Team Members: Louis Solovy & Hongyu Zeng

Professor Beichel

ECE:3360 Embedded Systems

Post-Lab Report 5

**1. Introduction**

The goal of this lab was to gain some experience with C-based programming of AVR microcontrollers, serial interface protocols, ADC and DAC. Since we decided to opt out of integrating the MAX518 we didn’t need to use the serial interface protocols (I2C) along with not reading the DAC output voltage. The lab’s objective was to create a C program that would get the ADC voltage reading and output it to the Serial monitor in the Arduino app. The C program consisted of having commands, G and M. If G was entered in the serial monitor log, it would output a single voltage measurement from the ADC. If M was entered, it would need to also have two more parameters, n (number of measurements) and dt (time between measurements). So if M,n,dt was entered into the serial monitor log it would get multiple voltage measurements from the ADC based on the parameters n and dt. We also need to use a potentiometer to allow for the ADC voltage to change by the user. The functionality of the potentiometer would allow for readings between 0-5V to be outputted to the serial monitor.

**2. Schematic**

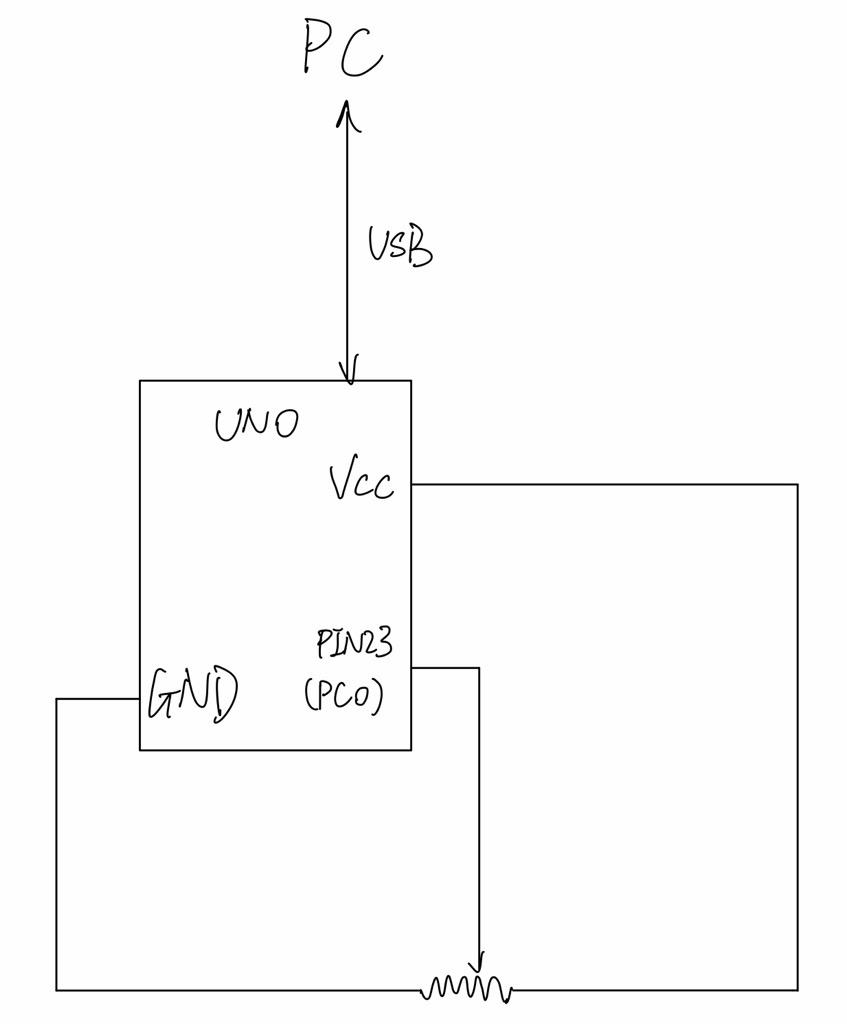
Figure 1: Circuit Diagram

Figure 1 shows the circuit diagram and how it was implemented for this lab. The I/O configure only consisted of having the potentiometer (B103). The front pin was connected to pin 23 (PC0 - ADC0) and then the back pin on the right connected to GND and the back pin on the left connected to Vcc. Since there was no datasheet provided to us that showed us the connection of the potentiometer was needed to look up how to. Since this lab didn’t really consist of creating a circuit, this was all that was needed to meet the objective of the lab.

