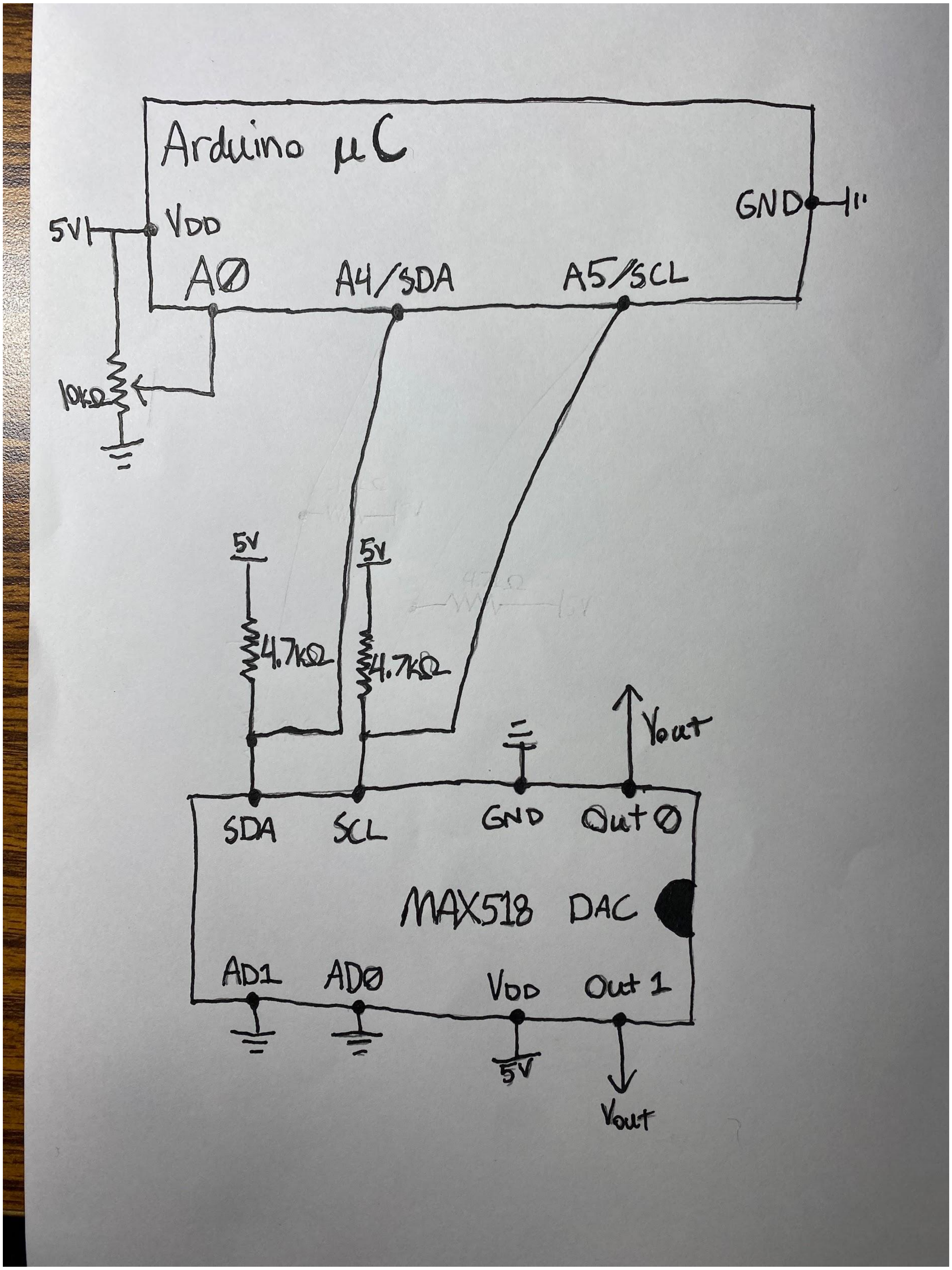
Authors: Max Finch, Tiger Slowinski
Team Members: Max Finch, Tiger Slowinski
ECE:3360 Embedded Systems
Post-Lab Report 5

1. Introduction

The objective is to create a system capable of measuring and generating analog signals that can be controlled remotely. This will be achieved by utilizing the internal A/D converter of the ATmega328P controller and an external two-channel D/A converter known as MAX518, which has an I2C interface. The system will be equipped with an RS232 interface (9600 8N1) to connect to a laptop or computer. Users can trigger analog voltage measurements and set the output voltage sequence for both DAC channels by sending commands through the RS232 interface.

