Lab 8
Jacob Hillebrand
CEE-345 Microprocessor System Design
IR Sensor Lab

In this lab, we did some exploration in proximity sensing using an IR LED and IR Receiver. The IR components were wired up to the FreedomBoard as shown in Fig. 1, which allowed the LED to be powered while the FreedomBoard read input from the sensor. To read this signal, the FreedomBoard's onboard Analog to Digital Converter (ADC) was used.

To make this system work, the Board was programmed to do a few things. First, the input pin from the IR LED Receiver was set to GPIO and initialized as input. Next, input was continuously read from this input pin, and was then passed through the ADC to assign the analog input to a binary number between 0x0 and 0xffff. With this number, then a decision as to the board's output could be made.

In Part 1 of this lab, the converted value was simply used as a voltage reference. Using the base known voltage, a calculation was performed that converted the number into the voltage value that was being read from the circuit. This value could was then assigned to a variable, which could be tracked and viewed in the Keil Debugger.

In Part 2 of this lab, we actually used the ADC and sensor circuit to interpret the voltage change in the IR circuit as given by the IR Sensor to display a color on the FreedomBoard's LED. If the proximity was determined to be a particular range, the IR Receiver would give a certain voltage, the ADC would read and convert the analog voltage to a binary value, and the color corresponding to that value would be sent to the LED. The code for both this part and the first part are documented in the below figures.

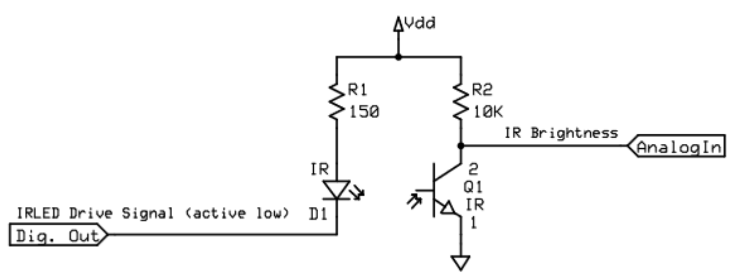

Figure 1: Wiring Diagram

```c
#include <MKL25Z4.H>
#include "gpio_defs.h"
#include "adc_defs.h"

/*  CEE-345 Microprocessor System Design
    Demonstration of simple ADC to measure DC Voltages
    Use ADC0_SE8, PTB0, J10, pin 2
    Use RGB LED on Freedom board */
/*
 * Simple and imprecise delay function
 */
 void Delay(unsigned int time_del) {
        // delay is about 1 millisecond * time_del
        volatile int t;

        while (time_del--) {
                for (t=4800; t>0; t--);
        }
}
/*
 * Cycle through the colors red, green and blue
 */

void redGreenBlue(void)
{
/* Each LED corresponds to a bit on a port
   Red LED connected to Port B (PTB), bit 18 (RED_LED_POS)
   Green LED connected to Port B (PTB), bit 19 (GREEN_LED_POS)
   Blue LED connected to Port D (PTD), bit 1 (BLUE_LED_POS)
   Active-Low outputs: Write a 0 to turn on an LED

   Turning LEDs on and off
   Turn on one LED: PTx->PDOR = ~ MASK(yyy_LED_POS) ;
   Turn on two LEDs: PTx->PDOR = ~ ( MASK(yyy_LED_POS) | MASK(zzz_LED_POS) ) ;
   Turn all LEDs off: PTx->PDOR = 0xFFFFFFFF ; */

           // set just red LED on
                 PTB->PDOR = ~ MASK(RED_LED_POS) ;
                 PTD->PDOR = 0xFFFFFFFF ;

                 // wait for 500ms
                 Delay(500) ;

                 // set just green on
                 PTB->PDOR = ~ MASK(GREEN_LED_POS) ;
                 PTD->PDOR = 0xFFFFFFFF ;

                 // wait for 500ms
                 Delay(500) ;

                 // set just blue on
                 PTB->PDOR = 0xFFFFFFFF ;
                 PTD->PDOR = ~ MASK(BLUE_LED_POS) ;

                 // wait for 500ms
                 Delay(500);
```

