Mitchell Larson
CE 3101
Lab 7
4/29/18

CE 3101 Lab Experiment 7

Abstract

Often in the computer electrons industry, components are controlled through the use of programmable microcontrollers acting as the brains of a system. The components being controlled by the microcontroller range from small to large voltage loads. While it may be possible to power small voltage DC loads using the GPIO pins for many microcontrollers, this is usually undesired behavior, since the pins were not designed for providing power, and leave the microcontroller at risk. Rather, than powering components using the GPIO pins, a better alternative would be powering the loads from external power sources and using the GPIO pins as a control mechanism. Several components exist for this functionality, such as the 2n7000 FET transistor. Using this component, the GPIO pins can be used to control the gate of the transistor, while isolating the board from damage resulting from high currents (since no current is supplied to the gate).

The purpose of this lab was to explore the use of this methodology, while completing the process of designing proper current limiting resistances for the load circuitry. To do this, Nucleo F446RE microcontrollers were used to interface with a temperature sensor, LCD panel, and keyboard to control an LED, a fan, and a sonic alert component. These 3 DC loads were controlled by using GPIO pins of the microcontroller, through a 2n7000 transistor. Additional detail regarding the application can be found in this document, along with the software and hardware design for the system.

Mitchell Larson
CE 3101
Lab 7
4/29/18

System Design

System design for this lab consisted of both software development and hardware development. To begin, software drivers from CE 2812 were imported for operation of the Nucleo F446RE microcontroller, and it's on board peripherals. The main component which would be utilized in this laboratory experiment were the temperature sensor and the LCD panel for user interfacing. After importing the drivers, a custom application was developed, which would drive 3 GPIO pins for controlling the gate pins of the MOSFETs used in the hardware circuitry.

The application that interfaced with the hardware was built for the following purpose. When the system was booted, an initial temperature would be read and used as the power-on temperature for the system. The system would then continuously monitor temperature and when the temperature being read surpassed the power-on temperature by 5 degrees Fahrenheit, the software would drive the 3 GPIO pins to logic-1. Connected to these 3 GPIO pins were the gate pins for 3 MOSFETS. These 3 MOSFETs controlled operation of an LED, a fan, and a sonic alert device. When the gate pins were driven to 1, it would cause the hardware to turn on. The devices would then stay on until the software detected that the temperature had dropped down to its original power-on temperature, causing the GPIO pins to be driven to logic-0, turning off the devices. Further detail on the control software and the circuit can be found below.

Software Design

As previously discussed, the software used in this laboratory experiment was mainly a port of a code base used in a previous class. From this, drivers used for the ADC, GPIO pins, LCD, and more were used. They will not be included in this report, as they are not essential to the purpose of this laboratory experiment. If needed, they can be accessed by requesting viewing rights to the private github repository by emailing larsonma@msoe.edu.

The application code, which is included in this report, essentially is a state machine which listens for both user input, and polls temperature data, and makes decisions based on them. On startup, the software will retrieve the temperature, and store it as the power-on temperature, which is used as a reference for the remaining lifecycle of the program. The software will then cycle through states, retrieving temperature data, retrieving user input, and displaying information to the user. When retrieving temperature data, an offset is added to the temperature, which is controlled by the user. This allows for the user to artificially adjust the temperature for development and experiment purposes. This adjusted temperature is then compared against the power-on temperature. If the temperature is more than 5 degrees above power on temperature, then the GPIO pins are driven to logic-1. If the temperature is at or below the power-on temperature, then the pins are driven to logic-0. The GPIO pins used were the pins controlling the on-board LEDS (PA 7, 8, and 9), which are also exposed for interfacing with external controls.

When reading user input, the software would check if any of the 'D', 'A', or 'B' keys were pressed on the keypad. If a 'D' was pressed, a help screen would be displayed to the user for 2 seconds, demonstrating how to use the offset feature. If an ‘A’ was pressed, the offset would be increased by 1, and if ‘B’ was pressed, the offset would be decremented by 1. Worth noting is that the keypad was not interrupt driven, but rather polled. That meant that the user would need to hold the key down for at most a quarter of a second for the input to be read.

