Zhengyuan Cai

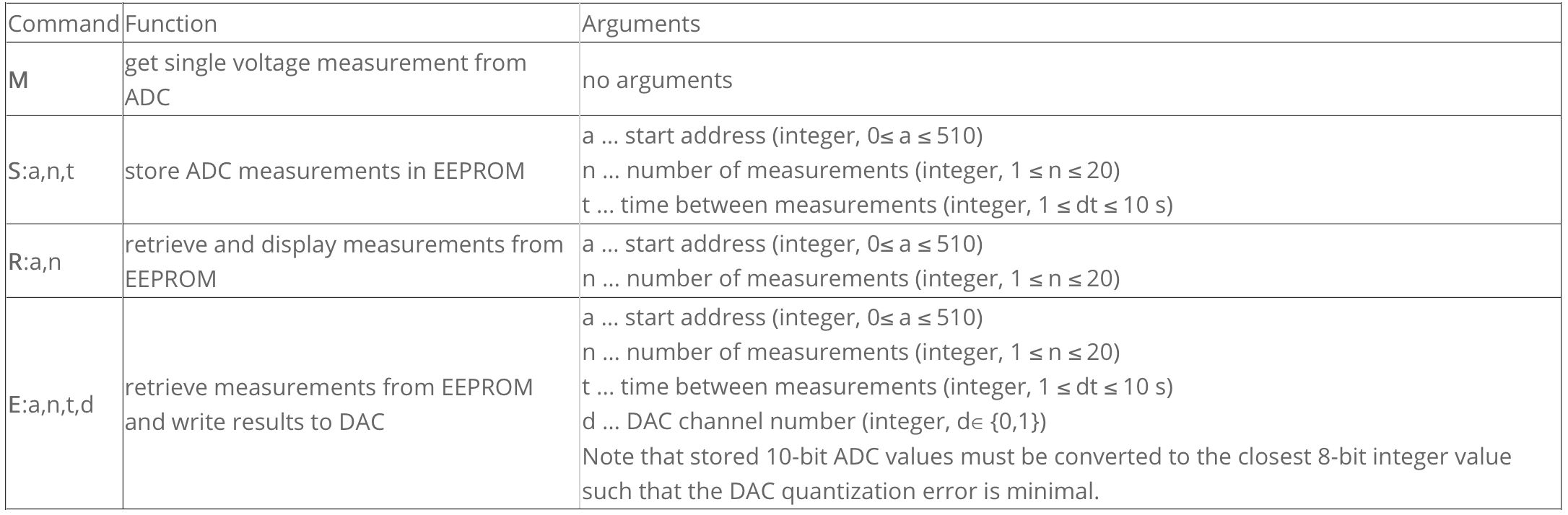
Ben Weinberg

Professor Beichel

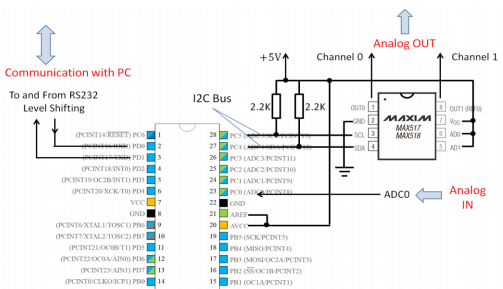
ECE:3360 Embedded Systems

Post-Lab Report 5

1. **Introduction**

The goal of this lab was to build a remote controllable analog data logging system (C language) using the built-in A/D converter (ADC) of the ATmega88PA controller and the MAX518, an external two-channel D/A converter (DAC) with an I2C interface. The analog interface system has an RS232 interface connected to a PC. The PC user will be able to trigger a single voltage measurement, storage of analog voltage measurements in EEPROM, data retrieval from EEPROM, as well as reproduction of stored measurements on one of the DAC channels by means of commands sent through the RS232 interface. The following command are implemented in the system. 

