Cole Brooks

Thomas Butler

55:036 Embedded Systems

Post-Lab Report 5

Introduction

The goal of this lab is to gain some experience with C-based programming of AVR microcontrollers. We are also going to get some experience using serial interface protocols like I2C, USART, and other RS232 compliant protocols. We’ll also get some experience using the onboard analog to digital converter (ADC).

In this lab, we are creating a terminal controlled voltage reading device. This project is almost entirely software based, because the hardware we used is all onboard the Arduino Uno. The only circuits we included on our breadboards were a few different voltage dividers to ensure that our voltage readings from the ADC were accurate.

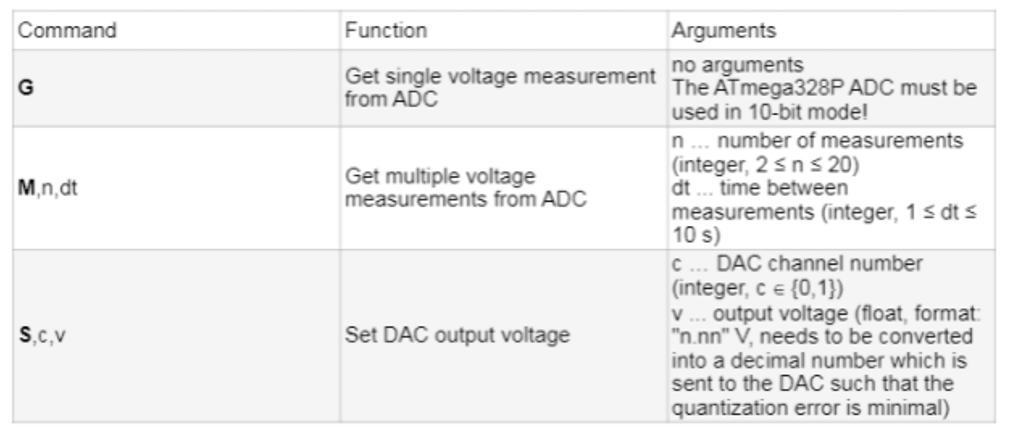
Schematic

Discussion

The Arduino used in the circuit was programmed with AVR-C code. That source code is included in the Source Code section of this lab report.

To complete our project, we verified the following requirements from the lab description:

|  |  |
| --- | --- |
| **Requirement** | **Status:** |
| The system responds to only the commands listed in the command table below. | Verified |
| When command G is sent from the serial monitor, the system gets a single voltage measurement from the ADC and displays it to the serial monitor. | Verified |
| When command M,n,dt is sent from the serial monitor, the system gets n voltage measurements from the ADC with dt time in seconds between each measurement. | Verified |
| The system responds to all commands as outlined in the table below. | Verified |



The planning for this lab occurred entirely online. Like the past labs, our primary mode of communication was an online messaging platform, Discord. Due to some complications with this lab, the midlab review was not required. This freed us up to work the lab on our own timeline. We began working on this lab Friday the 16th of April and finished on Monday the 19th.

The first step for this lab was figuring out how to get the microcontroller to communicate with our serial monitor. It was quite a simple task after researching USART on the internet. It also helps that we’re programming in C now, so finding tutorials on the internet is quite a bit easier. We began attempts at serial communication on Friday the 16th, and had fully functioning serial communication by Saturday the 17th. At this point, we were able to command the microcontroller to take a single reading from the ADC with the G command. Note that at this point, we were not getting proper readings from the ADC, but the response from the microcontroller was as expected.

After getting serial communication to work properly, we moved our focus to the analog to digital converter. This was much more challenging than getting serial communication to work. We spent the rest of the weekend studying the microprocessors data sheet in order to figure out everything we could about the onboard analog to digital converter. We were stuck for some time getting the same reading from the ADC, but after studying the data sheet some more, we were able to figure out what we were doing wrong and got the ADC working properly. At this point our ‘G’ command was working properly.

After getting the ADC working properly, we simply needed to add code to take a wider variety of inputs from the user. We added checks in our USART receive function to check for an ‘M’ command. We figured out delays in C using a library, and then added functionality for multiple measurements spanning different amounts of time.

As a last step for this lab, we spent a day attempting to find and fix any bugs in the system. We found that our receive function did not work exactly as we expected, but we were able to quickly fix the issue.