The actual software implementation is as follows:

```
/*
 * main.c
 *
 *  Created on: April 29, 2018
 *      Author: Mitchell Larson
 */
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

#include "ADC.h"
#include "keypad.h"
#include "lcd.h"
#include "timer.h"
#include "gpio.h"

const char *help                = " D-hlp";
const char *current_temp_msg    = "Temp: ";
const char *power_on_temp_msg   = "On Temp: ";
const char *offset_up_msg       = "Offset Up: A";
const char *offset_down_msg     = "Offset Down: B";

typedef enum {INIT, READ, RETRIEVE, DISPLAY} State;
typedef enum {CURRENT, HELP} Mode1;

static void initalize();
static void read_input(Mode1 *mode, int *offset);
static void print_current_temp(float current_temp, float power_on_temp, int offset);
static void print_help();

/**
 * The main method of the file contains the control flow structure for a program
 * that monitors the current temperature of the external world. On startup initial
 * temperature is recorded, and if the room temperature ever exceeds +5 degrees,
 * the output pins are set to logic-1, triggering external alarms connected to them.
 * The alarm stops when temperature reaches original turn-on temperature.
 * Inputs:
 *              none
 * Outputs:
 *              none
 */
int main(void){
        //State variables
        State state = INIT;
        Mode1 mode = CURRENT;

        //temperature data
        float current_temp;
        float power_on_temp;
        bool alarmActive = false;
        int offset = 0;

        while(1){
                //state machine
```

```
		switch(state){
			case INIT:
				initalize();
				power_on_temp = get_tempF();
				state = READ;
				break;
			case READ:
				read_input(&mode, &offset);	//pass pointers so copies
				state = RETRIEVE;				//are not passed
				break;
			case RETRIEVE:
				//retrieve temp and input voltage adjusting extremes if needed
				current_temp = get_tempF() + offset;
				if(current_temp >= power_on_temp +5 && alarmActive == false){
					//set MOSFET gate pins to 1
					*(GPIOA_ODR)  |= (0x380);
					alarmActive = true;
				}else if(alarmActive && current_temp <= power_on_temp){
					//Turn MOSFETs off
					*(GPIOA_ODR)  &= ~(0x380);
					alarmActive = false;
				}
				state = DISPLAY;
				break;
			case DISPLAY:
				switch(mode){
					case CURRENT:
						print_current_temp(current_temp, power_on_temp, offset);
						break;
					case HELP:
						print_help();
						delay_ms(2000);
						mode = CURRENT;
				}

				state = READ;
				delay_ms(250);	//wait 250 seconds until the sequence repeats
				break;
		}
	}
	return 0;
}

/**
 * This function will initialize the Analog to digital converter, the keypad,
 * the LCD, and the output pins driving the MOSFET gate terminals.
 * Inputs:
 *		none
 * Outputs:
 *		none
 */
static void initalize(){
	ADC_init();
	key_init();
	lcd_init(C_OFF);

	//set gpio pins for controlling MOSFETs to output mode
	for(int i = 7; i <= 9; i++){
		set_pin_mode('A',i,OUTPUT);
	}

	//initialize pins to 0
	*(GPIOA_ODR)  &= ~(0x380);
}

/**
```

```c
 * This function will read any input from the key pad, if any, and change one of
 *  the state variable based on the user input.
 * Inputs:
 *              *mode - pointer to the state variable mode
 *              *offset - pointer to the temperature offset
 * Outputs:
 *              none, but the state variables may change upon running this function
 */
static void read_input(Mode1 *mode, int *offset){
        switch(key_getchar_noblock()){
                case 'A':
                        *offset = *offset+1;
                        break;
                case 'B':
                        *offset = *offset -1;
                        break;
                case 'D':
                        *mode = HELP;
                        break;
                default:
                        break;
        }
}

/**
 * This helper function prints the current temperature to the LCD, along with
 * the power-on temperature. The temperature offset and help option are displayed
 * as well to provide additional information to the user.
 * Inputs:
 *              current_temp - temperature to display
 *              tempMode - the temperature mode the program is in
 * Outputs:
 *              none
 */
static void print_current_temp(float current_temp, float power_on_temp, int offset){
        lcd_reset();
        char buffer1[5];
        sprintf(buffer1,"%4.1f",current_temp);

        char buffer2[5];
        sprintf(buffer2,"%4.1f", power_on_temp);

        char buffer3[3];
        itoa(offset, buffer3, 10);

        lcd_print_string(current_temp_msg);
        lcd_print_string(buffer1);
        lcd_print_string(help);
        lcd_row1();
        lcd_print_string(power_on_temp_msg);
        lcd_print_string(buffer2);
        lcd_print_string(" ");
        lcd_print_string(buffer3);
}

/**
 * This helper function prints the help menu to the user.
 * Inputs:
 *              none
 * Outputs:
 *              none
 */
static void print_help(){
        lcd_reset();
        lcd_print_string(offset_up_msg);
        lcd_row1();
        lcd_print_string(offset_down_msg);
}
```