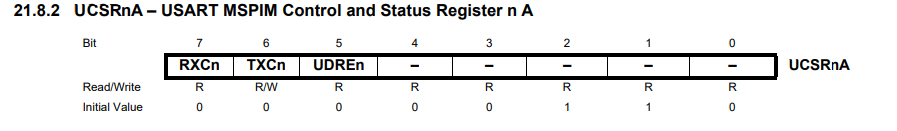
**3. Discussion**

The planning for this project consisted of two basic phases: Hardware and Software. The hardware design of this lab was very simple. It only consisted of using a potentiometer to change the input voltage that would be outputted to the ADC. Other than that, this was the only competent that was needed for the hardware of this lab.

For the software component of this lab, this would be the first lab that we would be using C language to program the microcontroller. The first step in constructing the software was to configure the USART. This was done by creating a function call usart\_init(), that would set the baud rate along with enabling the receiver and transmitter using TXEN0 and RXEN0. The way we set the baud rate was using the USART baud rate registers UBRR0H (high) and UBRR0L (low). Since the baud rate was 12 bits we put the high to 4 bits and the low to 8 bits. Then we needed to set the character size of the format to 8 bits by setting UCSR0C bit 1 and 2 to 1; to enable receiver and transmitter we set the bit 3 and 4 on UCSR0B to 1.

Next was to implement the usart\_transmit(char arr[]) and usart\_receives(). For the transmit function we put each char in the char array in the UDR0 using polling style. We only transmit the char when the USART data register is empty. This is done by checking if the bit 5 on the UCSR0A is set to 1. For receiving char, we are taking one char at a time. This is done when the bit 7 on UCSR0A is set to 1 which means that receiving is complete (refer to figure 2).

Figure 2: UCSR0A



For ADC, we first needed to configure it. We used the 5V Vcc for reference voltage by setting bit 6 to 1 on ADMUX (refer to figure 3). And since we are using PC0, we don’t need to change the bit 3 to bit 0 since its initial value is 0 which corresponds to ADC0. And to enable ADC, we set bit 7 on ADCSRA to 1 (refer to figure 4).The ADC value we have is a 10 bit resolution according to the datasheet we needed to have the input clock frequency between 50-200kHz. We needed to have a prescaler of 128 so our clock frequency was 125kHz, which is in the range. This was done by setting the bit 2-0 on ADCSRA to 1 (refer to figure 5).

.

