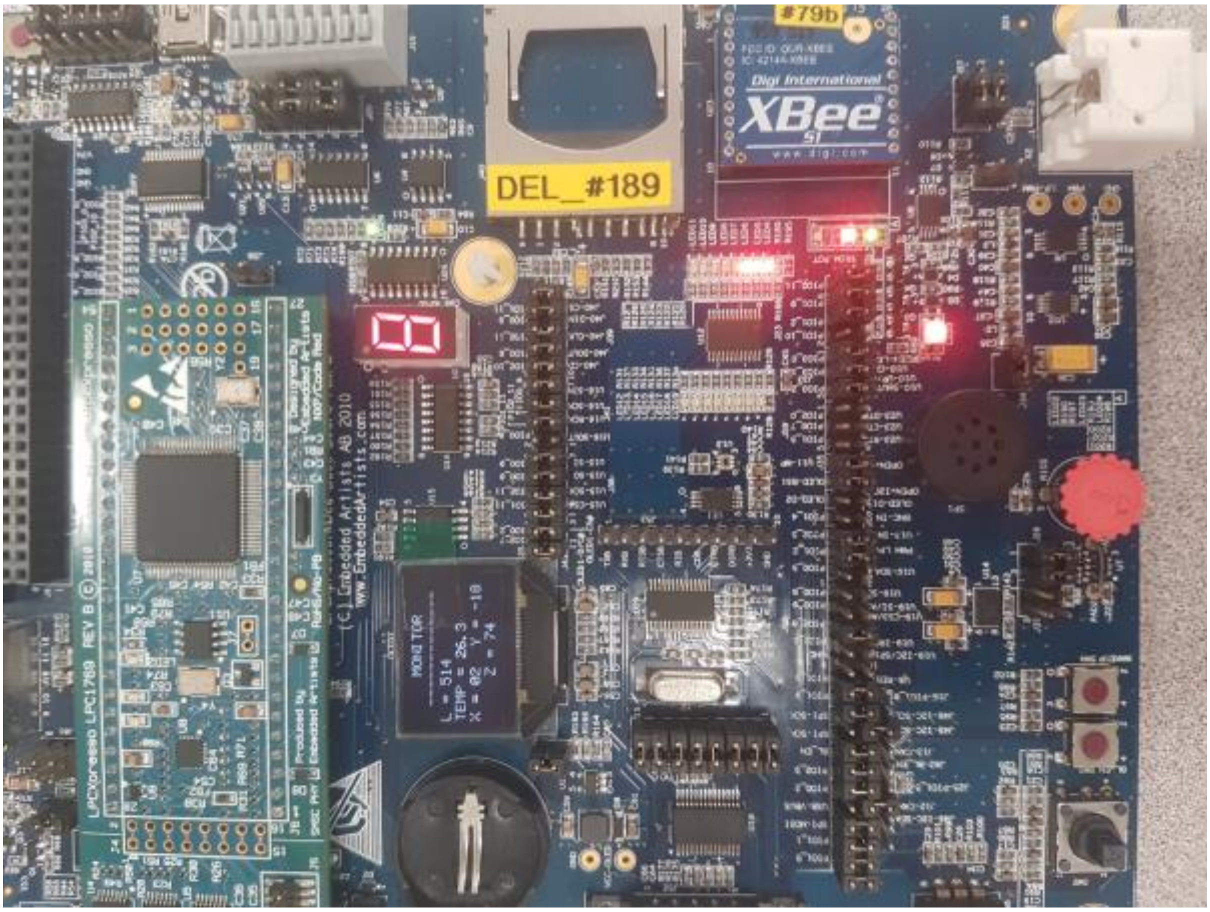


Figure 5.1: LPCXpresso Base Board doing BLINK_RED during MONITOR mode

5.2: LIGHT_SENSOR

The LIGHT_SENSOR makes use of the I2C interface. To enable LIGHT_SENSOR, we used light_read() from the light.c file. This gets the light read by LIGHT_SENSOR. For CUTE, when the LIGHT_SENSOR reads a value less than 50 lux, it means the device is in a dark place.

An interrupt service routine was also created to increase efficiency of CUTE. Since light sensor interrupt uses I2C, we made sure to add the initialization code below to set up I2C and choose function 2 based on the schematics in the Baseboard Briefing manual.

```
static void init_i2c(void) {
	PINSEL_CFG_Type PinCfg;

	/* Initialize I2C2 pin connect */
	PinCfg.Funcnum = 2;
	PinCfg.Pinnum = 10;
	PinCfg.Portnum = 0;
	PINSEL_ConfigPin(&PinCfg);
	PinCfg.Pinnum = 11;
	PINSEL_ConfigPin(&PinCfg);

	// Initialize I2C peripheral
	I2C_Init(LPC_I2C2, 100000);

	/* Enable I2C1 operation */
	I2C_Cmd(LPC_I2C2, ENABLE);
}
```

And to configure the hardware for light sensor interrupt, we inserted a jumper at PIO2_5 (P2.5). Also, light sensor interrupt cannot go through the NVIC directly, since it is an external peripheral. Thus, it goes through the GPIO as seen in the function 0, thus being initialized in init_GPIO().

```
	//light sensor int
	PinCfg.Portnum = 2;
	PinCfg.Pinnum = 5;
	PinCfg.Funcnum = 0;
	PINSEL_ConfigPin(&PinCfg);
	GPIO_SetDir(2, 1 << 5, 0);
```

Moreover, in the main(), We have set the range as 1000, as the sensor will return a value from 0 to 972. Since for our assignment, the trigger value is 50, we set the low threshold to be 50 and high threshold to be 1000. Hence, the interrupt will only give an interrupt under the value of 50 lux.

```
// Setup light limit for triggering interrupt
light_setRange(LIGHT_RANGE_1000);
light_setLoThreshold(lightLoLimit);
light_setHiThreshold(lightHiLimit);
light_setIrqInCycles(LIGHT_CYCLE_1);
light_clearIrqStatus();
```

