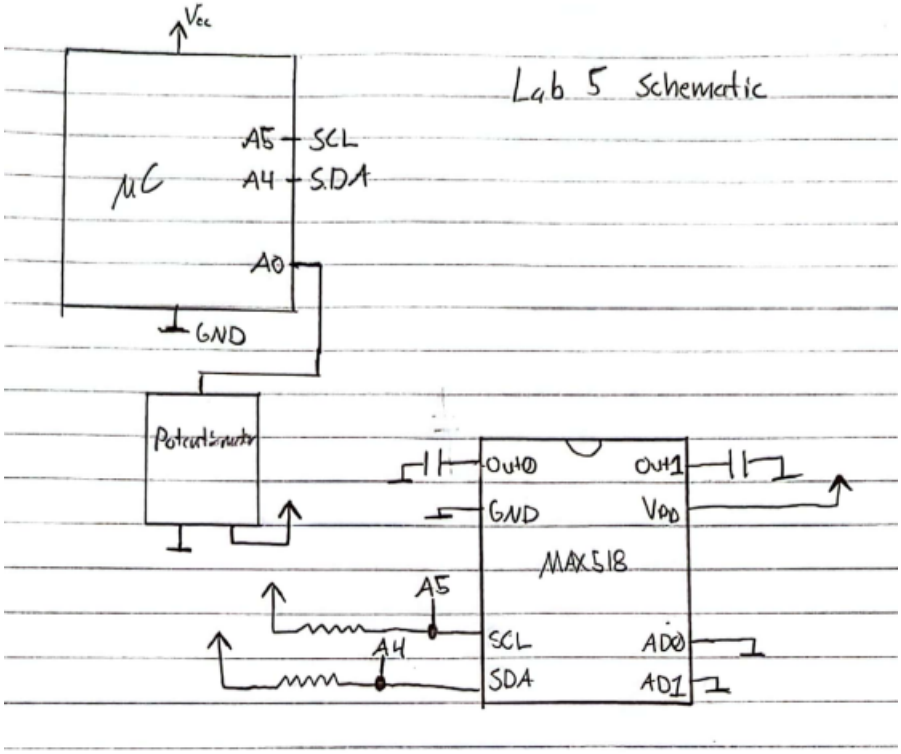
Authors: Ian Kuk, Brandon Cano
Professor Beichel
ECE:3360 Embedded Systems
Post-Lab 5 Report

1. Introduction

The goal of this lab was to create a microcontroller based circuit using a serial interface protocol, I2C, and be able to receive and transmit data from the microcontroller using C. We also had to use ADC and DAC voltage conversions for the different scenarios. We also had to be able to read in inputs from the serial monitor and be able to have the whole line be parsed and used for each respective command.

2. Schematic

This image shows a simplified view of how we implemented our circuit. We used a MAX518 chip as well as a potentiometer, both were connected to the microcontroller. The resistors connected to the SCL and SDA ports on the MAX518 both have a value of 4.7 kΩ. The decoupling capacitors connected to the Out0 and Out1 ports both have a value of 0.1 nF.



3. Discussion

For this lab we split it up into two parts, the hardware and software. For the hardware setup, we had to use a MAX518 chip and a potentiometer. For the potentiometer, all we needed to do was wire the side with 2 pins to power and ground, then the third pin on the other side was connected into the microcontroller at pin A0, which is an analog input. This allowed us to change the ADC voltage within the 0V - 5V range.

For the MAX518 chip we first wired up the Out0 and Out1 pins to ground with a 1nF decoupling capacitor, these are used to help monitor the DAC voltage that we set to either channel 0 or 1 respectively. Then we added the GND and Vdd connection to pins 2 and 7 respectively. For both pin 3 (SCL) and pin 4 (SDA) we had to add a 4.7 kΩ pull up resistor connected to Vcc. SCL had a wire connected to pin A5 on the microcontroller, this is because on the datasheet this is where the pin has the SCL functionality to it. SDA had a wire connected to pin A4 on the microcontroller because the datasheet showed us where the SDA functionality was located. Finally, we set the AD0 and AD1 outputs to ground since they aren't being used in this lab.