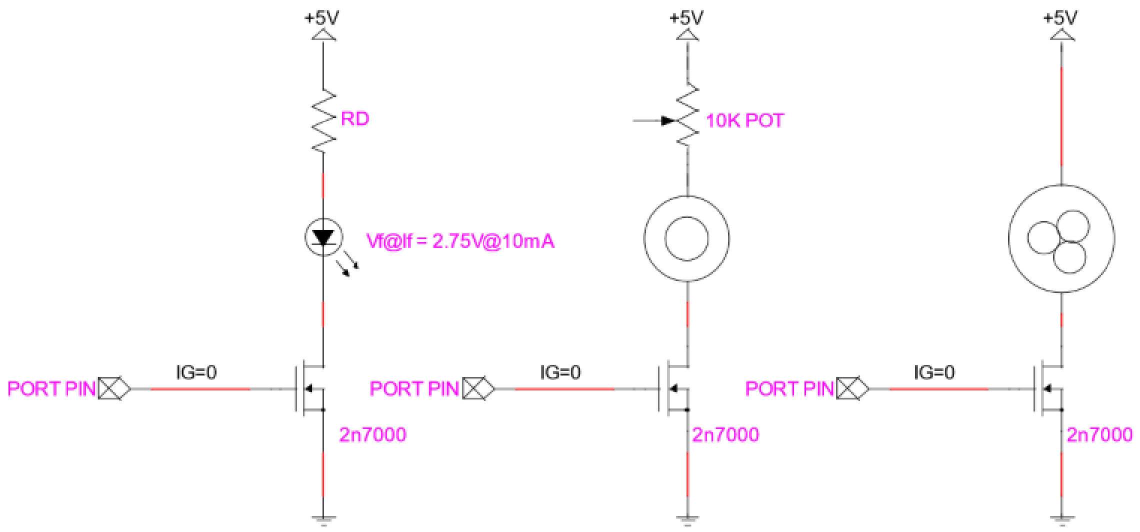
Hardware Design

The circuit being implemented was relatively simple. Essentially, 3 circuits with small DC loads would be switched on and off using the GPIO pins from the microcontroller. The 3 components being used as DC loads were a 2.75V, 10mA LED, a 5V ADDA 54F6 fan, and an 85dB son-alert siren. These DC loads were being controlled with 2n7000 transistors. The fan was self-regulating, so not much additional design was necessary for designing that portion of the circuit. The LED however, did not have self-regulating circuitry, so a current limiting resistor was needed to prevent the LED from burning out due to excessive current. Utilizing Kirchoff's voltage law, a V_{DD} of 5V, a LED voltage drop of 2.75V, and a current of 10 mA, the current limiting resistor was determined to be

$$R_D = \frac{(5V - 2.75V)}{10mA} = 225\Omega$$

Rounding up to the next closest available resistor meant that a 330Ω resistor would be used in the design. This would result in a slightly less bright LED, but would ensure the LED was still within safe operating range. With a power consumption of 23 mW (2.75V * 2.75V / 330 Ω), it would be safe to use a 330 Ω, 1/8 W resistor.

Since control of the sonic alert volume was desired, a 10KΩ potentiometer was used rather than a fixed resistor. Again, since the fan was self-regulating, no current limiting resistor was needed. This resulted in the following circuit.