Figure 3: ADMUX

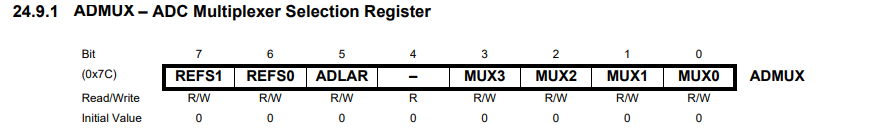


Figure 4: ADCSRA

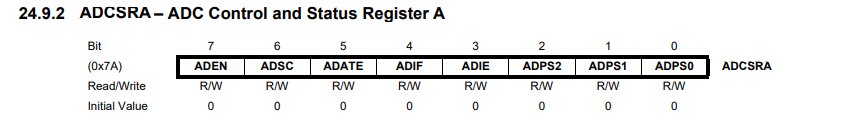
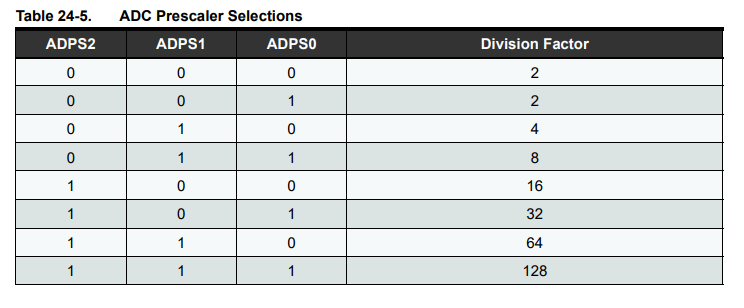
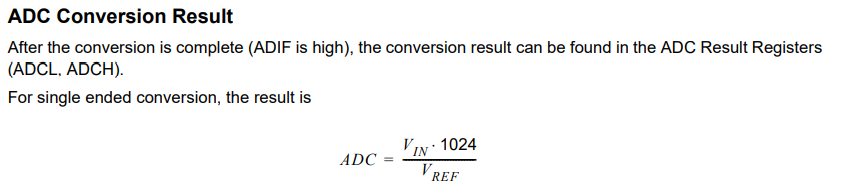


Figure 5: ADC Prescaler



To read ADC value, we set the bit 6 on ADCSRA to 1 to start the conversion from analog (voltage input to PC0) to digital values (0-1023). When the conversion is finished (the bit 6 will return to 0 when the conversion completes), we read the value from ADC. Then we convert it back to voltages by dividing 1024 times 5 (refer to figure 6).

Figure 6: ADC Conversion



Next part is how our program works overall. We have a main loop that first will print out “Please Give Command:” in the serial monitor. Then we have a nested for loop that will receive the characters entered in the serial monitor and save them to a char array. If the character that was entered was a space then the loop will break and echo what was entered by the user onto the serial monitor. It will then check if the first letter entered by the user was a G. If it is then the serial monitor will output the ADC voltage for a single measurement. If the first letter isn't G, we assume that it is M that has two parameters that are then split by commas. We then use “sscanf” to convert the string into integers for the sample time and interval. Then we have a loop that will loop until the sample time is reached and will print out the time at each interval along with the voltage at that interval. We also needed to check to make sure that the number of measurements was between 2 -20 or equal. We also needed to check and make sure that the interval was between 1-10 seconds or equal. If these conditions were not met an error message would be printed to the serial monitor. To allow for our program to work we implemented a delay that will wait the specified interval time and then print the data to the serial monitor. In our main program, when it prints the data to the serial monitor we have to use “sprintf” to format our data so that it prints in a nice manner.

Here are some notes on how to run and use our program. For G type command, type “G” and a space bar (this is how we check the end of a command). For M type command, type “M,”, value for sampling times, “,”, value for sampling interval, and a space bar. (for example, to sample 5 times with 3s interval, type “M,5,3 ”) Make sure you don’t miss the space bar in the end and don’t type space bar in the middle of a command. Also here are some helpu notes on sprintf: Bailey helped us configuring the Microchip settings. On AVR/GNU Linker, in General, select “use vprintf Library”; in Miscellaneous, “-lprintf\_flt -lm”.

Testing datasheet:

| Voltage reading is close to 0 when potentiometer is turned far left | X |
| --- | --- |
| Voltage reading is close to 5 when potentiometer is turned far right | X |
| G Command working correctly | X |
| When the parameters of M are in range its giving readings at correct time | X |
| When the sample times is smaller than 2 or greater than 20,  It’s giving an error for sampling time | X |
| When the interval is less than 1 or greater than 10,  It’s giving an error for interval time | X |

Libraries used:

#include <stdio.h>

#include <avr/io.h>

#include <util/delay.h>

#include <avr/pgmspace.h>

**4. Conclusion**

Overall, the goal of this lab was to gain some experience with C-based programming of AVR microcontrollers, serial interface protocols, and ADC. After completing this lab we would say that we have achieved this goal. During the lab there were some problems that we ran into when constructing our code. One of them being that when trying to just read the ADC and have it print to the serial monitor, nothing was printing. This was because we didn’t know that we needed to have a prescaler. At first we have to default but when going to get help from one of the TA’s, he told us that we needed to have an ADC prescaler of 128. After we changed this our reading was working. Another problem that we came across was figuring out how to connect the potiemeter to the microcontroller. Since there was no datasheet given to us we need to look up how to connect it. We didn’t find the exact datasheet, but we used our knowledge to help us connect it to the microcontroller. Some other small problems that we ran into were just the basics of writing in C, since both of us have had little to zero experience writing in C, stackoverflow was very helpful to us. After getting the basics of C down, the lab was pretty self-explanatory. In conclusion, after completing this lab we can now construct a C program that will get the voltage measurements from the ADC while using a potentiometer to change the readings.

**5. Appendix A: Source Code**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

/\*

\* Lab5.c

\*

\* Created: 4/9/2021 12:09:14 PM

\* Author : Hongyu & Louis

\* source used: https://embedds.com/programming-avr-usart-with-avr-gcc-part-1/

\* https://www.learncpp.com/cpp-tutorial/passing-arguments-by-address/

\* https://maker.pro/custom/tutorial/how-to-take-analog-readings-with-an-avr-microcontroller

\* https://stackoverflow.com/questions/905928/using-floats-with-sprintf-in-embedded-c

\* https://stackoverflow.com/questions/3889992/how-does-strtok-split-the-string-into-tokens-in-c

\* https://www.tutorialspoint.com/c\_standard\_library/c\_function\_sscanf.htm

\* usart\_prints and usart\_putc are only used when building and testing the program, no longer needed

\*/

// define clock frequency and calculate the corresponding baudrate

#ifndef F\_CPU

#define F\_CPU 16000000UL

#endif

#define USART\_BAUDRATE 9600

#define UBRR\_VALUE (((F\_CPU / (USART\_BAUDRATE \* 16UL))) - 1)

#include <stdio.h>

#include <avr/io.h>

#include <util/delay.h>

#include <avr/pgmspace.h>

void usart\_init();

//void usart\_prints(const char arr[]);

unsigned char usart\_receives();

//void usart\_putc(char t);

void usart\_transmit(char arr[]);

void readADC(float \*ptr);

void delay1s();

int main(void)

{

// temp char for holding received command char

unsigned char c;

// holding command

char command[25];

// holding string output

char outstr[25];

// holding float value for voltage

float v;

// how many times we should sample, based on M command argument

int sampletimes;

// sampling intervals based on M command argument

int interval;

// whether the M command arguments are valid

int valid; // 0 is invalid

// initialize usart

usart\_init();

// initialize ADC

ADC\_init();

// repeatly running the program

while (1) {

valid = 1;

int i = 0;

int counter = 0;

usart\_transmit("Please Give Command: ");

for (i=0;i<=24;i++){

c = usart\_receives(); // Get character

command[i] = c;

if (c == ' ') {

break; // space bar indicate the end of a command

}

}

command[i] = '\0';

// echo the command back to the screen

usart\_transmit(command);

usart\_transmit("\n");

// G type

if (command[0] == 'G') {

usart\_transmit("G type\n");

readADC(&v); // get adc reading and convert it to voltages, store in v

sprintf(outstr, "v = %.3f V\n", v); // give output to the serial monitor

usart\_transmit(outstr);

} else { // M type

usart\_transmit("M type\n");

char \*pch; // points to the start of the next string fragment

pch = strtok(command, ",");

while (pch != NULL) { // split the M command to get sampling times and intervals

if (counter == 0) {

usart\_transmit("It's M\n");

} else if (counter == 1) {

sscanf(pch, "%d", &sampletimes);

} else if (counter == 2) {

sscanf(pch, "%d", &interval);

}

counter ++;

pch = strtok (NULL, ","); // memories the modifed string so NULL for no new input

}

if (sampletimes < 2 || sampletimes > 20) { // checking if sampling times is valid

usart\_transmit("Invalid sampling times!\n");

valid = 0;

