3-2 Milestone Two: Enhancement One: Software Design and Engineering

Victoria Franklin

Southern New Hampshire University

CS-499 Computer Science Capstone

Professor Gene Bryant

Sunday, July 21, 2024

CS 499 Milestone Two

I have updated the model of a thermostat control board, gpiointerrupt.c. This file was used in the CS-350 class's final project. This code snippet's broader project includes a timer, GPIO interrupt, UART, I2C sensor, and temperature data-based LED control. I chose this artifact because I enjoyed working on it, and CS-350 was a great class overall. Embedded system components demonstrating my programming abilities include GPIO, UART, I2C, and Timer capabilities. A sense of accomplishment in mastering the optimization of embedded systems' performance and reliability is displayed in effectively managing resources like GPIO pins, timers, and peripherals (GPIO_init, initTimer). For that reason, I have included the artifact in my ePortfolio.

The code has been improved and clarified to initialize all global variables explicitly. Minimize the impact of global variables by enclosing them in functions or limiting their scope to only the regions where they are needed. I2C and UART operations must have solid mechanisms for handling and recovering errors. Use UART for debugging output instead of printf and UART_write; it will assist in verifying mistakes. Using UART_write instead of printf for debug output regularly shows that you can communicate with embedded systems and ensure compatibility with real-time OSes like TI-RTOS. Include comments explaining any confusing logic and briefly describing your functions with their parameters, purpose, and return values when writing essential areas like interrupt handlers or sensor setups. The variable is implemented by modifying the logic for buttons (gpioButtonFxn0 and gpioButtonFxn1) according to the button's state. By checking the timing logic to ensure it satisfies the application requirements, we ensured there were no unnecessary delays or overlaps. All peripherals, including timers and GPIO pins, are freed whenever unused.

Improving and altering the artifact was not just instructive but also a journey of learning and growth. I realized the importance of solid error-handling methods while making software. An excellent way to avoid crashes and ambiguous behavior is to handle any possible error scenarios gracefully. Troubleshooting problems such as erroneous sensor readings or failed I2C transactions were complex. The software, the wiring, or the hardware could be the source of the problem; a systematic approach is necessary for diagnosis. Thorough preparation, including understanding the system's requirements and potential issues, was crucial in ensuring the project's success. Overall, improving and altering the artifact highlighted the significance of excellent software engineering standards, including modular architecture, correct error handling, and comprehensive testing. It highlighted the difficulties of dealing with hardware, the importance of thorough preparation, and the necessity of methodical debugging.