1. **Schematic**



1. **Discussion**

The circuit was programmed with the source code and the output was verified through multimeter. The following items were verified:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Command “M”:*** |  |  |  |
| Displayed voltage correct? | |  | Yes |
| ADC in 10‐bit mode? |  |  | Yes |
| ***Command “S:a,n,t”:*** |  |  |  |
| Displayed voltage correct? | |  | Yes |
| Correct number of measurements? | |  | Yes |
| Time between measurements correct? | | | Yes |
| ***Command “R:a,n”:*** |  |  |  |
| Address working? |  |  | Yes |
| Number of samples (n) working? | |  | Yes |
| Measurements correct ‐ (a) before power cycling | | | Yes |
| (b) after power cycling |  | | Yes |
| **Command “E:a,n,t,d”:** |  |  |  |
| Address working? |  |  | Yes |
| Correct number of DAC values generated? | | | Yes |
| Time between DAC updates correct? | |  | Yes |
| DAC channel selection working? | |  | Yes |
| DAC voltages and message format correct? | | | Yes |

Hardware:

The 28-pin AVR development board with an 8 MHz crystal for frequency generation, ATmega88PA, MAX518 and B10K potentiometer was implemented in the lab, along with two resistors connected to the two wires of I2C interface with +Vdd all as shown in the diagram above.

Software:

We implemented serial communication through RS232 between the microcontroller and a computer, this would feed commands to perform on the microcontroller. Communication between the MAX518 and the microcontroller is implemented using I2C.

When communicating with the DAC, we need to give it 8 bits of data whereas the microcontroller’s ADC was used in 10 bit mode. We used Peter Fleury’s I2C library to communicate between the devices. The address for the DAC was 0b0101111X. The X would change depending on if you were reading from or writing to the DAC. We chose to hook up the A0 and A1 pins to Vcc meaning that the 2nd and 3rd to last bits were 1’s.

We used EEPROM to store values in because this was non volatile memory meaning that even after power cycled, the data would remain.

We also implemented checking the user inputs for errors (out of bounds or unexpected perams).

1. **Conclusion**

From this lab we learned about C-based programming of ACR microcontrollers, serial interface protocols, nonvolatile memory (EEPROM), ADC, and DAC. We also learned how to use external libraries with the microcontroller. All of these things will be useful to use for our final project if needed. We also learned how to turn a range of values into a discrete binary representation of them using an ADC and then do the inverse of that with the DAC.

The I2C communication was also VERY easy to use, especially using that library that was suggested to us in class.

1. **Appendix** A: Code

#define F\_CPU 8000000L // This should match the processor speed

#include <stdlib.h>

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#include <ctype.h>

#include "i2cmaster.h"

#include <avr/interrupt.h>

#include <avr/pgmspace.h> // Routine for FLASH (program memory)

#include <string.h>

#define BAUD\_RATE 9600 // Baud rate. The usart\_int routine

#define REF\_AVCC (1<<REFS0) // reference = AVCC = 5 V

#define REF\_INT (1<<REFS0)|(1<<REFS1) // internal reference 2.56 V

#define DevDAC 0b01011110 // I2C address of DAC

#define I2C\_WRITE 0

#define I2C\_READ 1

// Variables and #define for the RX ring buffer.

#define RX\_BUFFER\_SIZE 64

unsigned char rx\_buffer[RX\_BUFFER\_SIZE];

volatile unsigned char rx\_buffer\_head;

volatile unsigned char rx\_buffer\_tail;

// Function prototypes.

unsigned char uart\_buffer\_empty(void);

void usart\_prints(const char \*ptr);

void usart\_printf(const char \*ptr);

void usart\_init(void);

void usart\_putc(const char c);

unsigned char usart\_getc(void);

uint16\_t adc\_read(void);

void setValue(unsigned char channel, float voltage);

void resetDAC(unsigned char channel);

void EEPROM\_write(unsigned int uiAddress, uint8\_t ucData);

uint8\_t EEPROM\_read(unsigned int uiAddress);

int main(void)