Command	Function	Arguments
G	Get single voltage measurement from ADC	no arguments The ATmega328P ADC must be used in 10-bit mode!
S,c,v	Set DAC output voltage	c … DAC channel number (integer, c ∊ {0,1}) v ... output voltage (float, format: "n.nn" V, needs to be converted into a decimal number which is sent to the DAC such that the quantization error is minimal)
W,c,f,r	Generate a sine wave on DAC output	c ... DAC channel number (integer, c ∊ {0,1}) f ... frequency of waveform (integer, f ∊ {10,20} Hz), the signal frequency must be within +/- 5% r ... number of consecutive waveform cycles to generate (integer, 1≤ r ≤ 100) A lookup table for generating the sin wave can be found on ICON (file: “sin_table.csv”)

2. Schematic


Arduino µC
5V
VDD
GND
AØ
A4/SDA
A5/SCL
10kΩ
5V
5V
4.7kΩ
4.7kΩ
Vout
SDA
SCL
GND
Out Ø
MAX518 DAC
AD1
ADØ
VDD
Out 1
5V
Vout

3. Discussion

Hardware Explanation

ADC

Analog digital conversion (ADC) is to take in a variable analog voltage, and then creating a digital representation of that voltage digitally using a set number of bits, or the bit resolution. In the case of the ATMega328P, the 6 analog pins have a resolution of 10 bits (0 to 1023). By setting our reference voltage (VREF) to our VCC of 5V, any voltage between 0V and 5V can be expressed as an integer value between 0 and 1023. By multiplying the digitally converted voltage value by 5/1024, we can scale the value back down to a value between 0 and 5 (for purposes such as viewing the voltage through the serial monitor) that appears to be analog, but the 1024 points of precision constraint remains.

The ADMUX register is used to configure ADC on the ATMega328P. Bits REFSx are used to set VREF, bit ADLAR can change the bit resolution between 8 and 10 bits, and finally MUXx bits select which analog pin is to be used for conversion. ADCSRA and ADCSRB are control and status registers used for ADC, however only ADCSRA is applicable in the case of this lab. This register contains ADEN (initialized to 1 to enable conversion), ADSC (start conversion bit, which is to be manually set to high when a conversion is to be done, and is set automatically back to low when conversion is complete), ADIE (enables interrupts to be called upon completion of an ADC, not applicable in our case and is set to low), and finally 3 ADPSx bits. These bits determine a prescaler value for the ADC conversion, as the ADC uses the same internal clock as the microcontroller but requires a prescaler as the internal clock speed of 16MHz is too fast for the ADC. A prescaler of 128 is used in our case.

As the ATMega328P has 8 bit registers, but the ADC allows for 10 bit resolution, two registers ADCH and ADCL are used in tandem to store the final ADC value post conversion.

DAC

Our digital analog converter (DAC) device for this lab is the Maxim MAX518, an 8 pin converter that communicates with the microcontroller using two wire I2C protocol. The DAC reads in an 8 bit value (note the decreased resolution from the microcontroller's internal ADC), and can output a corresponding analog voltage out of its two analog out pins. VDD and GND pins on the DAC determine the extremes of the possible voltages out of the OUT pins on the DAC. Although the DAC was used alone for this lab, the pins A1 and A0 can be set high and low to create 4 distinct DAC addresses, making I2C communication with multiple DACs possible. Similarly to the scaling performed for the ADC, the user's desired analog voltage (in this case between 0 and 5), must be scaled to fall within the 8 bit resolution range of 0 to 255, which can be done by multiplying the desired analog voltage by 255/5 and then casting that value as an integer. This final value is sent to the DAC over I2C.

Software Explanation

I2C

We utilize the I2C protocol via a Two-Wire Interface (TWI) to communicate with the I2C interface on the MAX518 DAC used in this lab. Let's go through the functions used in our code:

[1] i2c_init();

This function handles initializing the TWI clock by configuring the TWSR and TWBR registers of the ATmega328P. Only needs to be called at the start of the program.