Category One - Software Design and Engineering

Original code

```
/*
 *  ======== gpiointerrupt.c ========
 */
#include <stdint.h>
#include <stddef.h>
#include <stdio.h>

// Timer Driver Header files
#include <ti/drivers/Timer.h>

// UART Driver Header files
#include <ti/drivers/UART.h>

// I2C Driver Header files
#include <ti/drivers/I2C.h>


/* Driver Header files */
#include <ti/drivers/GPIO.h>

/* Driver configuration */
#include "ti_drivers_config.h"

#define FALSE 0
#define TRUE 1
```

```
//flag to determine if button is pressed(1) or not (0)
int flag_button=0;
// enum to determine what button was pressed
enum button {left, right} button;
// count every time the period is over
int end_period_cnt = 0;
// temperature set point
int set_point = 23;
// convert period microseconds to millisecond
int period_in_milliseconds = 100;
// 1s in millisecond
int one_secound_in_millisecond = 1000;


// I2C Global Variables
static const struct {
uint8_t address;
uint8_t resultReg;
char *id;
} sensors[3] = {
{ 0x48, 0x0000, "11X" },
{ 0x49, 0x0000, "116" },
{ 0x41, 0x0001, "006" }
};
uint8_t txBuffer[1];
uint8_t rxBuffer[2];
I2C_Transaction i2cTransaction;
// Driver Handles - Global variables
I2C_Handle i2c;


// UART Global Variables
#define DISPLAY(x) UART_write(uart, &output, x);
char output[64];
int bytesToSend;
// Driver Handles - Global variables
UART_Handle uart;
// Make sure you call initUART() before calling this function.



// Driver Handles - Global variables
Timer_Handle timer0;   // Timer driver handle
volatile unsigned char TimerFlag = 0;


void initI2C(void)
{
    int8_t i, found;
    I2C_Params i2cParams;
    DISPLAY(snprintf(output, 64, "Initializing I2C Driver - "));
    // Init the driver
    I2C_init();
    // Configure the driver
    I2C_Params_init(&i2cParams);
    i2cParams.bitRate = I2C_400kHz;
    // Open the driver
    i2c = I2C_open(CONFIG_I2C_0, &i2cParams);
```

```
    if (i2c == NULL)
    {
        DISPLAY(snprintf(output, 64, "Failed\n\r"));
        while (1);
    }
    DISPLAY(snprintf(output, 32, "Passed\n\r"));
// Boards were shipped with different sensors.
// Welcome to the world of embedded systems.
// Try to determine which sensor we have.
// Scan through the possible sensor addresses
/* Common I2C transaction setup */
    i2cTransaction.writeBuf = txBuffer;
    i2cTransaction.writeCount = 1;
    i2cTransaction.readBuf = rxBuffer;
    i2cTransaction.readCount = 0;
    found = false;
    for (i=0; i<3; ++i)
    {
        i2cTransaction.slaveAddress = sensors[i].address;
        txBuffer[0] = sensors[i].resultReg;
        DISPLAY(snprintf(output, 64, "Is this %s? ", sensors[i].id));
        if (I2C_transfer(i2c, &i2cTransaction))
        {
            DISPLAY(snprintf(output, 64, "Found\n\r"));
            found = true;
            break;
        }
        DISPLAY(snprintf(output, 64, "No\n\r"));
    }
    if(found)
    {
        DISPLAY(snprintf(output, 64, "Detected TMP%s I2C address: %x\n\r", sensors[i].id,
i2cTransaction.slaveAddress));
    }
    else
    {
        DISPLAY(snprintf(output, 64, "Temperature sensor not found, contact
professor\n\r"));
    }
}
int16_t readTemp(void)
{
    int16_t temperature = 2;
    i2cTransaction.readCount = 2;
    if (I2C_transfer(i2c, &i2cTransaction))
    {
/*
* Use the TMP sensor datasheet to convert the obtained
* data to a temperature in degrees C
*/
        temperature = (rxBuffer[0] << 8) | (rxBuffer[1]);
        temperature *= 0.0078125;
/*
* For a 2's complement representation, the MSB must be 1
* a negative number requiring a negative sign extension to be used
*/
        if (rxBuffer[0] & 0x80)
        {
```

```
            temperature |= 0xF000;
        }
    }
    else
    {
        DISPLAY(snprintf(output, 64, "Error reading temperature
sensor(%d)\n\r",i2cTransaction.status))
        DISPLAY(snprintf(output, 64, "Please power cycle your board by unplugging USB and
plugging back in.\n\r"))

    }
    return temperature;
}

// UART Global Variables
char output[64];
int bytesToSend;
// Driver Handles - Global variables

void initUART(void)
{
    UART_Params uartParams;
// nit the driver
    UART_init();
// Configure the driver
    UART_Params_init(&uartParams);
    uartParams.writeDataMode = UART_DATA_BINARY;
    uartParams.readDataMode = UART_DATA_BINARY;