Figure 2: Part 1, Snippet 1

```c
/*
 *  Initialise on board LEDs.
 */
void Init_LED() {
        /* Configuration steps
             1. Enable pins as GPIO ports
             2. Set GPIO direction to output
             3. Ensure LEDs are off */

    // Make 3 pins GPIO
        PORTB->PCR[RED_LED_POS] &= ~PORT_PCR_MUX_MASK;
        PORTB->PCR[RED_LED_POS] |= PORT_PCR_MUX(1);
        PORTB->PCR[GREEN_LED_POS] &= ~PORT_PCR_MUX_MASK;
        PORTB->PCR[GREEN_LED_POS] |= PORT_PCR_MUX(1);
        PORTD->PCR[BLUE_LED_POS] &= ~PORT_PCR_MUX_MASK;
        PORTD->PCR[BLUE_LED_POS] |= PORT_PCR_MUX(1);

        // Set ports to outputs
        PTB->PDDR |= MASK(RED_LED_POS) | MASK(GREEN_LED_POS);
        PTD->PDDR |= MASK(BLUE_LED_POS);

    // Turn off LEDs
        PTB->PSOR = MASK(RED_LED_POS) | MASK(GREEN_LED_POS);
        PTD->PSOR = MASK(BLUE_LED_POS);
}

/*
 *  Initialise ADC
 */
void Init_ADC() {
        /* Set the ADC0_CFG1 to 0x9C, which is 1001 1100
           1 --> low power conversion
           00 --> ADIV is 1, no divide
           1 --> ADLSMP is long sample time
           11 --> MODE is 16 bit conversion
           00 --> ADIClK is bus clock / 2 */


        //enter a value to configure ADC channel 0 register CFG1
        ADC0->CFG1 = 0x9C;

        /* Set the ADC0_SC2 register to 0
           0 --> DATRG - s/w trigger
           0 --> ACFE - compare disable
           0 --> ACFGT - n/a when compare disabled
           0 --> ACREN - n/a when compare disabled
           0 --> DMAEN - DMA is disabled
           00 -> REFSEL - defaults V_REFH and V_REFL selected */

        ADC0->SC2 = 0;
}
```

Figure 3: Part 1, Snippet 2

```c
   unsigned res = 0 ;

        // Write to ADC0_SC1A
        //   0 --> AIEN Conversion interrupt diabled
        //   0 --> DIFF single end conversion
        //   01000 --> ADCH, selecting AD8

       ADC0->SC1[0] = ADC_CHANNEL ; // writing to this clears the COCO flag

        // test the conversion complete flag, which is 1 when completed
   while (!(ADC0->SC1[0] & ADC_SC1_COCO_MASK)); // empty loop

   // Read results from ADC0_RA as an unsigned integer
   res = ADC0->R[0] ; // reading this clears the COCO flag

   return res;
}


/*----------------------------------------------------------------------------
  MAIN function
 *----------------------------------------------------------------------------*/

// declare volatile to ensure changes seen in debugger
volatile float measured_voltage ;  // scaled value
volatile unsigned res;            // raw value

int main (void) {

       /* Configuration steps
          1. Enable clock to GPIO ports
          2. Enable clock to GPIO ports
          3. Set GPIO direction to output
          4. Ensure LEDs are off */

       // Enable clock to ports B and D
       SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTD_MASK | SIM_SCGC5_PORTE_MASK;

       // Enable clock to ADC
       SIM->SCGC6 |= (1UL << SIM_SCGC6_ADC0_SHIFT) ;

       // initialise LED
       Init_LED();

       // Initialise ADC
       Init_ADC();

  // end of configuration code
       while (1) {
    // This flashes the lights which is good to show it is working
               redGreenBlue();
       // measure the voltage
    res = Measure();
               // scale to an actual voltage, assuming VREF accurate
               //hints: measured voltage is equal to Vref multiplied by the ratio
               //of the resolution over the entire ADC range
    measured_voltage = VREF * res/ADCRANGE ;
       }
}
```