[2] i2c_start();

This function issues a start condition (see figure 1) and sends address and transfer direction. We passed in the address of the MAX518 DAC, which based on our configuration was "0x58". Setting pins A1 and A0 of the MAX518 to ground as one possible address. Since there are 4 different bit combinations of A0 and A1 we can have a max of 4 MAX518's if we wanted to on the same bus (see figure 2). It will return 0 if the i2c device is accessible and 1 if it has failed to access the device.

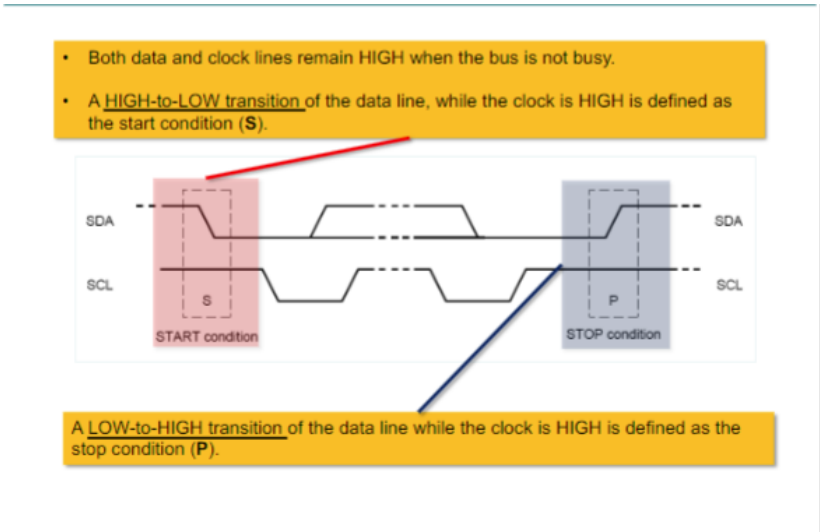


Figure 1

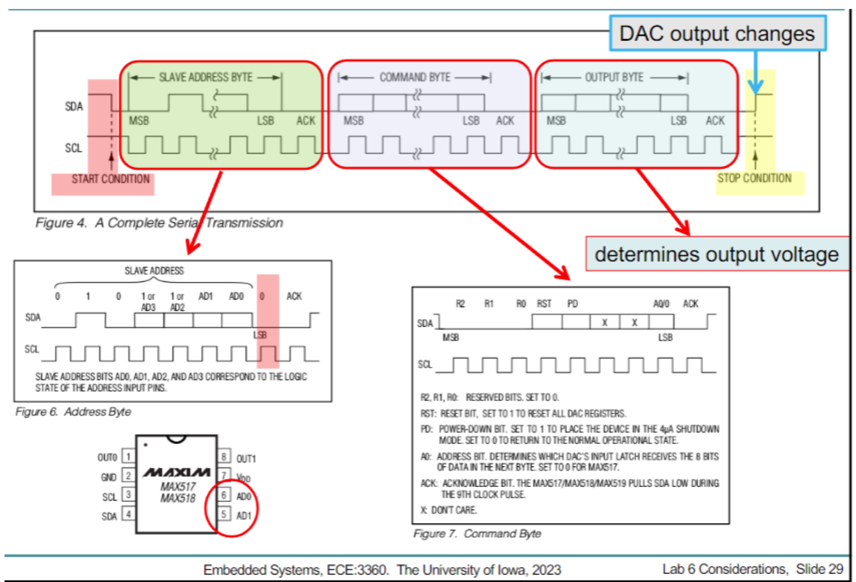


Figure 2

[3] i2c_write();

This function sends a single byte to an I2C device. It sends data to the previously addressed device, waits until the TWINT bit in the TWCR register is set to 1 (waits until transmission of data completes). It will then check the TWI status register to verify whether the byte transfer was successful. It will return 0 for failure and 1 if it was a success.

[3] i2c_stop();

This function sends “1” bits via TWINT, TWEN and TWSTO to “start” the stop condition. It waits until the TWSTO bit is turned back to 0 and bus released to complete the operation.

To see examples of how I2C is used in process in our code, please see lines (158-161) and (136-141).

USART

We used the microcontroller to send and receive data from the serial monitor interface via the Arduino IDE. In this lab, the bits we sent are represented via RS-232 voltage levels. The functions we used for USART come from the ATmega328P datasheet.

USART stands for Universal Synchronous/Asynchronous Receiver Transmitter.