For the software section of this lab we used the C programming language. For this, we had to use a few different libraries in order to make the task more manageable for us, most of which are built into the software already. The one exception was the I2C serial communication, from the github page, https://github.com/alx741/avr_i2c, we downloaded the 'i2cmaster.h' and 'twimaster.c' files and added them in the same directory as our 'main.c' file in order to use their functionality.

At the start of the program we first enable the standard C library functions 'printf' and 'scanf' so that we can read and display information from and to the user. Then we have to set up the ADC channel so that we can read the ADC voltage when needed. Then we initialized some variables that will be used for storing the values of the user inputs. Inside the main loop, we constantly check for a user input. Once we receive a command, we first check from the buffer which command was entered, either G, S, or W.

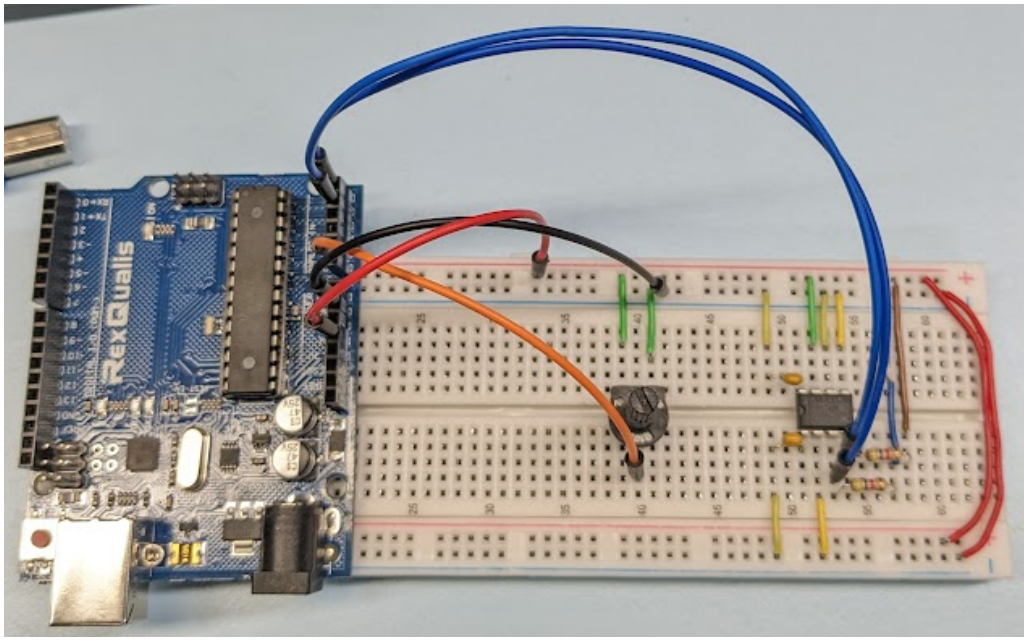
If 'G' was received, then all we do is read in the ADC voltage with the 'read_adc(pin)' function. If 'S' was received, then we check to make sure that the DAC channel and voltage value was within their acceptable values, if not a message will explain the error and do nothing, otherwise if all the values are good, then we set the DAC voltage to whatever channel was specified using 'set_dac_voltage(channel, voltage)'. If 'W' was received then we check if the channel, wave frequency, and number of cycles all entered correctly, if not then a message will display informing the user and nothing will happen. Otherwise we generate a sine wave, which can be seen on an oscilloscope. If neither G, S, or W was entered first then the default case of the switch statement is called informing that it was an invalid command.