{

unsigned char c;

char str[50];

int totala, totaln, totalt, totald;

int d1, d2, d3,comma;

sei(); // Enable interrupts

usart\_init(); // Initialize the USART

// usart\_printf(fdata); // Print a string from FLASH

// usart\_prints(sdata); // Print a string from SRAM

// initialize ADC

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

ADMUX |= (1<<REFS0); // set reference and channel

usart\_prints("\r\n\r\nWelcome!\r\n\r\n");

i2c\_init(); // init i2c

resetDAC(0); // reset channel 0

resetDAC(1); // reset channel 1

setValue(0,2.0);

setValue(1,4.0);

while(1){

c = usart\_getc();

usart\_putc(c); // Echo back the character

switch(c){

case 'M':

/\*

Get measurement from ADC

FORMAT: 'M'

no args

\*/

usart\_prints("\r\n"); // go to next line

uint16\_t adcval = adc\_read(); // read and print the value of the ADC

break; // end case

case 'S': //

/\*

Store ADC measurement in EEPROM

FORMAT: 'S:a,n,t'

a ... start address (integer, 0? a ? 510)

n ... number of measurements (integer, 1 ? n ? 20)

t ... time between measurements (integer, 1 ? dt ? 10 s)

\*/

// Check for the ':'

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if(c != ':'){ // error check for ':'

sprintf(str, "\r\nExpected ':' but got '%c'\r\n", c);

usart\_prints(str);

break; // break out of case (get new input)

}

comma = 0;

// next value needs to be a digit

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (!isdigit((int)c)){ // error check for a digit

sprintf(str, "\r\nExpected a number but got '%c'\r\n", c);

usart\_prints(str);

break;

}

d1 = (int) c - '0';

totala = d1;

// next value needs to be a digit or a comma

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d2 = (int) c - '0';

totala = d1\*10 + d2;

}

else comma = 1;

// if previous was not a comma, next needs to be an int or comma

if (comma == 0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d3 = (int) c - '0';

totala = (d1\*100) + (d2\*10) + d3;

}

else comma = 1;

if (totala > 510){

usart\_prints("\r\nNumber is too large!");

break;

}

}

// if previous was not a comma, next needs to be a comma

if (comma == 0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

}

comma = 0;

// next value needs to be a digit

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (!isdigit((int)c)){ // error check for a digit

sprintf(str, "\r\nExpected a number but got '%c'\r\n", c);

usart\_prints(str);

break;

}

d1 = (int) c - '0';

totaln = d1;

// next value needs to be a digit or a comma

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d2 = (int) c - '0';

totaln = d1\*10 + d2;

if (totaln > 20){

usart\_prints("\r\nNumber is too large!");

break;

}

}

else comma = 1;

// if previous was not a comma, next needs to be a comma

if (comma==0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

}

comma = 0;

// next value needs to be a digit

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (!isdigit((int)c)){ // error check for a digit

sprintf(str, "\r\nExpected a number but got '%c'\r\n", c);

usart\_prints(str);

break;

}

d1 = (int) c - '0';

totalt = d1;

// next value needs to be a digit or a comma

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d2 = (int) c - '0';

totalt = d1\*10 + d2;

if (totalt > 10){

usart\_prints("\r\nNumber is too large!");

break;

}

}

// implementation

for(int j = 0;j<totaln;j++){

sprintf(str, "\r\nt = %d s, ", (j\*totalt));

usart\_prints(str);

uint16\_t adcval = adc\_read();

uint8\_t partA = (uint8\_t)((adcval & 0xFF00) >> 8);

uint8\_t partB = (uint8\_t)(adcval & 0x00FF);

unsigned int address = totala+(j\*2);

EEPROM\_write(address, partA);

EEPROM\_write(address+1, partB);

sprintf(str, " addr: %d", address);

usart\_prints(str);

for(int i = 0; i<totalt;i++){

\_delay\_ms(1000);