Calculations

Voltage Divider Equation:

Voltage Between Resistors = Vcc \* ( RGround / ( RGround + RVcc ) )

Conversion from ADC 10 bit to Voltage:

ADC = ( Vin \* 1024 ) / Vref

Conclusion

In this section we will discuss what we have learned in this lab and how we expect to be able to apply the concepts from this lab in the future. After discussing the lessons we have learned from this lab, we will briefly talk about things that we thought went well with this lab.

The first lesson we learned was how much different coding a microcontroller with C is than coding with assembly. Almost everything is quicker and more readable. After reflecting as a group we came to the conclusion that it was a good exercise to learn assembly code because before that we had no experience dealing with registers or simple instructions. Now that we have experience with such a low level of programming, we have a deeper understanding of our higher-level code.

The next lesson we learned was about serial communication. At first, it seemed very confusing and complicated to get our microcontrollers communicating with our PCs. Obviously, communication between devices is an integral part of embedded systems, so it is very important to have a solid understanding of serial communications. It was very helpful for us to take some time to research USART and I2C before programming anything, because once we understand the protocols, they weren’t actually as complicated as once thought. It also helps that these protocols have been implemented many times before, so there was no shortage of assistance on the internet. Almost every system we use in the future will use some form of serial communication, so learning about different types of serial communication in this lab will be a great help in our future careers.

The final lesson we learned was that not everything can be found on the internet, even when dealing with C code. After having such an easy time figuring out serial communication, we attempted to do the same thing to configure out ADC. Unfortunately, none of the tutorials that we found on the internet were doing exactly the same thing that we were attempting to do, and we got lost in the weeds quite a few times. We realized that it would be more time effective to simply sit down and read the data sheet to get the information we needed.

Reflecting on this lab, some things that went well were our time management, our problem solving process, and our communication. We felt as though our commitment to doing well in those areas helped us to outperform ourselves in this lab.

Time management for any class is difficult, but we have found that especially true for Embedded Systems. We have found that working consistently on a project from when it is assigned is the best way to get through these labs. For some of the other labs we have felt like we were cutting very close to the deadline, but for this one, we were able to confidently turn our lab in a full day ahead of schedule. This is thanks to us creating a schedule early, and sticking to our commitments throughout that schedule. It will be imperative to continue good time management for the project to come, as well as in our careers.

Our problem solving process was also much better for this lab. In the past we have let each other deal with bugs as they arrive, attempting to finish the lab in parallel rather than swarming an issue as it comes up. In this lab, however, we were able to finish most of the features quite quickly, and then we worked together to swarm the bugs. This lead to a dramatic increase in the rate at which we were able to solve problems. Through this experience we have learned the value in working simple problems on our own, and coming together to figure out the more difficult ones.

The last thing that we felt went very well in this lab was our communication. This has been a common theme throughout the labs in this class. We have used technology to communicate effectively and efficiently with each other, leading to success in this class. Without communication, none of the other parts of the lab would be possible, so we want to focus on strong and efficient communication on the project as well as in our future careers.

Appendix A: Source Code

/\*

\* Lab5.c

\*

\* Created: 4/8/2021 10:24:57 AM

\* Author : Cole Brooks

\* Thomas Butler

\*

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define *F\_CPU* 16000000L // 16 MHz

#include <util/delay.h>

#define USART\_BAUDRATE 9600

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

#define ADC\_CONVERSION 5.0/1024

////////////////////////////////////////////////////////////////////

//

// Function: usart\_tx

//

// Arguments: uint8\_t data:

//

// Purpose: Sends data to serial monitor

//

////////////////////////////////////////////////////////////////////

void usart\_tx(*uint8\_t* data)

{

//wait while previous byte is completed

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

// Transmit data

UDR0 = data;

}

///////////////////////////////

//

// Function: print

//

// Purpose: print message byte by byte

//

///////////////////////////////

void print(char msg[]){

int i;

for (i = 0; i < *strlen*(msg); i++) {

usart\_tx(msg[i]);

}

}

////////////////////////////////////////////////////////////////////

//

// Function: usart\_init

//

// Arguments: Void

//

// Purpose: set baud rate, enable rx and tx, and set

// frame formate: 8 data, 2 stop bit

//

////////////////////////////////////////////////////////////////////

void usart\_init(void)

{

// Set baud rate

UBRR0H = (*uint8\_t*)(UBRR\_VALUE>>8);