In order to read the ADC voltage, we call our function 'read_adc(*uint8_t pin*)' which takes the pin number as an input, for us it is 0. We first read in the value from the pin, then have to convert it to an 8-bit value, and we wait while it converts before doing anything. Then we have to convert that 8-bit value to a readable voltage. To do this, we have to divide by 1023, the reason for this is that since it's a 10-bit number that means we get a prescaler of 2^10. Then we multiply by 5 since the voltage range is from 0V - 5V.

To set the DAC voltage to either channel 0 or 1 we call our function 'set_dac_voltage(*uint8_t channel*, float *voltage*)'. This is where the I2C serial communication starts being utilized in this lab. From the library we call 'i2c_start()' to enable write mode the the DAC_ADDRESS that we defined as 0b01011000 up at the top. We then call 'i2c_write()' and add the channel number that we will read the output from (0 -> Out0 and 1 -> Out1 on the MAX518). Then we convert the floating point voltage value to an 8-bit value by multiplying it by 51 and write that to the DAC as well, then we stop the communication since we are now done with our task. The results can be viewed and verified on a multimeter.

In order for us to generate the sine wave, we use our function 'create_sine_wave(*uint8_t channel*, int *frequency*, int *cycles*)'. First we start off with some calculations, including the period from the frequency. Then we find the timing delay based on the frequency to get a rough estimate for either 10 or 20 Hz. We then defined an array of the different sine values in the 8-bit format to make displaying a sine wave easier. To show the wave we have a for loop that will run for the amount of cycles that were given,

then another one nested that runs through all 64 items in the ‘sine_values’ array. Inside of this we start our connection to the DAC_ADDRESS, write the channel, then sine value, then close the connection with a brief delay calculated above and repeat for the amount of cycles. To view and verify this sine wave, we have to connect an oscilloscope to the circuit and when the command is run, we can see the waveform generated.



4. Conclusion

From this lab, we both learned how to use C to program a microcontroller along with being able to use I2C for serial communication between the circuit and the monitor and being able to control and read the ADC and DAC voltages.