As for light_setIrqInCycles(LIGHT_CYCLE_1). This is to insure the denouncing of the light sensor readings. And light_clearIrqStatus() simply clear the I2C.

```
LPC_GPIOINT ->IO2IntClr |= 1 << 5;
LPC_GPIOINT ->IO2IntEnF |= 1 << 5; //light sensor
light_enable();

NVIC_ClearPendingIRQ(EINT3_IRQn);
NVIC_EnableIRQ(EINT3_IRQn);
```

5.3: ACCELEROMETER

The ACCELEROMETER makes use of the I2C interface. To enable ACCELEROMETER, we used acc_read() from the acc.c file. This gets the x, y and z values read by ACCELEROMETER. For CUTE, a movement had to be detected. To detect movement, the difference between the magnitudes of the previous x, y and z values and current x, y and z values is calculated. The ACCELEROMETER reads x, y and z values of the elderly's position at every iteration. To calculate magnitude, a getAccMagnitude() function was created that square roots the sum of the squares of x,y and z.

```
//Gets Acceleration Magnitudes
int getAccMagnitude(int x, int y, int z) {
      //    acc_read(&*x, &*y, &*z);
      return sqrt(pow(x, 2) + pow(y, 2) + pow(z, 2));
}
```

An integer variable oldAccMagnitude is used to store the value of the magnitude of the previous x, y and z values.

```
acc_read(&xOld, &yOld, &zOld);
oldAccMagnitude = getAccMagnitude(xOld, yOld, zOld);
```

When the difference in magnitudes is more than 7, a movement is detected. CUTE detects a movement in darkness when the LIGHT_SENSOR reads a value lower than the LIGHT_LOW_WARNING, which is 50 lux, and the difference in magnitudes read by the accelerometer is more than the ACC_THRESHOLD, which is 7.

```
//Indicate walking in dark
                  acc_read(&xNew, &yNew, &zNew);
                  if ((isDark = 1) && ((oldAccMagnitude -
getAccMagnitude(xNew, yNew, zNew)) > ACC_THRESHOLD)) {
                        isWalkingDark = 1;
                  }
```

To indicate that the person is moving in darkness, CUTE does BLINK_BLUE.

```
else if(isWalkingDark){
                        blink_BLUE();
                  }
```