    uartParams.readReturnMode = UART_RETURN_FULL;
    uartParams.baudRate = 115200;
// Open the driver
    uart = UART_open(CONFIG_UART_0, &uartParams);
    if (uart == NULL) {
/* UART_open() failed */
        while (1);
    }
}


void timerCallback(Timer_Handle myHandle, int_fast16_t status)
{
    TimerFlag = 1;
}


void initTimer(void)
{
    Timer_Params params;
// Init the driver
    Timer_init();
// Configure the driver
    Timer_Params_init(&params);
    params.period = 100000;
    params.periodUnits = Timer_PERIOD_US;
    params.timerMode = Timer_CONTINUOUS_CALLBACK;
    params.timerCallback = timerCallback;
// Open the driver
    timer0 = Timer_open(CONFIG_TIMER_0, &params);
```

```
    if (timer0 == NULL) {
/* Failed to initialized timer */
        while (1) {}
    }
    if (Timer_start(timer0) == Timer_STATUS_ERROR) {
/* Failed to start timer */
        while (1) {}
    }
    }


/*
 *  ======== gpioButtonFxn0 ========
 *  Callback function for the GPIO interrupt on CONFIG_GPIO_BUTTON_0.
 *
 *  Note: GPIO interrupts are cleared prior to invoking callbacks.
 */
void gpioButtonFxn0(uint_least8_t index)
{
    if(flag_button==0){
        flag_button = 1;
        button = left;
    }
    else if (flag_button==1){
        flag_button = 0;
        button = left;
    }
}

/*
 *  ======== gpioButtonFxn1 ========
 *  Callback function for the GPIO interrupt on CONFIG_GPIO_BUTTON_1.
 *  This may not be used for all boards.
 *
 *  Note: GPIO interrupts are cleared prior to invoking callbacks.
 */
void gpioButtonFxn1(uint_least8_t index)
{
    if(flag_button==0){
        flag_button = 1;
        button = right;
    }
    else if(flag_button==1){
        flag_button = 0;
        button = right;
    }
}


void interpretation_check(){
    if(flag_button == 1){
        printf("right button was pushed.\n");
        if(button == left){
            set_point++;
        }
        else if (button == right){
            set_point--;
        }
```

```
    }
    else
        printf("no button was pushed.\n");

}

void temperature_check(){
    readTemp();
    printf("current temperature is %d\n",readTemp());
}

void LED_check_print(){
    int heat;
    if( readTemp()>=set_point){
        heat = 0;
        GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_OFF);
        DISPLAY(snprintf(output, 64, "<%02d, %02d, %d, %04d>\n\r", readTemp(), set_point,
heat, (period_in_milliseconds*end_period_cnt)/one_secound_in_millisecond));
        end_period_cnt=0;
    }
    else if (readTemp()<set_point){
        heat = 1;
        GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
        DISPLAY(snprintf(output, 64, "<%02d, %02d, %d, %04d>\n\r", readTemp(), set_point,
heat, (period_in_milliseconds*end_period_cnt)/one_secound_in_millisecond));
        end_period_cnt=0;
    }
}


/*
 *  ======== mainThread ========
 */
void *mainThread(void *arg0)
{
    /* Call driver init functions */
    GPIO_init();
    initUART();
    initI2C();
    initTimer();
    /* Configure the LED and button pins */
    GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
    GPIO_setConfig(CONFIG_GPIO_BUTTON_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
    /* Install Button callback */
    GPIO_setCallback(CONFIG_GPIO_BUTTON_0, gpioButtonFxn0);

    /* Enable interrupts */
    GPIO_enableInt(CONFIG_GPIO_BUTTON_0);

    /*
     *  If more than one input pin is available for your device, interrupts
     *  will be enabled on CONFIG_GPIO_BUTTON1.
     */
    if (CONFIG_GPIO_BUTTON_0 != CONFIG_GPIO_BUTTON_1) {
        /* Configure BUTTON1 pin */
        GPIO_setConfig(CONFIG_GPIO_BUTTON_1, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);

        /* Install Button callback */
```

```
        GPIO_setCallback(CONFIG_GPIO_BUTTON_1, gpioButtonFxn1);
        GPIO_enableInt(CONFIG_GPIO_BUTTON_1);
    }
    while(TRUE){
        if (TimerFlag==1){
            end_period_cnt++;
            if(end_period_cnt==5)
                temperature_check();
            if(end_period_cnt==10){
                interpretation_check();
                LED_check_print();
            }

if(end_period_cnt==2||end_period_cnt==4||end_period_cnt==6||end_period_cnt==8)
                interpretation_check();

        }
    }
    return (NULL);
}
```

