Nathan Hanuscin
Github root directory: https://github.com/nhanuscin/HappyFunStuff

**Date Submitted:** **11/04/2019**

------------------------------------------------------------------------------------

## Task 01:

Youtube Link: https://www.youtube.com/watch?v=djN0R5nj5Vs

**Modified Schematic (if applicable): N/A**

**Modified Code:**

```
/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>

/* Driver Header files */
#include <ti/drivers/GPIO.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>

/* Board Header file */
#include "Board.h"

/*
 *  ======== mainThread ========
 */
void *mainThread(void *arg0)
{
    /* 1 second delay */
    uint32_t time = 1;

    /* Call driver init functions */
    GPIO_init();
    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();

    /* Configure the LED pin */
    GPIO_setConfig(Board_GPIO_LED0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);

    /* Turn on user LED */
    GPIO_write(Board_GPIO_LED0, Board_GPIO_LED_ON);

    while (1) {
        sleep(time);
        GPIO_toggle(Board_GPIO_LED0);
    }
}
```

---------------------------------------------------------------------------------

## Task 02:

Youtube Link: https://www.youtube.com/watch?v=qyd_K8_GodA

**Modified Schematic (if applicable): N/A**

**Modified Code:**

```
/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>

/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/drivers/ADC.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>

/* Board Header file */
#include "Board.h"

/*
 *  ======== mainThread ========
 */
void *mainThread(void *arg0)
{
    /* 1 second delay */
    uint32_t time = 1;
    uint16_t adcValue0 = 0;
    //uint32_t adcValue0MicroVolt;
    uint16_t threshold = 675;
    uint16_t trigger = 0;

    /* Call driver init functions */
    GPIO_init();
    ADC_init();
    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();
    ADC_Handle adc;
    ADC_Params params;
    ADC_Params_init(&params);
    adc = ADC_open(Board_ADC0, &params);
    if (adc == NULL) {
        //Display_printf(display, 0, 0, "Error initializing ADC channel 0\n");
        while (1);
    }

    /* Configure the LED pin */
    GPIO_setConfig(Board_GPIO_LED0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
    while (1)
    {
        int_fast16_t res;
        res = ADC_convert(adc, &adcValue0);
        if (res == ADC_STATUS_SUCCESS)
        {
            //Display_printf(displayHandle, 1, 0, "ADC Reading %d", adcValue0);
            if(adcValue0 >= threshold)
            {
                GPIO_write(Board_GPIO_LED0, Board_GPIO_LED_ON);
                trigger = 1;
            }
            else
            {
                GPIO_write(Board_GPIO_LED0, Board_GPIO_LED_OFF);
                trigger = 0;
            }
        }
    sleep(time);
    }
}
-----------------------------------------------------------------------------------
```

## Task 03:

Youtube Link: https://www.youtube.com/watch?v=SP10OqZu7YI

Modified Schematic (if applicable): N/A

Modified Code:

```
/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>

/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/drivers/ADC.h>
#include <ti/display/Display.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>

/* Board Header file */
#include "Board.h"

/*
 *  ======== mainThread ========
 */
void *mainThread(void *arg0)
```

```
{
    /* 1 second delay */
    uint32_t time = 1;
    uint16_t adcValue0 = 0;
    //uint32_t adcValue0MicroVolt;
    uint16_t threshold = 660;
    uint16_t trigger = 0;

    /* Call driver init functions */
    GPIO_init();
    ADC_init();
    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();
    Display_Handle displayHandle;
    Display_Params displayParams;
    Display_Params_init(&displayParams);
    displayHandle = Display_open(Display_Type_UART, NULL);

    ADC_Handle adc;
    ADC_Params params;
    ADC_Params_init(&params);
    adc = ADC_open(Board_ADC0, &params);
    if (adc == NULL) {
        Display_printf(displayHandle, 0, 0, "Error initializing ADC channel 0\n");
        while (1);
    }

    /* Configure the LED pin */
    GPIO_setConfig(Board_GPIO_LED0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);

    while (1)
    {
        int_fast16_t res;
        res = ADC_convert(adc, &adcValue0);
        if (res == ADC_STATUS_SUCCESS)
        {
            Display_printf(displayHandle, 1, 0, "ADC Reading %d", adcValue0);
            if(adcValue0 >= threshold)
            {
                GPIO_write(Board_GPIO_LED0, Board_GPIO_LED_ON);
                trigger = 1;
            }
            else
            {
                GPIO_write(Board_GPIO_LED0, Board_GPIO_LED_OFF);
                trigger = 0;
            }
        }
    sleep(time);
    }
}
-----------------------------------------------------------------------------------
```