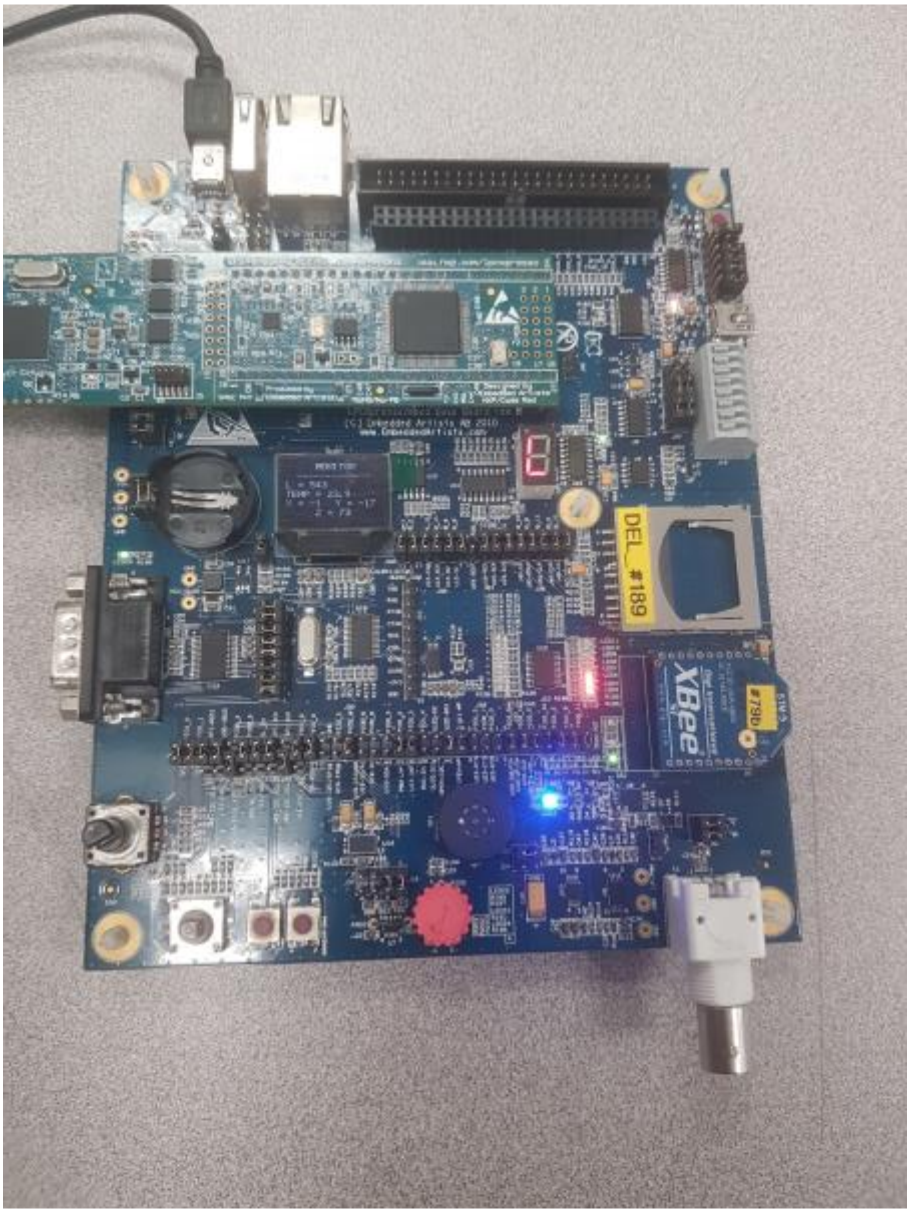


Figure 5.3.1: LPCXpresso Base Board doing BLINK_BLUE during MONITOR mode

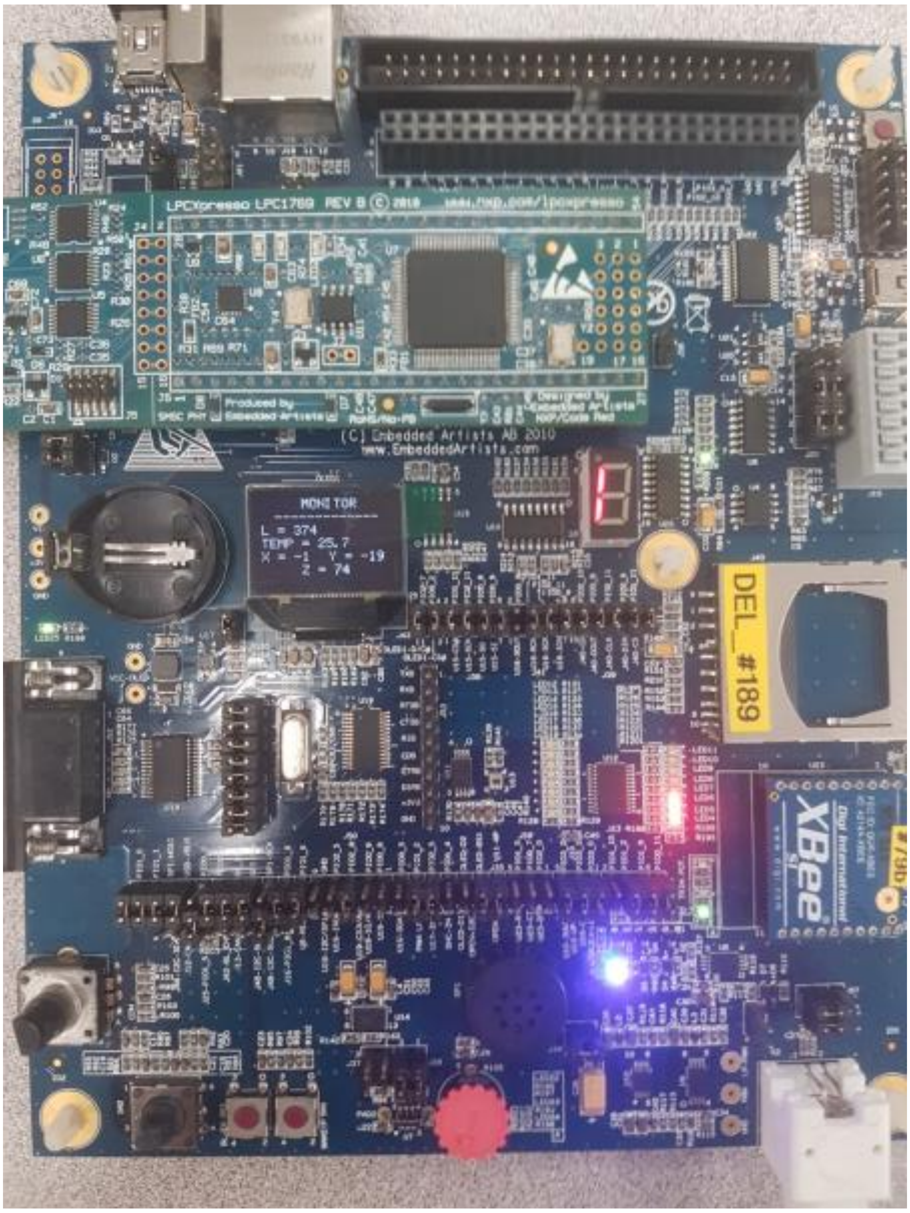


Figure 5.3.2: LPCXpresso Base Board doing BLINK_PURPLE during MONITOR mode

6: Update messages to CEMS on MONITOR mode

The messages sent to CEMS make use of the UART interface. The following code is the code for configuring the UART.

```
void pinsel_uart3(void) {
	PINSEL_CFG_Type PinCfg;
	PinCfg.Funcnum = 2;
	PinCfg.Pinnum = 0;
	PinCfg.Portnum = 0;
	PINSEL_ConfigPin(&PinCfg);
	PinCfg.Pinnum = 1;
	PINSEL_ConfigPin(&PinCfg);
}

void init_uart(void) {
	UART_CFG_Type uartCfg;
	uartCfg.Baud_rate = 115200; //rate of data transfer
	uartCfg.Databits = UART_DATABIT_8;
	uartCfg.Parity = UART_PARITY_NONE;
	uartCfg.Stopbits = UART_STOPBIT_1;
	//pin select for uart3
	pinsel_uart3();

	//Configure Xbee *Baudrate = 9600bps 8N1
	UART_ConfigStructInit(&uartCfg);

	//supply power and setup working parts for uart3
	UART_Init(LPC_UART3, &uartCfg);
	//enable transmit for uart3
	UART_TxCmd(LPC_UART3, ENABLE);
}
```

The highlighted code configures the wireless UART.