}

}

break; // end case

case 'R':

/\*

Retrieve and display measurements from EEPROM

FORMAT: 'R:a,n'

a ... start address (integer, 0? a ? 510)

n ... number of measurements (integer, 1 ? n ? 20)

\*/

// Check for the ':'

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if(c != ':'){

printf("\r\n Expected ':' but got '%c'\r\n", c);

break; // break out of case (get new input)

}

comma = 0;

// next value needs to be a digit

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (!isdigit((int)c)){ // error check for a digit

sprintf(str, "\r\nExpected a number but got '%c'\r\n", c);

usart\_prints(str);

break;

}

d1 = (int) c - '0';

totala = d1;

// next value needs to be a digit or a comma

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d2 = (int) c - '0';

totala = d1\*10 + d2;

}

else comma = 1;

// if previous was not a comma, next needs to be an int or comma

if (comma == 0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d3 = (int) c - '0';

totala = (d1\*100) + (d2\*10) + d3;

}

else comma = 1;

if (totala > 510){

usart\_prints("\r\nNumber is too large!");

break;

}

}

// if previous was not a comma, next needs to be a comma

if (comma == 0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

}

comma = 0;

// next value needs to be a digit

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (!isdigit((int)c)){ // error check for a digit

sprintf(str, "\r\nExpected a number but got '%c'\r\n", c);

usart\_prints(str);

break;

}

d1 = (int) c - '0';

totaln = d1;

// next value needs to be a digit or a comma

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d2 = (int) c - '0';

totaln = d1\*10 + d2;

if (totaln > 20){

usart\_prints("\r\nNumber is too large!");

break;

}

}

/\*

Retrieve and display measurements from EEPROM

FORMAT: 'R:a,n'

a ... start address (integer, 0? a ? 510)

n ... number of measurements (integer, 1 ? n ? 20)

\*/

// implementation

for(int j = 0;j<totaln;j++){

unsigned int address = totala+(j\*2);

sprintf(str, "\r\naddr: %d , ", address);

usart\_prints(str);

uint8\_t partA = EEPROM\_read(address);

uint8\_t partB = EEPROM\_read(address+1);

uint16\_t whole = ((uint16\_t)partA << 8) | partB;

float adcvalfloat = (float)whole\*5/1023;

sprintf(str, "v = '%.3f'", adcvalfloat);

usart\_prints(str);

}

break; // end case

case 'E':

/\*

Retrieve measurements from EEPROM and write to DAC

FORMAT: 'E:a,n,t,d'

a ... start address (integer, 0? a ? 510)

n ... number of measurements (integer, 1 ? n ? 20)

t ... time between measurements (integer, 1 ? dt ? 10 s)

d ... DAC channel number (integer, d is {0 or 1})

Note that stored 10-bit ADC values must be converted to the closest

8-bit integer value such that the DAC quantization error is minimal.

\*/

// Check for the ':'

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if(c != ':'){

printf("\r\n Expected ':' but got '%c'\r\n", c);

break; // break out of case (get new input)

}

comma = 0;

// next value needs to be a digit

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (!isdigit((int)c)){ // error check for a digit

sprintf(str, "\r\nExpected a number but got '%c'\r\n", c);

usart\_prints(str);

break;

}

d1 = (int) c - '0';

totala = d1;

// next value needs to be a digit or a comma

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d2 = (int) c - '0';

totala = d1\*10 + d2;

}

else comma = 1;

// if previous was not a comma, next needs to be an int or comma

if (comma == 0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d3 = (int) c - '0';

totala = (d1\*100) + (d2\*10) + d3;

}

else comma = 1;

if (totala > 510){

usart\_prints("\r\nNumber is too large!");

break;

}

}

// if previous was not a comma, next needs to be a comma

if (comma == 0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

}

comma = 0;