Enhanced Code

```
/*
 *  ======== gpiointerruptenhancement.c ========
 */
#include <stdint.h>
#include <stddef.h>
#include <stdio.h>

#include <ti/drivers/Timer.h>
#include <ti/drivers/UART.h>
#include <ti/drivers/I2C.h>
#include <ti/drivers/GPIO.h>

#include "ti_drivers_config.h"

// Global variables
volatile int flag_button = 0;
enum { LEFT, RIGHT } button;
int end_period_cnt = 0;
int set_point = 23;
const int period_in_milliseconds = 100;
const int one_second_in_milliseconds = 1000;

// I2C Variables
static const struct {
    uint8_t address;
    uint8_t resultReg;
    char *id;
} sensors[3] = {
    { 0x48, 0x0000, "11X" },
    { 0x49, 0x0000, "116" },
    { 0x41, 0x0001, "006" }
```

```
};
uint8_t txBuffer[1];
uint8_t rxBuffer[2];
I2C_Transaction i2cTransaction;
I2C_Handle i2c;

// UART Variables
char uartOutput[64];
UART_Handle uart;

// Timer Variables
Timer_Handle timer0;
volatile unsigned char TimerFlag = 0;

// Function prototypes
void initI2C(void);
int16_t readTemp(void);
void initUART(void);
void timerCallback(Timer_Handle myHandle, int_fast16_t status);
void initTimer(void);
void gpioButtonFxn0(uint_least8_t index);
void gpioButtonFxn1(uint_least8_t index);
void interpretation_check(void);
void temperature_check(void);
void LED_check_print(void);

void initI2C(void) {
    int8_t i;
    bool found = false;
    I2C_Params i2cParams;

    UART_write(uart, "Initializing I2C Driver\n", sizeof("Initializing I2C Driver\n"));

    I2C_init();
    I2C_Params_init(&i2cParams);
    i2cParams.bitRate = I2C_400kHz;
    i2c = I2C_open(CONFIG_I2C_0, &i2cParams);

    if (i2c == NULL) {
        UART_write(uart, "Failed to open I2C driver\n", sizeof("Failed to open I2C
driver\n"));
        while (1);
    }

    for (i = 0; i < 3; ++i) {
        i2cTransaction.slaveAddress = sensors[i].address;
        txBuffer[0] = sensors[i].resultReg;
        sprintf(uartOutput, "Checking sensor %s...", sensors[i].id);
        UART_write(uart, uartOutput, strlen(uartOutput));

        if (I2C_transfer(i2c, &i2cTransaction)) {
            sprintf(uartOutput, "Sensor %s found\n", sensors[i].id);
            UART_write(uart, uartOutput, strlen(uartOutput));
            found = true;
            break;
        } else {
            UART_write(uart, "No\n", sizeof("No\n"));
        }
```

```
    }

    if (found) {
        sprintf(uartOutput, "Detected TMP%s I2C address: %x\n", sensors[i].id,
i2cTransaction.slaveAddress);
        UART_write(uart, uartOutput, strlen(uartOutput));
    } else {
        UART_write(uart, "Temperature sensor not found\n", sizeof("Temperature sensor not
found\n"));
    }
}

int16_t readTemp(void) {
    int16_t temperature = 0;

    i2cTransaction.readCount = 2;
    if (I2C_transfer(i2c, &i2cTransaction)) {
        temperature = (rxBuffer[0] << 8) | rxBuffer[1];
        temperature *= 0.0078125;

        if (rxBuffer[0] & 0x80) {
            temperature |= 0xF000;
        }
    } else {
        UART_write(uart, "Error reading temperature sensor\n", sizeof("Error reading
temperature sensor\n"));
        UART_write(uart, "Please power cycle your board\n", sizeof("Please power cycle
your board\n"));
    }

    return temperature;
}

void initUART(void) {
    UART_Params uartParams;

    UART_init();
    UART_Params_init(&uartParams);
    uartParams.writeDataMode = UART_DATA_BINARY;
    uartParams.readDataMode = UART_DATA_BINARY;
    uartParams.readReturnMode = UART_RETURN_FULL;
    uartParams.baudRate = 115200;

    uart = UART_open(CONFIG_UART_0, &uartParams);
    if (uart == NULL) {