For CUTE, upon entering MONITOR mode each time, “Entering MONITOR Mode. \r\n” is sent to CEMS.

```
mode = MONITOR;
monitorMsg = "Entering MONITOR Mode.\r\n";
UART_Send(LPC_UART3, (uint8_t *) monitorMsg, strlen(monitorMsg),
       BLOCKING);
```

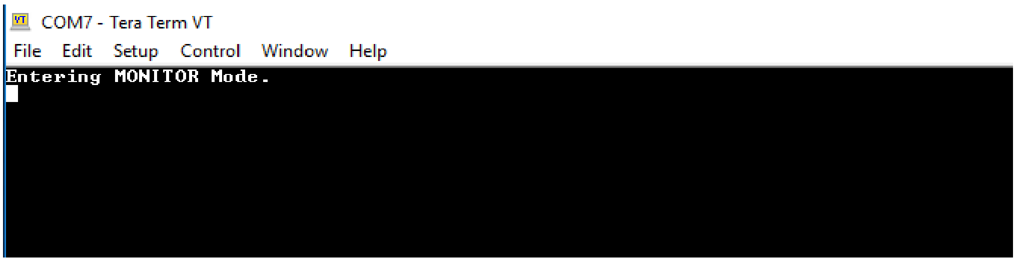


Figure 6.1: Entering MONITOR Mode on Tera Term

Transmission of the currently sampled sensor values from the TEMPERATURE_SENSOR, LIGHT_SENSOR, and ACCELEROMETER, to CEMS, occurs once each time the SEGMENT_DISPLAY shows ‘F’.

```
if (sevenseg_Count == 16) {
```

The transmitted data follow the following format:

NNN_-_T*****_L*****_AX*****_AY*****_AZ*****\r\n

```
sprintf(result, "%d%d%d_-_T%.1f_L%d_AX%d_AY%d_AZ%d\r\n", N[0],
            N[1], N[2], temperature / 10.0, light, x, y, z);
UART_Send(LPC_UART3, (uint8_t *) result, strlen(result),
            BLOCKING);
```

Figure 6.2: Transmitted Data on Tera Term

NNN represents a 3-digits value that starts from 000 and increments by 001 each time such set of sensor data is transmitted to CEMS from CUTE. NNN never resets itself to 000, unless CUTE itself is powered on from a power off state. It is assumed that 999 will never be reached.

To implement this, a two dimensional integer array called N was used. It is initialised as {0, 0, 0} upon being powered on.

```
int N[3] = { 0, 0, 0 };
…
//Setup counter for NNN in Transmission
     N[2]++;
     if (N[2] == 10) {
            N[2] = 0;
            N[1]++;
    }

     if (N[1] == 10) {
            N[1] = 0;
            N[0]++;
}
```

If BLINK_RED is happening at the instant a scheduled transmission to CEMS is happening, the following message is sent before the usual sensor values from the TEMPERATURE_SENSOR, LIGHT_SENSOR, and ACCELEROMETER:

Fire was Detected.\r\n

```
if (isFire) {
	fireMsg = "Fire was Detected.\r\n";
	UART_Send(LPC_UART3, (uint8_t *) fireMsg, strlen(fireMsg),
		BLOCKING);
}
```

If BLINK_BLUE is happening at the instant a scheduled transmission to CEMS occurs, the following message is sent before the usual sensor values from the TEMPERATURE_SENSOR, LIGHT_SENSOR, and ACCELEROMETER:

Movement in darkness was Detected.\r\n

```
if (isWalkingDark) {
	darkMsg = "Movement in darkness was Detected.\r\n";
	UART_Send(LPC_UART3, (uint8_t *) darkMsg, strlen(darkMsg),
		BLOCKING);

}
```

Messages due to BLINK_RED occurs before messages due to BLINK_BLUE.

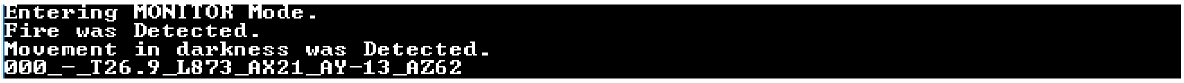


Figure 6.3: Messages sent to CEMS

```
//Send transmission to UART at 'F'
if (sevenseg_Count == 16) {
	if (isFire) {
		fireMsg = "Fire was Detected.\r\n";
		UART_Send(LPC_UART3, (uint8_t *) fireMsg, strlen(fireMsg),
				BLOCKING);
	}
	if (isWalkingDark) {
		darkMsg = "Movement in darkness was Detected.\r\n";
		UART_Send(LPC_UART3, (uint8_t *) darkMsg, strlen(darkMsg),
				BLOCKING);
	}
	sprintf(result, "%d%d%d_-_T%.1f_L%d_AX%d_AY%d_AZ%d\r\n", N[0],
			N[1], N[2], temperature / 10.0, light, x, y, z);
	UART_Send(LPC_UART3, (uint8_t *) result, strlen(result),
			BLOCKING);

	//Setup counter for NNN in Transmission
	N[2]++;
	if (N[2] == 10) {
		N[2] = 0;
		N[1]++;
```

```
        }

        if (N[1] == 10) {
              N[1] = 0;
              N[0]++;
```