5. Appendix A: Source Code

```
/*
 * main.c
 *
 * Created: 4/10/2023 11:48:56 AM
 *  Author: Brandon Cano, Ian Kuk
 */
#define F_CPU 16000000L
#define UART_BAUD 9600
#define DAC_ADDRESS 0b01011000
#define slave 0101100
#define __DELAY_BACKWARD_COMPATIBLE__

#include <ctype.h>
#include <stdint.h>
#include <stdio.h>

#include <avr/io.h>
#include <util/delay.h>
#include <util/twi.h>

// files 'twimaster.c' and 'i2cmaster.h' came from: https://github.com/alx741/avr_i2c
#include "twimaster.c"
```

```
int uart_putchar(char, FILE*);
int uart_getchar(FILE*);

// setup file stream to use with printf/scanf
static FILE uart_io = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);

static float read_adc(uint8_t pin) {
	ADMUX = (ADMUX & 0xf0) | pin;
	ADCSRA |= 1 << ADSC;		// conversion to 8 bit

	while (!(ADCSRA & (1 << ADIF))); // wait for conversion to complete

	ADCSRA |= 1 << ADIF;

	return 5.0f * (ADCL + (ADCH << 8)) / 1023.0f;
}

void set_dac_voltage(uint8_t channel, float voltage) {
	i2c_start(DAC_ADDRESS | I2C_WRITE);  // set device address and write mode
	i2c_write(channel);                  // write to DAC channel, either 0 or 1
	uint8_t byte_val = 51*voltage;       // convert from float to 8bit
	i2c_write(byte_val);                 // write DAC voltage
	i2c_stop();
}

void create_sine_wave(uint8_t channel, int frequency, int cycles) {
	double period = 1.0f / frequency;     // gets the period of the wave

	double t;                             // will give the proper delay for a displayable frequency
	if (frequency == 10) {                // determine which frequency to write
		t = ((period*800.0f) / 64.0f);  // f=10
	} else {
		t = ((period*600.0f) / 64.0f);  // f=20
	}

	// sine wave values
	uint8_t sine_values[64] = {
		128, 141, 153, 165, 177, 188, 199, 209, 219, 227, 234, 241, 246, 250, 254, 255,
		255, 255, 254, 250, 246, 241, 234, 227, 219, 209, 199, 188, 177, 165, 153, 141,
		128, 115, 103,  91,  79,  68,  57,  47,  37,  29,  22,  15,  10,   6,   2,   1,
		  0,   1,   2,   6,  10,  15,  22,  29,  37,  47,  57,  68,  79,  91, 103, 115
	};

	for (int i=0; i<cycles; i++) {                // number of cycles
		for (int j=0; j<64; j++) {            // amount of values for sine wave
			i2c_start(DAC_ADDRESS | I2C_WRITE);
			i2c_write(channel);         // write to channel
			i2c_write(sine_values[j]);  // write voltage value in wave
			i2c_stop();
			_delay_ms(t);               // add delay
		}
	}
}


int main() {
	// enables printf and scanf
	stdout = stdin = &uart_io;

	i2c_init();

	// ADC setup
	ADMUX |= 0b01 << REFS0;
	ADCSRA = 1 << ADEN | 0b110 << ADPS0;

	UCSR0A = 1 << U2X0;
	UBRR0L = (F_CPU / (8UL * UART_BAUD)) - 1;
```

```
          UCSR0B = 1 << TXEN0 | 1 << RXEN0;

          // initialize variables for inputs
          float voltage;
          char buf[32];

          uint8_t channelInput[1];
          uint8_t sineWaveInputs[2];

          for (;;) {
            printf("> ");
            if (fgets(buf, sizeof buf - 1, stdin) == NULL)   // grab user input
              return;

            switch (buf[0]) {                               // check for command (G,S,W)
            case 'G':                              // reads the ADC voltage
              printf("ADC Voltage = %0.3f V\n", read_adc(0));
              break;
            case 'S':
              if (sscanf(buf, "%*c,%hhd,%f", &channelInput[0], &voltage) > 0) {
                if(channelInput[0] != 0 && channelInput[0] != 1) {
                    printf("Invalid DAC channel %d", channelInput[0]);
               }
               else if (voltage > 5 || voltage < 0) {
                    printf("Voltage out of range %0.2f", voltage);
               }
               else {
                    printf("DAC Channel %hhd set to %0.2f V (177d)\n", channelInput[0], voltage);
                    set_dac_voltage(channelInput[0], voltage);
               }
          }
          break;
          case 'W':
            if (sscanf(buf, "%*c,%hhd,%hhd,%hhd", &channelInput[0], &sineWaveInputs[0], &sineWaveInputs[1]) > 0) {
              if (channelInput[0] != 0 && channelInput[0] != 1) {
                    printf("Invalid DAC channel %d", channelInput[0]);
               }
               else if (sineWaveInputs[0] != 10 && sineWaveInputs[0] != 20) {
                    printf("Invalid sine wave frequency %d", sineWaveInputs[0]);
               }
               else if (sineWaveInputs[1] < 1 || sineWaveInputs[1] > 100) {
                    printf("Invalid number of waveform cycles %d", sineWaveInputs[1]);
               }
               else {
                   printf("Generating %hhd sine wave cycles with f=%hhd on DAC channel %hhd\n", sineWaveInputs[1],
sineWaveInputs[0], channelInput[0]);
                   create_sine_wave(channelInput[0], sineWaveInputs[0], sineWaveInputs[1]);
               }
          }
          break;
          default:
            printf("Invalid Command.");
            break;
          }

  putchar('\n');
  }
}

int uart_putchar(char c, FILE *s) {
          // CRLF insertion
          if (c == '\n') {
                    uart_putchar('\r', s);
          }

          while (!(UCSR0A & (1 << UDRE0)));
          UDR0 = c;
```

```
        return 0;
}

int uart_getchar(FILE *s) {
        // wait for data to be received
        while (!(UCSR0A & (1 << RXC0)));
        uint8_t c = UDR0;

        return c;
}
```