[1] USART_Init(unsigned int ubrr);

This function sets the baud rate (bits per second), enables the receiver and transmitter and sets the frame format. The “ubrr” variable is calculated outside the function as FOSC/16/BAUD-1, where FOSC is the microcontroller's clock speed and BAUD is the desired baud rate for data transmission.

[1] USART_Recieve(void);

This function waits for bit RXC0 to be set which signals that it has received data. It will return the UDR0 register when received.

[1] USART_Recieveunsigned char data);

Similar deal, except we wait for the transmit buffer to be empty (UDRE0 = 1). When it is emptied we can put “data” into UDR0.

Decimal to String, String to Decimal

Multiple standard C libraries were used to perform conversions between C-style string character arrays and numbers of integer, float, and decimal type. *atoi()* can be passed elements of a character array containing ascii characters for integers, and produces an integer with each place value of the integer corresponding to its respective ascii character in the original character array.

dtostrf() was likewise used to convert floating point values to a C-style string, with the function taking in the desired character array to be modified, total desired characters/digits (right justified), and decimal points of precision as parameters.

4. Conclusion

This lab provided useful insight into the broadly applicable techniques of analog to digital conversion and digital to analog conversion, along with a primer to using the equally useful I2C communication protocol which can be used to communicate with devices with a low number of pins and wires. Additionally, peripherals such as serial monitoring for both transmitting information and viewing data was used.