// next value needs to be a digit

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (!isdigit((int)c)){ // error check for a digit

sprintf(str, "\r\nExpected a number but got '%c'\r\n", c);

usart\_prints(str);

break;

}

d1 = (int) c - '0';

totaln = d1;

// next value needs to be a digit or a comma

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d2 = (int) c - '0';

totaln = d1\*10 + d2;

if (totaln > 20){

usart\_prints("\r\nNumber is too large!");

break;

}

}

else comma = 1;

// if previous was not a comma, next needs to be a comma

if (comma==0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

}

comma = 0;

// next value needs to be a digit

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (!isdigit((int)c)){ // error check for a digit

sprintf(str, "\r\nExpected a number but got '%c'\r\n", c);

usart\_prints(str);

break;

}

d1 = (int) c - '0';

totalt = d1;

// next value needs to be a digit or a comma

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a number or ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

if (isdigit((int) c)){

d2 = (int) c - '0';

totalt = d1\*10 + d2;

if (totalt > 10){

usart\_prints("\r\nNumber is too large!");

break;

}

}

else comma = 1;

// if previous was not a comma, next needs to be a comma

if (comma==0){

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if ((!isdigit((int)c)) && (c!=',')){ // error check for a digit

sprintf(str, "\r\nExpected a ',' but got '%c'\r\n", c);

usart\_prints(str);

break;

}

}

// next value needs to be 1or 0

c = usart\_getc(); // get the character

usart\_putc(c); // echo back the character

if (c!='0' && c!='1'){ // error check for a digit

sprintf(str, "\r\nExpected 1 or 0 but got '%c'\r\n", c);

usart\_prints(str);

break;

}

totald = (int) c - '0';

/\*

Retrieve measurements from EEPROM and write to DAC

FORMAT: 'E:a,n,t,d'

a ... start address (integer, 0? a ? 510)

n ... number of measurements (integer, 1 ? n ? 20)

t ... time between measurements (integer, 1 ? dt ? 10 s)

d ... DAC channel number (integer, d is {0 or 1})

Note that stored 10-bit ADC values must be converted to the closest

8-bit integer value such that the DAC quantization error is minimal.

\*/

// implementation

for(int j = 0;j<totaln;j++){

unsigned int address = totala+(j\*2);

sprintf(str, "\n\rt=%d s, DAC Chann: %d , ",(j\*totalt), totald);

usart\_prints(str);

uint8\_t partA = EEPROM\_read(address);

uint8\_t partB = EEPROM\_read(address+1);

uint16\_t whole = ((uint16\_t)partA << 8) | partB;

float adcvalfloat = (float)whole\*5/1023;

sprintf(str, "V = '%.3f'", adcvalfloat);

usart\_prints(str);

setValue(totald, adcvalfloat);

for(int i = 0; i<totalt;i++){

\_delay\_ms(1000);

}

}

break; // end case

default:

usart\_prints("\r\n Expected 'M', 'S', 'R', or 'E'");

break; // break out of case (get new input)

}

usart\_prints("\r\n=====\r\n");

}

return(1);

}

ISR(USART\_RX\_vect)

{

// UART receive interrupt handler.

// To do: check and warn if buffer overflows.

char c = UDR0;

rx\_buffer[rx\_buffer\_head] = c;

if (rx\_buffer\_head == RX\_BUFFER\_SIZE - 1)

rx\_buffer\_head = 0;

else

rx\_buffer\_head++;

}

void usart\_init(void)

{

// Configures the USART for serial 8N1 with

// the Baud rate controlled by a #define.

unsigned short s;

// Set Baud rate, controlled with #define above.

s = (double)F\_CPU / (BAUD\_RATE\*16.0) - 1.0;

UBRR0H = (s & 0xFF00);

UBRR0L = (s & 0x00FF);

// Receive complete interrupt enable: RXCIE0

// Receiver & Transmitter enable: RXEN0,TXEN0

UCSR0B = (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0);

// Along with UCSZ02 bit in UCSR0B, set 8 bits

UCSR0C = (1<<UCSZ01)|(1<<UCSZ00)|(1<<UPM01)|(1<<USBS0);

DDRD |= (1<< 1); // PD0 is output (TX)

DDRD &= ~(1<< 0); // PD1 is input (Rx)

// Empty buffers

rx\_buffer\_head = 0;

rx\_buffer\_tail = 0;

}

void usart\_printf(const char \*ptr){

// Send NULL-terminated data from FLASH.