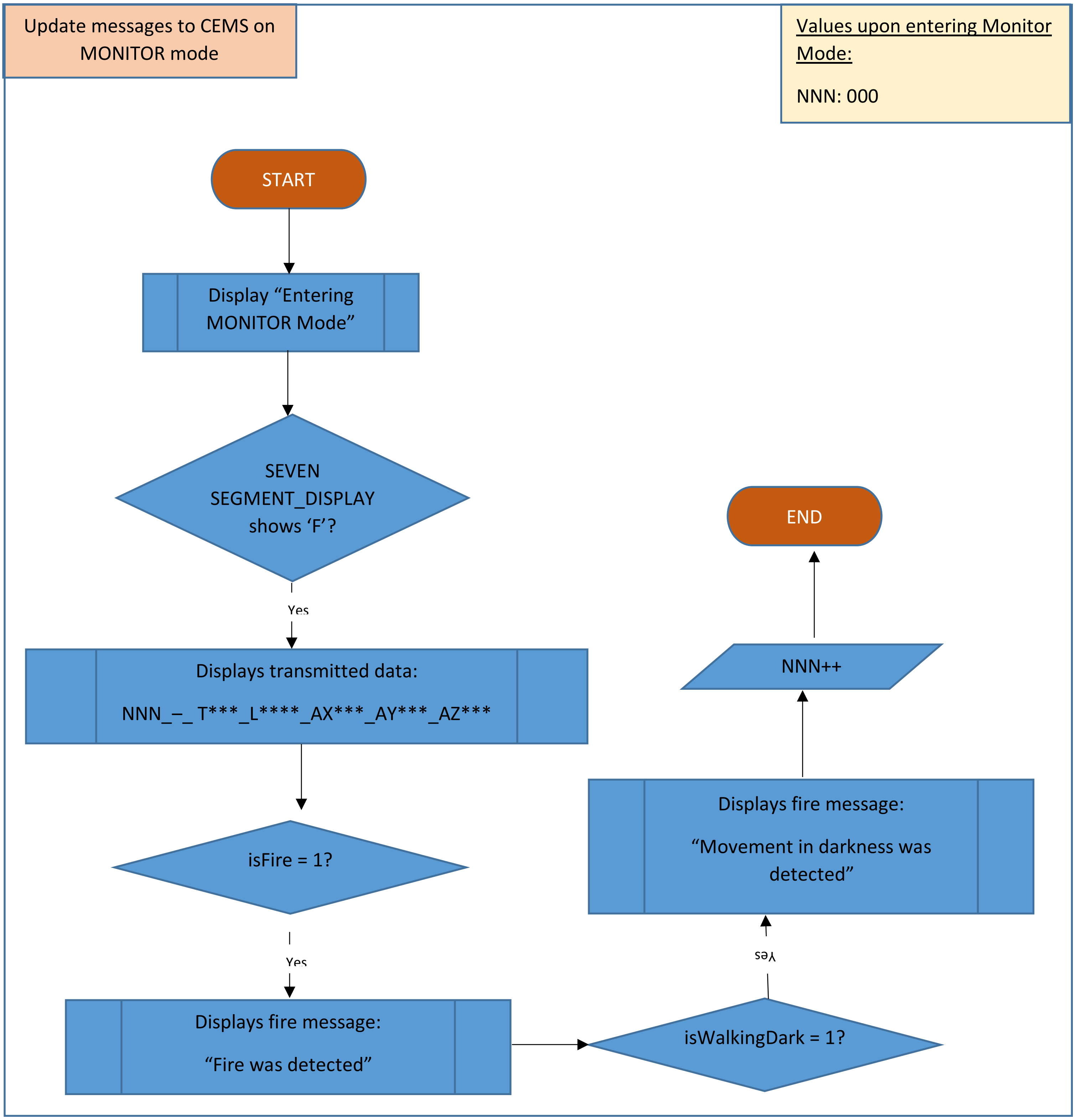


Figure 6.4: Update messages to CEMS during MONITOR mode flowchart

7: GAME mode

As mentioned in Section 1, CUTE system is extended to incorporate a GAME mode. This mode is entered upon turning on GAME_SWITCH, which is the rotary switch, from STABLE or MONITOR mode.

Upon entering GAME mode, random hexadecimal values will be displayed on SEGMENT_DISPLAY and the 16 LED array will light up one by one and reset once all have lit up. The elderly wins the game when he or she presses SW3 at the instant the number of SEGMENT_DISPLAY is equal to the number of LEDs lit up.

7.1: GAME_SWITCH

GAME_SWITCH is used for switching between STABLE/MONITOR and GAME modes and it is the rotary switch SW5. To enable this switch, both jumpers in J56 have to be inserted. The helper file rotary.h was included in the code. An integer rotary direction was used in setting rotary direction in the void function.

```
//Sets rotary direction to change between STABLE and GAME modes
void setRotaryDirection(int flag){

	if(flag == 1){
		rotary_dir = rotary_read();
		if((rotary_dir == 1)){
			mode = GAME;
			score = 0;

		}
		if((rotary_dir == 2)){
			mode = STABLE;
		}
		else if((rotary_dir == 0)){
			if(prevVal == 1){
				mode = GAME;
				score = 0;
			}
			else if(prevVal==2){
				mode = STABLE;
			}
		}
		LPC_GPIOINT->IO0IntClr = 1<<24;
		LPC_GPIOINT->IO0IntClr = 1<<25;
	}
}
```

An interrupt service routine is also set up for this switch to make switching to GAME mode more efficient.

```
//rotary
      LPC_GPIOINT->IO0IntClr = 1<<24;
      LPC_GPIOINT->IO0IntEnF |= 1<<24;
      LPC_GPIOINT->IO0IntClr = 1<<25;
      LPC_GPIOINT->IO0IntEnF |= 1<<25;

      NVIC_ClearPendingIRQ(EINT3_IRQn);
      NVIC_EnableIRQ(EINT3_IRQn);
```

In the EINT3 handler, you need to identify exactly which GPIO interrupt triggered your EINT3 handler. This is because multiple EINT3 interrupts can come from port 0 or port 2. Anyway, you identify the specific GPIO interrupt by checking the GPIO interrupt status register as shown below.

Interrupt lines to NVIC run only from the internal peripherals/controllers*. Interrupts from external peripherals (such as the light sensor interrupts), if needed, will have to be brought in through GPIO or EINTx (x=0,1,2,3), and can only be used to trigger GPIO or EINTx interrupts.

For light sensor if an interrupt occurs, then it will set the flag isDark to be 1. Then clears the interrupt. Similarly, the rotary interrupts clear then sets the flag in the rotary function to be 1. We made sure our Interrupt Service Routines was as short as possible so that our main code cannot run while the control is in an interrupt handler.

7.2: Update SEGMENT_DISPLAY on GAME mode

The update is similar to the concept explained in Section 3. However, the algorithm on how it update on GAME mode is different as it generates random hexadecimal values every second instead of increment.

```
sevenseg_Count_Game = rand() % 16 + 0;
```

The if conditions mentioned in Section 3 still apply here but only the condition on the sevenseg_Count reaching a value more than 15 does not apply here as the range for the random values is [0, F].