Figure 4: Part 1, Snippet 3

```c
#ifndef ADC_DEFS_H
#define ADC_DEFS_H


// Freedom KL25Z ADC Channel
//choose one of the the following values and fill in the value of your choice
//into an appropriate space in the parenthesis (8, 3.3, 0xffff)
#define ADC_CHANNEL (8) // on port B
#define VREF (3.3) // reference voltage
#define ADCRANGE (0xffff) // maximum for a 16 bit conversion

#endif
```

Figure 5: Part 1 adc_defs header file

```c
#ifndef GPIO_DEFS_H
#define GPIO_DEFS_H

#define MASK(x) (1UL << (x))

// Freedom KL25Z LEDs
#define RED_LED_POS (18)                // on port B
#define GREEN_LED_POS (19)      // on port B
#define BLUE_LED_POS (1)                // on port D

#endif
~
```

Figure 6: Part 1 gpio_defs header file

```c
#include <MKL25Z4.H>
#include "user_defs.h"
#include "LEDs.h"

//This program is to light RGB LED according to range of the IR LED.

//the Threshold parameters are used to sweep through valid IR Levels of the IR_LED
//IR_LED
int Threshold[NUM_RANGE_STEPS] = {1100, 650, 400, 270, 200, 0};

const int Colors[NUM_RANGE_STEPS][3] = {{ 1, 1, 1}, // white
                        { 1, 0, 1}, // magenta
                        { 1, 0, 0}, // red
                        { 1, 1, 0}, // yellow
                        { 0, 0, 1}, // blue
                        { 0, 1, 0}};// green


void Init_ADC(void) {
        SIM->SCGC6 |= (1UL << SIM_SCGC6_ADC0_SHIFT);
        ADC0->CFG1 =  0x9C; //ADC_CFG1_ADLPC_MASK | ADC_CFG1_ADIV(0) | ADC_CFG1_ADLSMP_MASK | ADC_CFG1_MODE
        ADC0->SC2 = 0;
}

//switch on and off IR_LED   ????????????????
void Control_IR_LED(unsigned int led_on) {
        if (led_on) {
                        PTB->PCOR =  MASK(IR_LED_POS);
        } else {
                        PTB->PSOR = MASK(IR_LED_POS);
        }
}

//initilize IR_LED by using the PCR register; identify the pin number
//on the PORTB to connect it to PORTB
void Init_IR_LED(void) {
        PORTB->PCR[IR_LED_POS] &= ~PORT_PCR_MUX_MASK;
        PORTB->PCR[IR_LED_POS] |= PORT_PCR_MUX(1);
        PTB->PDDR |= MASK(IR_LED_POS);

        // start off with IR LED turned off
        Control_IR_LED(0);
}

unsigned Measure_IR(void) {
        volatile unsigned res=0;

        ADC0->SC1[0] = IR_PHOTOTRANSISTOR_CHANNEL; // start conversion on channel 0

        while (!(ADC0->SC1[0] & ADC_SC1_COCO_MASK));
        res = ADC0->R[0];
        // complement result since voltage falls with increasing IR level
        // but we want result to rise with increasing IR level
        return 0xffff-res;
}
```