}

if (interval < 1 || interval > 10 ) { // checking if the interval is valid

usart\_transmit("Invalid interval!\n");

valid = 0;

}

//usart\_transmit("interval is: ");

//char temp[10];

//sprintf(temp, "%d\n", interval);

//usart\_transmit(temp);

if (valid == 1) { // if all argumetns are valid

int j = 0;

int curtime = 0;

int k;

while(1) { // keeps giving samples until break

readADC(&v); // keep calling G command

sprintf(outstr, "t = %d s, ", curtime);

usart\_transmit(outstr);

sprintf(outstr, "v = %.3f V\n", v);

usart\_transmit(outstr);

if (j >= sampletimes - 1) {

break; // reaches the sampling times give

}

j++;

curtime += interval;

delay1s();

for (k = 0; k < interval; k++) { // delay specific amount of time

delay1s();

}

}

}

}

}

}

void usart\_init() {

// Set baud rate

UBRR0H = (unsigned char)(UBRR\_VALUE>>8);

UBRR0L = (unsigned char)UBRR\_VALUE;

// Set frame format to 8 data bits, no parity, 1 stop bit

UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);

//enable transmission and reception

UCSR0B |= (1<<RXEN0)|(1<<TXEN0);

}

/\*void usart\_prints(const char arr[]){

// Uses polling (and it blocks).

int j;

for (j = 0; j <= strlen(arr)-1; j++) {

while (!( UCSR0A & (1<<UDRE0))) {};

UDR0 = arr[j];

}

} \*/

unsigned char usart\_receives() {

// Wait for byte to be received

while(!(UCSR0A&(1<<RXC0))){};

// Return received data

return UDR0; // get char from buffer

}

/\*void usart\_putc(char t) {

while (!( UCSR0A & (1<<UDRE0))) {};

UDR0 = t;

} \*/

void usart\_transmit(char arr[]) {

// Uses polling (and it blocks).

int j;

for (j = 0; j <= strlen(arr)-1; j++) {

while (!( UCSR0A & (1<<UDRE0))) {}; // until buffer empty

UDR0 = arr[j]; // put the char to the buffer

}

}

void ADC\_init() {

ADMUX = (1 << REFS0); // AVCC with external capacitor at AREF pin

ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // enable ADC and set prescalar to /128

}

void readADC(float \*ptr) {

ADCSRA |= 1 << ADSC; // start conversion

while (ADCSRA & (1 << ADSC)) { // until finished conversion

}

uint16\_t digitalV = ADC; // get ADC reading

//char temp[20];

//sprintf(temp, "%d", digitalV);

//usart\_transmit(temp);

\*ptr = (digitalV/1024.0)\*5.0; // change value where ptr is pointing at

//\*ptr = 6/5.0;

}

void delay1s() {

\_delay\_ms(1000);

}

**6. Appendix B: References**

Atmel, “Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash”, <<https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf>> August 2013.

Embeds, “Programming AVR USART with AVR-GCC”, <<https://embedds.com/programming-avr-usart-with-avr-gcc-part-1/>> January 2015.

LearnCpp, “Passing arguments by address” ,<<https://www.learncpp.com/cpp-tutorial/passing-arguments-by-address/>> December 2020.

MakerPro, “Use and AVR microcontroller incorporated ADC to take analog readings from your surrounding”, <<https://maker.pro/custom/tutorial/how-to-take-analog-readings-with-an-avr-microcontroller>> June 2018.

Stackoverflow, “Using floats with sprintf() in embedded C”, <<https://stackoverflow.com/questions/905928/using-floats-with-sprintf-in-embedded-c>> January 2020.

Stackoverflow, “ How does strtok() split the string into tokens in C”, <<https://stackoverflow.com/questions/3889992/how-does-strtok-split-the-string-into-tokens-in-c>> April 2020.

Tutorialspoint, “ C library function - sscanf()”, <<https://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm>> January 2021.