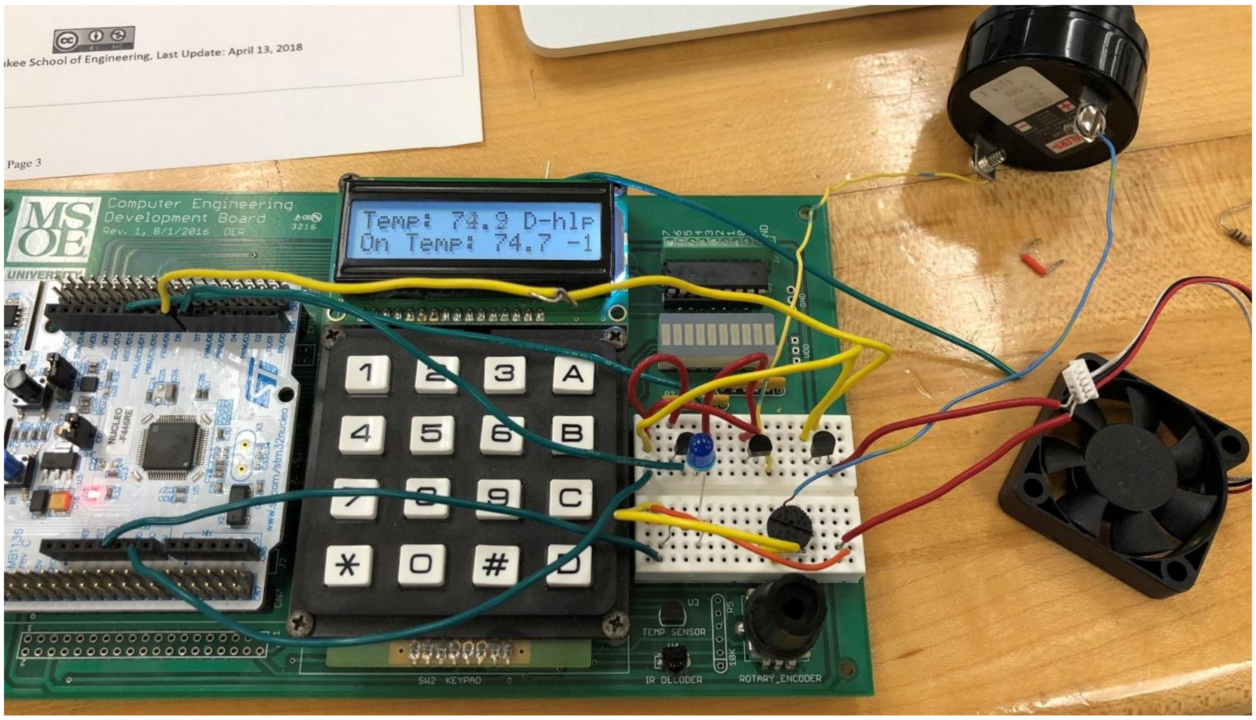


Credit - Dr. Meier, Milwaukee School of Engineering

Where the 2n7000 gate pins were hooked up to PA7, PA8, and PA9 of the microcontroller, respectively.

Mitchell Larson
CE 3101
Lab 7
4/29/18

The following image show the completion of the circuit implemented on a breadboard.



Demo

A demo of the experiment can be found at this link
https://msoe.box.com/s/ze5dxvufme2sams32sb8u06komins4p5

Conclusion

The purpose of this lab was to explore and design a system where DC loads were controller with transistors switched using a microcontroller. This was a design methodology which had never been tested in previous courses. Such a design method is vital for systems interfacing with external DC loads that needed to be provided with power that is larger than the microcontroller pins can provide. Previous design labs all made use on board components, with little risk of controller damage. This laboratory experiment provided essential experience which allows for much greater design diversity options for future projects and laboratory experiments. Due to the powerful material reinforced by this experiment, it is probably one of the most useful experiments that have been performed thus far in the CE track. Because of this, and the successful nature of the implementation, it would be appropriate to conclude this experiment as a success.