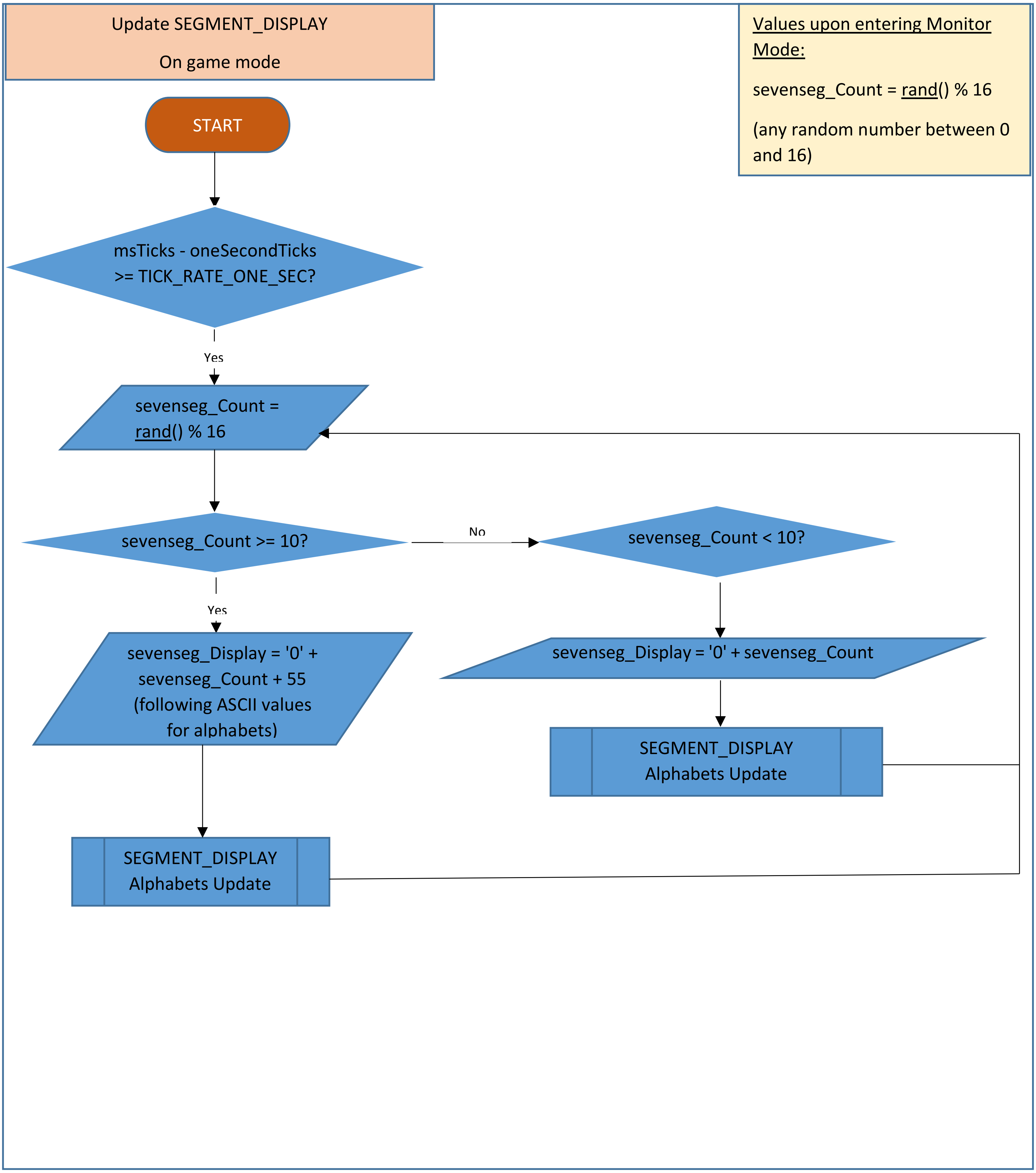


Figure 7.2: Update SEGMENT_DISPLAY on GAME mode

7.3: 16 LEDs

As mentioned in Section 7, the 16 LEDs light up one by one every second and reset once all LEDs have lit up.

```
countLed = 0;
while (count <= 65535) {
...

      if ((msTicks - oneSecondTicks) >= TICK_RATE_ONE_SEC) {
                  oneSecondTicks = msTicks;
                  …
                  pca9532_setLeds(count, 0xffff);
                  count = (2 * count) + 1;
                  countLed++;

…
```

countLed is an integer variable that keeps track of the numbers of LEDs that have lit up. When countLed is equal to the sevenseg_Count and SW3 is pressed, the player wins the game.

```
if ((btnSW3 == 0) && (countLed == sevenseg_Count_Game)) {
            oled_clearScreen(OLED_COLOR_BLACK);
            sprintf(scoreOled, "Score: %d\r\n", score);
            oled_putString(10, 10, scoreOled, OLED_COLOR_WHITE,
                                          OLED_COLOR_BLACK);
            oled_putString(10, 20, "--------------", OLED_COLOR_WHITE,
                                          OLED_COLOR_BLACK);
            oled_putString (20, 50, "YOU WIN!", OLED_COLOR_WHITE,
OLED_COLOR_BLACK);
            playSong(win);

…
```