        UART_write(uart, "Failed to open UART driver\n", sizeof("Failed to open UART
driver\n"));
        while (1);
    }
}

void timerCallback(Timer_Handle myHandle, int_fast16_t status) {
    TimerFlag = 1;
}

void initTimer(void) {
    Timer_Params params;
```

```
    Timer_init();
    Timer_Params_init(&params);
    params.period = 100000;
    params.periodUnits = Timer_PERIOD_US;
    params.timerMode = Timer_CONTINUOUS_CALLBACK;
    params.timerCallback = timerCallback;

    timer0 = Timer_open(CONFIG_TIMER_0, &params);
    if (timer0 == NULL) {
        UART_write(uart, "Failed to initialize timer\n", sizeof("Failed to initialize
timer\n"));
        while (1);
    }

    if (Timer_start(timer0) == Timer_STATUS_ERROR) {
        UART_write(uart, "Failed to start timer\n", sizeof("Failed to start timer\n"));
        while (1);
    }
}

void gpioButtonFxn0(uint_least8_t index) {
    if (flag_button == 0) {
        flag_button = 1;
        button = LEFT;
    } else {
        flag_button = 0;
        button = LEFT;
    }
}

void gpioButtonFxn1(uint_least8_t index) {
    if (flag_button == 0) {
        flag_button = 1;
        button = RIGHT;
    } else {
        flag_button = 0;
        button = RIGHT;
    }
}

void interpretation_check(void) {
    if (flag_button == 1) {
        sprintf(uartOutput, "Right button pressed\n");
        UART_write(uart, uartOutput, strlen(uartOutput));

        if (button == LEFT) {
            set_point++;
        } else if (button == RIGHT) {
            set_point--;
        }
    } else {
        sprintf(uartOutput, "No button pressed\n");
        UART_write(uart, uartOutput, strlen(uartOutput));
    }
}

void temperature_check(void) {
    int16_t temperature = readTemp();
```

```
    sprintf(uartOutput, "Current temperature: %d\n", temperature);
    UART_write(uart, uartOutput, strlen(uartOutput));
}

void LED_check_print(void) {
    int heat;

    int temperature = readTemp();
    if (temperature >= set_point) {
        heat = 0;
        GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_OFF);
    } else {
        heat = 1;
        GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
    }

    sprintf(uartOutput, "<%02d, %02d, %d, %04d>\n", temperature, set_point, heat,
(period_in_milliseconds * end_period_cnt) / one_second_in_milliseconds);
    UART_write(uart, uartOutput, strlen(uartOutput));

    end_period_cnt = 0;
}

void *mainThread(void *arg0) {
    GPIO_init();
    initUART();
    initI2C();
    initTimer();

    GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
    GPIO_setConfig(CONFIG_GPIO_BUTTON_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
    GPIO_setCallback(CONFIG_GPIO_BUTTON_0, gpioButtonFxn0);
    GPIO_enableInt(CONFIG_GPIO_BUTTON_0);

    if (CONFIG_GPIO_BUTTON_0 != CONFIG_GPIO_BUTTON_1) {
        GPIO_setConfig(CONFIG_GPIO_BUTTON_1, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
        GPIO_setCallback(CONFIG_GPIO_BUTTON_1, gpioButtonFxn1);
        GPIO_enableInt(CONFIG_GPIO_BUTTON_1);
}
    while(TRUE){
            if (TimerFlag==1){
                end_period_cnt++;
                if(end_period_cnt==5)
                    temperature_check();
                if(end_period_cnt==10){
                    interpretation_check();
                    LED_check_print();
                }

if(end_period_cnt==2||end_period_cnt==4||end_period_cnt==6||end_period_cnt==8)
                    interpretation_check();

            }TimerFlag = 0; // Reset TimerFla
        }
        return (NULL);
    }
```