// Uses polling (and it blocks).

char c;

while(pgm\_read\_byte\_near(ptr)) {

c = pgm\_read\_byte\_near(ptr++);

usart\_putc(c);

}

}

void usart\_putc(const char c){

// Send "c" via the USART. Uses poling

// (and it blocks). Wait for UDRE0 to become

// set (=1), which indicates the UDR0 is empty

// and can accept the next character.

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

UDR0 = c;

}

void usart\_prints(const char \*ptr){

// Send NULL-terminated data from SRAM.

// Uses polling (and it blocks).

while(\*ptr) {

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

UDR0 = \*(ptr++);

}

}

unsigned char usart\_getc(void)

{

// Get char from the receiver buffer. This

// function blocks until a character arrives.

unsigned char c;

// Wait for a character in the buffer.

while (rx\_buffer\_tail == rx\_buffer\_head);

c = rx\_buffer[rx\_buffer\_tail];

if (rx\_buffer\_tail == RX\_BUFFER\_SIZE-1)

rx\_buffer\_tail = 0;

else

rx\_buffer\_tail++;

return c;

}

unsigned char uart\_buffer\_empty(void)

{

// Returns TRUE if receive buffer is empty.

return (rx\_buffer\_tail == rx\_buffer\_head);

}

uint16\_t adc\_read(void)

{

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

while(ADCSRA & (1<<ADSC)); // wait for conversion complete

float adcvalfloat = (float)ADC\*5/1023; // to store the float value of the ADC

char charadc[30]; // to store the character array for the ADC measurement

sprintf(charadc,"v = %.3f V",adcvalfloat); // merge the float value and characters into the character array

usart\_prints(charadc); // print the ADC measurement (format: 'v = X.XXX V')

return ADC;

}

void resetDAC(unsigned char channel) // Resets all DAC registers

{

i2c\_start\_wait(DevDAC+I2C\_WRITE);

i2c\_write(0b00010000+channel); // Send the RESET command

i2c\_stop();

}

void EEPROM\_write(unsigned int uiAddress, uint8\_t ucData)

{

/\* Wait for completion of previous write \*/

while(EECR & (1<<EEPE)) ;

/\* Set up address and Data Registers \*/

EEAR = uiAddress;

EEDR = ucData;

/\* Write logical one to EEMPE \*/

EECR = 0x04;

/\* Start eeprom write by setting EEPE \*/

EECR |= (1<<EEPE);

}

uint8\_t EEPROM\_read(unsigned int uiAddress)

{

/\* Wait for completion of previous write \*/

while(EECR & (1<<EEPE));

/\* Set up address register \*/

EEAR = uiAddress;

/\* Start eeprom read by writing EERE \*/

EECR |= (1<<EERE);

/\* Return data from Data Register \*/

return EEDR;

}

void setValue(unsigned char channel,float voltage)

{

unsigned char d;

float tmp;

voltage = voltage + 0.00;

tmp = (voltage\*100)\*(255.0);

tmp = tmp/500.0;

d = (unsigned char)(tmp+0.5);

i2c\_start\_wait(DevDAC+I2C\_WRITE); // Issue START and then send the address

i2c\_write(0x00+channel); // Select DAC0 or DAC1

i2c\_write(d); // Write data

i2c\_stop(); // Issue a STOP

}

1. **Appendix B: References**

Beichel, Reinhard. *Lab5 ECE: 3360 Embedded Systems, Spring 2019*. University of Iowa. 4/12/2019