7.4: Update GRAPHICS_DISPLAY on GAME mode

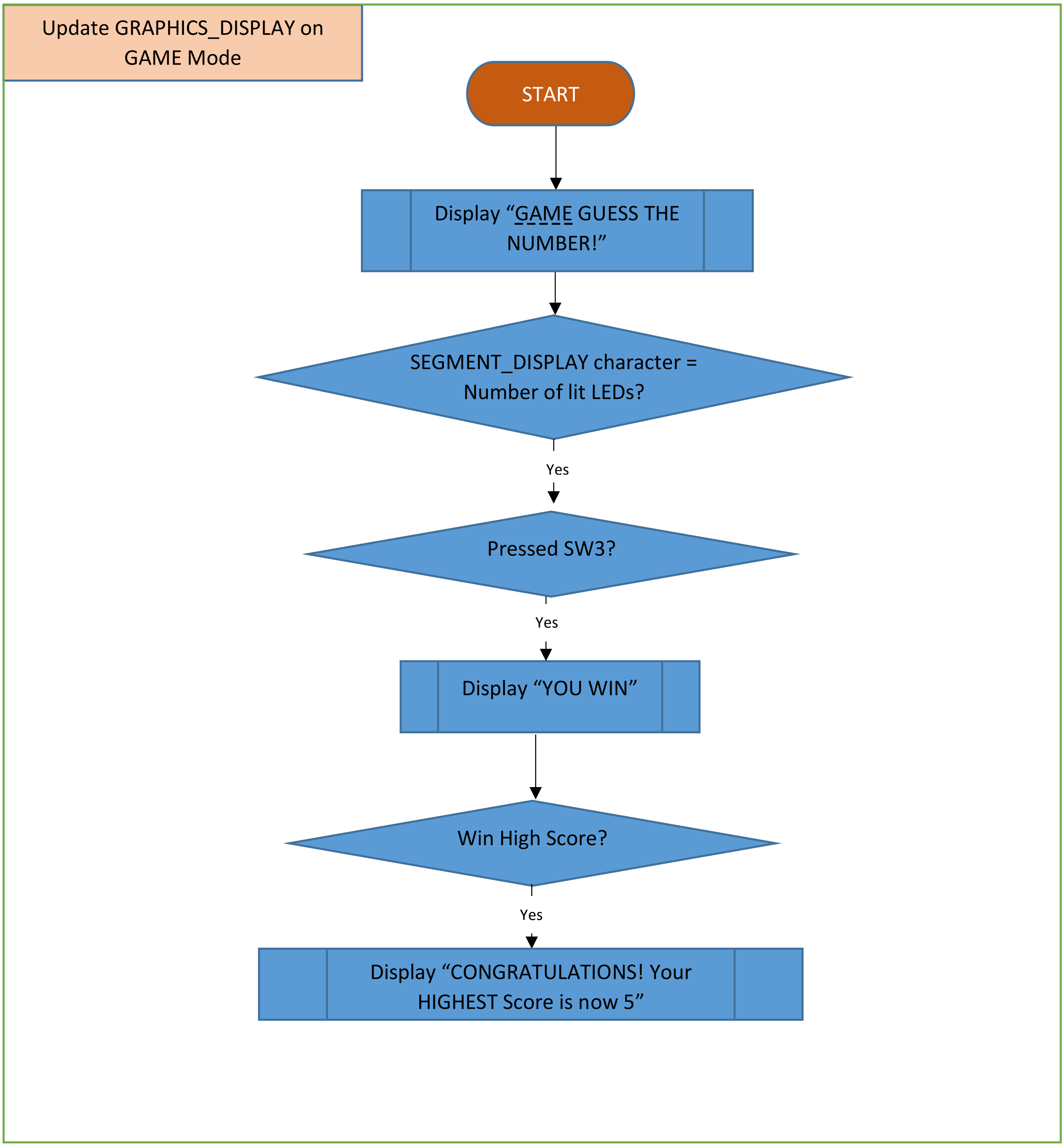


Figure 7.4: Update GRAPHICS_DISPLAY on GAME mode

```
oled_clearScreen(OLED_COLOR_BLACK);
oled_putString(30, 0, "GAME", OLED_COLOR_WHITE,
        OLED_COLOR_BLACK);
oled_putString(10, 10, "--------------", OLED_COLOR_WHITE,
        OLED_COLOR_BLACK);
oled_putString(30, 20, "GUESS", OLED_COLOR_WHITE,
        OLED_COLOR_BLACK);
oled_putString(30, 30, "THE", OLED_COLOR_WHITE,
        OLED_COLOR_BLACK);
oled_putString(30, 40, "NUMBER!", OLED_COLOR_WHITE,
        OLED_COLOR_BLACK);
```