**Grading scheme:** 30% Coding, 30% Documentation, 40% Execution/Video.

## Task 04:

Youtube Link: https://www.youtube.com/watch?v=OK4Dm4CJ6cY

**Modified Schematic (if applicable): N/A**

**Modified Code:**

```
/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>

/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/drivers/ADC.h>
#include <ti/display/Display.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>

/* Board Header file */
#include "Board.h"

uint16_t threshold = 0;
uint16_t trigger = 0;
uint16_t adcValue0 = 0;

void gpioButtonFxn0(uint_least8_t index)
{
    /* Clear the GPIO interrupt and decrement threshold */
    if(threshold < 250)
    {
        // Ensure threshold doesn't go below zero
        threshold = 0;
    }
    else
    {
        threshold -= 250; // decrement by 250
    }
}
void gpioButtonFxn1(uint_least8_t index)
{
    /* Clear the GPIO interrupt and increment threshold */
    if(threshold > 3845)
    {
        // Ensure threshold doesn't go above max ADC range
        threshold = 4095;
    }
    else
    {
        threshold += 250; // increment by 250
    }
}

/*
 *  ======== mainThread ========
 */
void *mainThread(void *arg0)
{
    /* 1 second delay */
    uint32_t time = 1;
    /* Call driver init functions */
    GPIO_init();

    ADC_init();

    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();
    Display_Handle displayHandle;
    Display_Params displayParams;
    Display_Params_init(&displayParams);
    displayHandle = Display_open(Display_Type_UART, NULL);

    ADC_Handle adc;
    ADC_Params params;
    ADC_Params_init(&params);
    adc = ADC_open(Board_ADC0, &params);
    if (adc == NULL) {
        Display_printf(displayHandle, 0, 0, "Error initializing ADC channel 0\n");
        while (1);
    }

    /* Configure the LED pin */
    GPIO_setConfig(Board_GPIO_LED0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
    /* install Button callback */
    GPIO_setCallback(Board_GPIO_BUTTON0, gpioButtonFxn0);
    GPIO_setCallback(Board_GPIO_BUTTON1, gpioButtonFxn1);
    /* Enable Interrupts */
    GPIO_enableInt(Board_GPIO_BUTTON0);
    GPIO_enableInt(Board_GPIO_BUTTON1);

    while (1)
    {
        int_fast16_t res;
        res = ADC_convert(adc, &adcValue0);
        if (res == ADC_STATUS_SUCCESS)
        {
            Display_printf(displayHandle, 1, 0, "ADC Reading %d", adcValue0);
            if(adcValue0 >= threshold)
            {
                GPIO_write(Board_GPIO_LED0, Board_GPIO_LED_ON);
                trigger = 1;
            }
            else
            {
                GPIO_write(Board_GPIO_LED0, Board_GPIO_LED_OFF);
                trigger = 0;
            }
        }
    Display_printf(displayHandle, 1, 0, "Threshold Value %d", threshold);
    sleep(time);
    }
}
-----------------------------------------------------------------------------------
```

**Grading scheme:** 30% Coding, 30% Documentation, 40% Execution/Video.