5. Appendix A: Source Code

```c
/*
 * LAB5.c
 * C language file for lab 5 in ECE:3360
 * Created: 4/10/2023 12:52:52 PM
 * Authors : Max Finch, Tiger Slowinski
 */

#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "twimaster.c"
#include <util/delay.h>

#define FOSC 16000000 // Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
#define DAC 0x58


void USART_Init( unsigned int ubrr)
{
    ///*Set baud rate */
    UBRR0H = (unsigned char)(ubrr>>8);
    UBRR0L = (unsigned char)ubrr;
    /*Enable receiver and transmitter */
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);
    /* Set frame format: 8data, 2stop bit */
    UCSR0C = (1<<USBS0)|(3<<UCSZ00);
}
void USART_Transmit( unsigned char data )
{
    /* Wait for empty transmit buffer */
    while ( !( UCSR0A & (1<<UDRE0)) )
    ;
    /* Put data into buffer, sends the data */
    UDR0 = data;
}
unsigned char USART_Receive( void )
{
    /* Wait for data to be received */
    while ( !(UCSR0A & (1<<RXC0)) );
    /* Get and return received data from buffer */
    //Dtostrf
    return UDR0;
}
```

```
void ADC_Init(void)
{
    ADMUX = (1<<REFS0); //set to VCC and to A0
    ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);

}
void get_ADC(void)
{
    ADCSRA = ADCSRA | (1<<ADSC);

    while (!(ADCSRA & (1 << ADIF)));
    ADCSRA |= 1 << ADIF;

    float ADC_val = (ADC*5.0/1024.0);
    char buffer[7];
    dtostrf(ADC_val,4,2,buffer);
    USART_Transmit('v');
    USART_Transmit('=');
    for(int i = 0; i < sizeof(buffer); i++){
        USART_Transmit(buffer[i]);
    }
    USART_Transmit(' ');
    USART_Transmit('V');
    USART_Transmit(' ');
}
unsigned char data[11];
int sinn[] = {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};

void W_command()
{
    int out = data[2]-'0';
    int x = 0;
    int r;
    if(data[8] == ' ') //single digit
    {
        r = data[7]-'0';
        x =1;

    }else if(data[9] == ' ') //two digits
    {
        int i = data[7]-'0';
        i = i*10;
        int j = data[8]-'0';
        r = i+j;
        x = 2;

    }else  //else three digits
    {
        int i = (data[7]-'0')*100;
        int j = (data[8]-'0')*10;
```

```
        int k = data[9] - '0';
        r = i+j+k;
        x = 3;
    }

    char first_msg [] = {'G','e','n','e','r','a','t','i','n','g',' ' };
    for(int i = 0; i < sizeof(first_msg); i++){
        USART_Transmit(first_msg[i]);
    }
    //printing r
    char r_string [x];
    itoa(r, r_string, 10);
    for(int i = 0; i < sizeof(r_string); i++){
        USART_Transmit(r_string[i]);
    }
    char second_msg[] = {' ','s','i','n','e',' ','w','a','v','e',' ','c','y','c',
    'l','e','s',' ','w','i','t','h',' ','f','='};
    for(int i = 0; i < sizeof(second_msg); i++){
        USART_Transmit(second_msg[i]);
    }
    //printing frequency
    USART_Transmit(data[4]);
    USART_Transmit(data[5]);
    USART_Transmit(' ');
    char msgggg[] = {'H','z',' ','o','n',' ','D','A','C',' ','c','h','a','n','n','e','l',' '};
    for(int i = 0; i < sizeof(msgggg); i++){
        USART_Transmit(msgggg[i]);
    }
    //print channel
    USART_Transmit(data[2]);

    for(int i=0; i<r; i++)
    {
        for(int j=0; j<(sizeof(sinn)/2); j++)
        {
            i2c_start(DAC);
            i2c_write(out);
            i2c_write(sinn[j]);
            if(data[4] == '1'){_delay_us(1250);}
            if(data[4] == '2'){_delay_us(500);}
            i2c_stop();
        }
    }

}

void S_command()
{

    int out = data[2]-'0';

    char fltstr[5];
    strncpy(fltstr, data + 4, 4);
    fltstr[4] = 0;
    float final_val = atof(fltstr);
    int final_int = final_val*51;
```

```
    i2c_start(DAC);
    i2c_write(out);   //0 or 1
    i2c_write(final_int);
    i2c_stop();
    char msg [] = {'D','A','C',' ','c','h','a','n','n','e','l',' '};
    for(int i = 0; i < sizeof(msg); i++){
        USART_Transmit(msg[i]);
    }
    USART_Transmit(data[2]);
    char msgg [] = {' ','s','e','t',' ', 't', 'o',' '};
    for(int i = 0; i < sizeof(msgg); i++){
        USART_Transmit(msgg[i]);
    }
    USART_Transmit(data[4]);
    USART_Transmit(data[5]);
    USART_Transmit(data[6]);
    USART_Transmit(data[7]);
    USART_Transmit(' ');
    USART_Transmit('V');
    USART_Transmit(' ');
    USART_Transmit('(');
    char msggg [3];
    itoa(final_int, msggg, 10);
    for(int i = 0; i < sizeof(msggg); i++){
        USART_Transmit(msggg[i]);
    }
    USART_Transmit('d');
    USART_Transmit(')');
    USART_Transmit(' ');
}


int main(void)
{

    USART_Init(MYUBRR);
    ADC_Init();
    i2c_init();

    while(1)
    {

        /* Wait for data to be received */
        int i = 0;
        int len = 0;
        unsigned char c = USART_Receive();

        if ( c == 'G')
        {
            get_ADC();
        }
        //goto G function
        else if(c == 'S')
        {
            len = 8;
            data[0] = c;
```

```
            i=1;
            while(i != len)
            {
                data[i] = USART_Receive();
                i++;
            }
            S_command();
        }else if(c == 'W')
        {
            len = 50;
            data[0] = c;
            i=1;
            bool done = false;
            while(i != len)
            {
                data[i] = USART_Receive();
                if(data[i] == ' ')
                {
                    i = len;
                    }else{
                    i++;
                }
            }
            W_command();
        }
    }


}
```

6. Appendix B: References

Atmel Corporation. *AVR Instruction Set Manual - Microchip Technology*. <https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-0856-AVR-Instruction-Set-Manual.pdf>.

Arduino. *Arduino UNO Rev3 with Long Pins*.<https://docs.arduino.cc/retired/boards/arduino-uno-rev3-with-long-pins>

Damadmai. “PFLEURY/TWIMASTER.C at Master · Damadmai/Pfleury.” *GitHub*, <https://github.com/damadmai/pfleury/blob/master/twimaster.c>

Analog Devices. "MAX517/MAX518/MAX519: Low-Power, Single-Supply, Voltage-Output, 8-Bit DACs in μMAX." Datasheet. Analog Devices, 2004, <https://www.analog.com/media/en/technical-documentation/data-sheets/MAX517-MAX519.pdf.>

AVR-GCC Libraries: I2C Master Library, <https://damadmai.github.io/pfleury/group__pfleury__ic2master.html>

Lab Lecture Slides provided by Professor Beichel