…

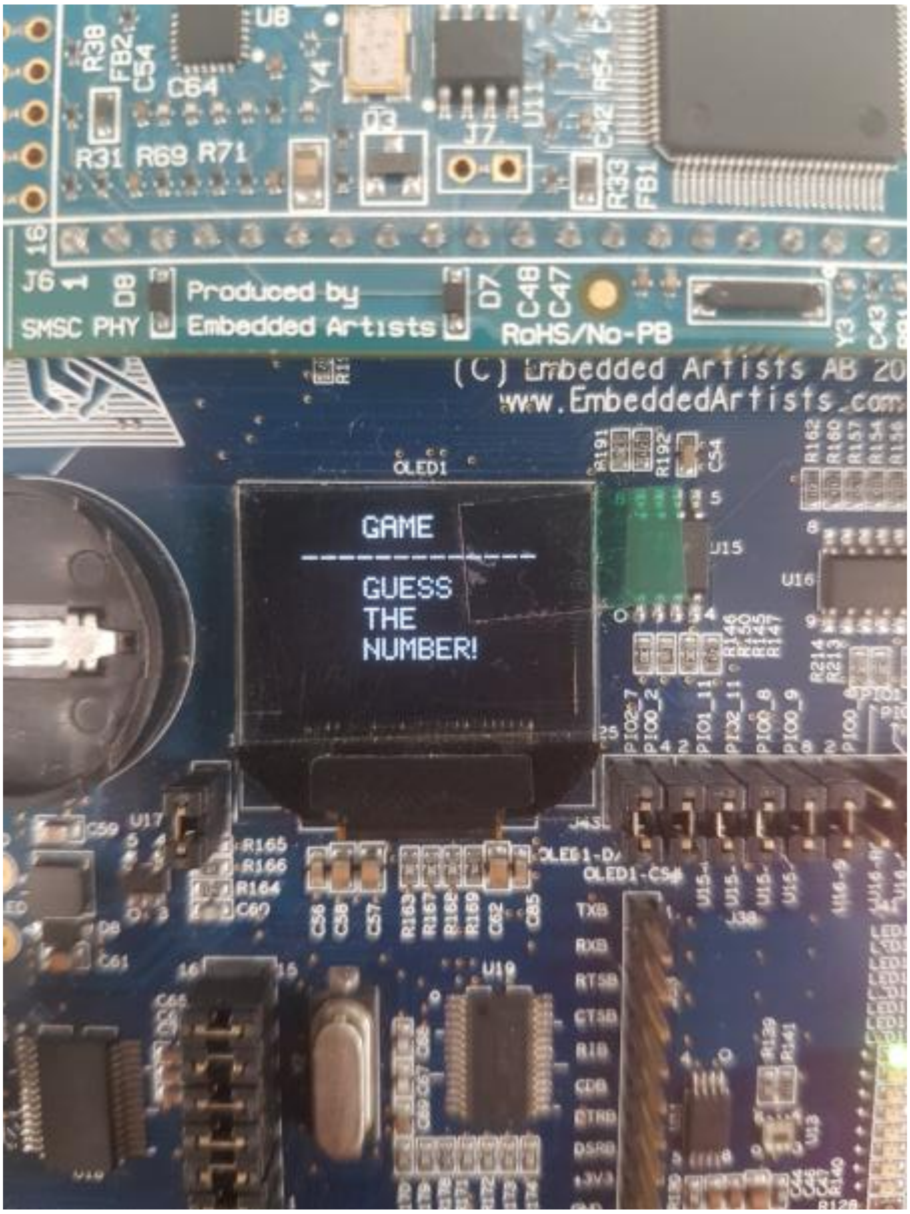


Figure 7.4.1: Guess the Number

```
sprintf(scoreOled, "Score: %d\r\n", score);
oled_putString(10, 10, scoreOled, OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
oled_putString(10, 20, "-------------", OLED_COLOR_WHITE,
            OLED_COLOR_BLACK);
oled_putString (20, 50, "YOU WIN!", OLED_COLOR_WHITE, OLED_COLOR_BLACK);
playSong(win);
```

…

Figure 7.4.2: You win

```
//if it is the highest score, then send UART
if(score < prevScore){
    prevScore = score;
    sprintf(scoreStr, "CONGRATULATIONS! Your Highest Score is now %d.\r\n",
prevScore);
    UART_Send(LPC_UART3, (uint8_t *) scoreStr, strlen(scoreStr),
            BLOCKING);
playSong(champ);
score = 0;
}
```

7.5: Update messages to CEMS on GAME mode

When the player has acheieved a new high score, a message will be sent to the UART Terminal.

```
sprintf(scoreStr, "CONGRATULATIONS! Your Highest Score is now %d.\r\n",
prevScore);
UART_Send(LPC_UART3, (uint8_t *) scoreStr, strlen(scoreStr),
      BLOCKING);
```

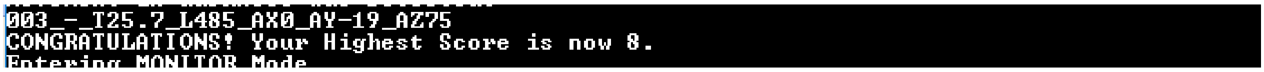


Figure 7.5: Screenshot of High Score on Tera Term

8: Using EINT3 Handler

In the EINT3 handler, you need to identify exactly which GPIO interrupt triggered your EINT3 handler. This is because multiple EINT3 interrupts can come from port 0 or port 2. Anyway, you identify the specific GPIO interrupt by checking the GPIO interrupt status register as shown below.

Interrupt lines to NVIC run only from the internal peripherals. Interrupts from external peripherals (such as the light sensor interrupts), if needed, will have to be brought in through GPIO or EINTx (x=0,1,2,3), and can only be used to trigger GPIO or EINTx interrupts.

For light sensor if an interrupt occurs, then it will set the flag isDark to be 1. Then clears the interrupt. Similarly, the rotary interrupts clear then sets the flag in the rotary function to be 1. We made sure our Interrupt Service Routines was as short as possible so that our main code cannot run while the control is in an interrupt handler.

```
//EINT3 Interrupt Handler, GPIO0 in NVIC with EINT3
void EINT3_IRQHandler(void) {
	//light sensor
	if ((LPC_GPIOINT ->IO2IntStatF >> 5) & 0x1) {
		isDark = 1;
		light_setLoThreshold(0);
		LPC_GPIOINT ->IO2IntClr |= (1 << 5); //clear GPIO
		light_getIrqStatus(); //Clears I2C
	}

	//rotary switch
	if (((LPC_GPIOINT->IO0IntStatF>>24)&0x1) | ((LPC_GPIOINT-
>IO0IntStatF>>25)&0x1)) {
		LPC_GPIOINT->IO0IntClr = 1<<24;
		LPC_GPIOINT->IO0IntClr = 1<<25;
		setRotaryDirection(1);
	}

}
```

9: Conclusion

In summary, CUTE uses two main modes- STABLE and MONITOR. STABLE mode is the mode CUTE enters upon being powered on while MONITOR mode is mode CUTE enters after MODE_TOGGLE is pressed during STABLE mode. During MONITOR mode, the SEGMENT_DISPLAY increments from 0 to F every second and resets after that. At every ‘5’, ‘A’ and ‘F’ displayed by the SEGMENT_DISPLAY, the GRAPHICS_DISPLAY updates itself with the sensor values at that instant. When temperature of the environment is more than 45°C, CUTE does BLINK_RED to indicate that fire is detected and sends a message to UART Terminal which updates itself with sensor values at every ‘F’ displayed by SEGMENT_DISPLAY. When the light intensity is less than 50 lux and movement is detected by the difference in magnitudes of more than 7, CUTE does BLINK_BLUE to indicate that movement in darkness is detected and sends a message to the UART Terminal. CUTE is extended to include a GAME mode to entertain elderly where they feel a sense of achievement upon winning the game and even achieving a new high score. The game is played by pressing SW3 at the instant when the number displayed by the SEGMENT_DISPLAY is equal to the number of LEDs lit. CUTE aims to accompany the elderly and entertain them in the absence of a caretaker.