Figure 7: Part 2, Snippet 1

```c
void Display_Range(int b) {
	unsigned i;

	for (i=0; i<NUM_RANGE_STEPS-1; i++) {
		if (b > Threshold[i])
			break;
	}
	Control_RGB_LEDs(Colors[i][RED], Colors[i][GREEN], Colors[i][BLUE]);
}

void Delay_us(volatile unsigned int time_del) {
	// This is a very imprecise and fragile implementation!
	time_del = 9*time_del + time_del/2;
	while (time_del--) {
		;
	}
}
/*----------------------------------------------------------------------------
  MAIN function
 *----------------------------------------------------------------------------*/
int main (void) {
	unsigned on_brightness=0, off_brightness=0;
	static int avg_diff;
	int diff;
	unsigned n;

	Init_ADC();
	Init_RGB_LEDs();
	Init_IR_LED();
	Control_RGB_LEDs(0, 0, 0);

	while (1) {
		diff = 0;
		for (n=0; n<NUM_SAMPLES_TO_AVG; n++) {
			// measure IR level with IRLED off
			Control_IR_LED(0);
			Delay_us(T_DELAY);
			off_brightness = Measure_IR();

			// measure IR level with IRLED on
			Control_IR_LED(1);
			Delay_us(T_DELAY);
			on_brightness = Measure_IR();

			// calculate difference
			diff += on_brightness - off_brightness;
		}
		avg_diff = diff/NUM_SAMPLES_TO_AVG;

		// light RGB LED according to range
		Display_Range(avg_diff);

	}
}
```

Figure 8: Part 2, Snippet 2

```c
#include <MKL25Z4.H>
#include "user_defs.h"
#include "LEDs.h"

void Init_RGB_LEDs(void) {
	// Enable clock to ports B and D
	SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTD_MASK;;

	// Make 3 pins GPIO
	PORTB->PCR[RED_LED_POS] &= ~PORT_PCR_MUX_MASK;
	PORTB->PCR[RED_LED_POS] |= PORT_PCR_MUX(1);
	PORTB->PCR[GREEN_LED_POS] &= ~PORT_PCR_MUX_MASK;
	PORTB->PCR[GREEN_LED_POS] |= PORT_PCR_MUX(1);
	PORTD->PCR[BLUE_LED_POS] &= ~PORT_PCR_MUX_MASK;
	PORTD->PCR[BLUE_LED_POS] |= PORT_PCR_MUX(1);

	// Set ports to outputs
	PTB->PDDR |= MASK(RED_LED_POS) | MASK(GREEN_LED_POS);
	PTD->PDDR |= MASK(BLUE_LED_POS);
}

void Control_RGB_LEDs(unsigned int red_on, unsigned int green_on, unsigned int blue_on) {
	if (red_on) {
			PTB->PCOR = MASK(RED_LED_POS);
	} else {
			PTB->PSOR = MASK(RED_LED_POS);
	}
	if (green_on) {
			PTB->PCOR = MASK(GREEN_LED_POS);
	}	else {
			PTB->PSOR = MASK(GREEN_LED_POS);
	}
	if (blue_on) {
			PTD->PCOR = MASK(BLUE_LED_POS);
	}	else {
			PTD->PSOR = MASK(BLUE_LED_POS);
	}
}
```

Figure 9: Part 2, Snippet 3

```c
#ifndef LEDS_H
#define LEDS_H

// Freedom KL25Z LEDs
#define RED_LED_POS (18)		// on port B
#define GREEN_LED_POS (19)	// on port B
#define BLUE_LED_POS (1)		// on port D

// function prototypes
void Init_RGB_LEDs(void);
void Control_RGB_LEDs(unsigned int red_on, unsigned int green_on, unsigned int blue_on);

#endif
```

Figure 10: Part 2 LED header file

```c
#ifndef USER_DEFS_H
#define USER_DEFS_H

#define MASK(x) (1UL << (x))

// I/O pin assignments
#define IR_LED_POS (1) // on port B bit 1


#define IR_PHOTOTRANSISTOR_CHANNEL (8) // on port B bit 0

#define T_DELAY (5000)

#define NUM_RANGE_STEPS (6)

#define NUM_SAMPLES_TO_AVG (10)

#define RED (0)
#define GREEN (1)
#define BLUE (2)

#endif
```

Figure 11: Part 2 ADC_defs header file