UBRR0L = (*uint8\_t*)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);

}

////////////////////////////////////////////////////////////////////

//

// Function: adc\_read

//

// Arguments: void

//

// Purpose: reads from ADC and returns value

//

////////////////////////////////////////////////////////////////////

*uint16\_t* adc\_read(void)

{

// Trigger a voltage read

ADCSRA |= (1<<ADSC);

while(ADSC == 1)

{

// Conversion Processing

}

return ADCW;

}

////////////////////////////////////////////////////////////////////

//

// Function: usart\_rx

//

// Arguments: Void

//

// Purpose: Receives data from serial monitor

//

////////////////////////////////////////////////////////////////////

*uint16\_t* usart\_rx(void)

{

// Wait for byte to be received

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

// Return received data

return UDR0;

}

////////////////////////////////////////////////////////////////////

//

// Function: adc\_init

//

// Arguments: void

//

// Purpose: initializes the analog to digital converter

//

////////////////////////////////////////////////////////////////////

void adc\_init(void)

{

// Set ADC voltage reference and input channel (Port A1)

ADMUX |= (1<<REFS0) | (1<<MUX0); //AVcc as voltage reference

// Set up the status register

ADCSRA |= (1<<ADEN) | (1<<ADSC) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);

}

////////////////////////////////////////////////////////////////////

//

// Function: adc\_get\_double

//

// Arguments: void

//

// Purpose: gets an ADC conversion and converts the output to a double

//

//

////////////////////////////////////////////////////////////////////

double adc\_get\_double(void)

{

double adc = adc\_read();

return adc \* 5 / 1024;

}

/////////////////////////

// Function: measure\_multiple

// Purpose: execute multiple measurements following M command

/////////////////////////

void measure\_multiple(int n, int dt){

char out[5];

int dt\_out = dt;

while (n > 0){

// delay stuff

int dt\_buff = dt;

while(dt\_buff > 0)

{

*\_delay\_ms*(1000);

dt\_buff --;

}

print("t = ");

*sprintf*(out,"%i",dt\_out);

print(out);

print(" s, V = ");

double v = adc\_get\_double();

*sprintf*(out,"%d.%02u", (int) v, (int) ((v - (int) v ) \* 100) );

print(out);

print("V\n");

n = n - 1;

dt\_out = dt\_out + dt;

}

}

////////////////////////

// Function: get\_string

// Purpose: gets string from serial monitor

///////////////

const char\* get\_string(char input\_str[]){

char buffer[10];

*uint16\_t* input = usart\_rx();

int i = 0;

while (input != 10){

*itoa*(input, buffer, 10);

//print(buffer);

input\_str[i] = *atoi*(buffer);

i = i + 1;

input = usart\_rx();

}

return input\_str;

}

////////////////////////////////////////////////////////////////////

// Function: main

//

// Purpose: Drives the program

//

////////////////////////////////////////////////////////////////////

int main(void)

{

// Initializations

usart\_init();

adc\_init();

// main loop

while (1)

{

char input\_str[10] = {};

char output\_str[20];

get\_string(input\_str);

print("Input: ");

print(input\_str);

print("\n");

if (input\_str[0] == 71){ // G

double v = adc\_get\_double();

//print("G");

*sprintf*(output\_str,"V = %d.%02u V", (int) v, (int) ((v - (int) v ) \* 100) );

print(output\_str);

}

if (input\_str[0] == 77){ // M

int n;

if ((input\_str[1] == 44) && (input\_str[4] == 44)){ // n is two digit number

n = ((input\_str[2]-48)\*10) + (input\_str[3]-48);

if ((input\_str[5] == '1') && (input\_str[6] == '0')){

measure\_multiple(n,10);

}

else{

measure\_multiple(n,input\_str[5] - 48);

}

}

else{

n = input\_str[2]-48;

if ((input\_str[4] == '1') && (input\_str[5] == '0')){

measure\_multiple(n,10);

}

else{

measure\_multiple(n,input\_str[4] - 48);

}

}

}

print("\n");

}

}

Appendix B: References

ATmega328p, 3/10/21 < <https://uiowa.instructure.com/files/14761502/download?download_frd=1> >

ARDUINO\_V2.pdf, 3/10/21 < <https://uiowa.instructure.com/files/14762893/download?download_frd=1> >