CEEN 1060-003

Microprocessor Applications

Laboratory Assignment #11

Serial Peripheral Interface Applications

Austin B Koch

Collin Petersen

Riley Hester

Department of Computer and

Electronics Engineering

Peter Kiewit Institute

University of Nebraska

Due Date: 05/04/2015

**Introduction**

Serial Peripheral Interfaces are a vital part of our everyday means of communication. This lab serves as an introduction to processes that implement this technology. This lab allows students the opportunity to use the 1286 microcontroller to generate communication channels between itself and a PC through the manipulation of registers on the microcontroller as well as functions of AVR studio 4.

This lab report will provide background for the lab, followed by the procedure implemented. Results will be provided following the procedure, and then the additional questions provided will be answered. The report will then conclude. Attached will be the appendix, which will contain the code generated for this lab. This Lab took approximately seven hours to complete in total.

**Background**

For this lab, a UART RS232 Interface chip was connected to the PC with a DB9 connector. (Appendix 1) This allowed to the microcontroller the ability to receive and transmit data between itself and the computer. The microcontroller uses serial communication to send bytes to the computer in a half duplex configuration which signifies that it is capable of reception and transmission of data, but not at the same time.

In serial communication, in order for two platforms to be able to properly communicate, certain sets of rules, or a protocol, must be agreed upon which determines how data is packed, how many bits constitute a character, and when the data begins and ends. Another important factor is the bps (bits per second,) otherwise referred to as the baud rate, which determines the rate of information transfer.  
 To configure the microcontroller to use a SPI the UCSRA/B/C and UBRR registers must be manipulated. To use a faster baud rate, by setting the U2X bit on the UCSRA register, the baud rate will double. The process is different between transmission and reception so the user must be careful when configuring the AVR. It’s important to have code that verifies that one operation has been completed before moving on to the next.

**Procedure**

We began this lab by first setting up the peripheral devices using the provided diagram information (Appendix 1). After our circuit was set up we began Activity 1. Activity 1 had us configure a message in the code that would be sent to the computer’s terminal program. To do this we first began by configuring the USART register. We doubled the baud rate by setting the U2X1 bit on the UCSR1A register. We then set the UCSR1C register to 06H to declare zero parity and determine the character size and stop bit the device will use. The physical baud rate was then determined by setting the UBRR register to 000CH.

We then enabled the transmit and receive functions by manipulating the UCSR1B register. We created an array with the string we wanted to print, “Collin” in this case. A while loop was then set up to send the string to the computer terminal using the array and a 1ms delay to generate a reasonable print speed. The device was then activated and the terminal was monitored. Collin would print out in reasonably spaced out periods of time. We consulted the TA and got this activity signed off.

Next, for activity 2 we needed to use the terminal program on the PC to send a message to the microcontroller to print out the contents of the message on the LCD. We implemented our LCD code from previous labs to quicken the process. The program begins by initializing the LCD and then configures the USART and sets its information the same way from activity 1. Another array was instantiated to hold the contents of the sent message. A while loop is set up to serially read a received bit to an index of the array until the ‘\r’ character is read, (sent using a function of the terminal, which we failed to understand at first, setting us back a couple of hours,) at which point the microcontroller will print out the contents of the array to the LCD and repeat the process. Once we finished writing the program, we had the TA check off our work.

Activity 3 was a hybrid of the first two activities, and as such we used the same code to initialize and configure our SPI registers and LCD information. In addition the functions already implemented, we wrote read and write to EEPROM functions that would be used to take received data to write to the memory when obtained and to read written data upon the startup of the microcontroller.

After the configuring the LCD and USART, we set PORTE as an output. We created a do while loop to print out the data in EEPROM. After this is completed, the device waits for the SW1 switch to be pressed, upon which it enters a loop that will send a message to the computer that has been predetermined in the to\_Computer function. After it is sent, the program uses a new array to store the values being read from the computer’s terminal until the ‘\r’ character is read. The received data is then saved to EEPROM and the data is printed to the LCD. We got the TA to come sign off our operations and then cleaned up our area.

Activity 4 was not completed upon instruction due to time constraints.

**Results**

The results of this lab were fairly straightforward. Activity 1 resulted in the successful printing of the “Collin” string onto the Computer Terminal. We adjusted our delay to make the values print out a much more reasonable speed. Activity 2 also was successful. The message “kowabunga” was finally printed to the LCD after many hours of debugging. The main issue we faced was the configuration of the terminal itself. A box was unselected, so no ‘\r’ was printed by the computer so it our while loops became infinite.

Activity 3 went smoothly and the results were true to the objective. We were able to print “Hello world” after receiving the string from the computer. Upon reset the string remained printed to the LCD.

**Additional Questions**

**1. Are**

**2. The AVR microcontroller functions in digital values of high and low based on analogue values common to Transistor Transistor Logic (TTL,) which works with 0V for low and 3V for high. In order to work as an SPI, the device must convert those analogue values to values understood by the other components, where 3-25V serves as a low (Space in rs232,) and -3 – (-25)V serves as a high (mark in rs232).**

**3. 500000bps for 0 and 100000000 for bps**

**4. Baud Rate = Fosc/ 16(0+1), where Fosc = 8MHZ. When the bit is high the baud rate is doubled.**

**5. UDREn is the Data Registry empty bit that will set when the the transmit dta buffer is empty, signifying that it is ready to receive data. It can be used to generate an interrupt but primarily functions as a means to handshake communication channels.**

**RXC is the USART receive complete flag which will go high when data has been received by not read. Once the buffer is empty it is cleared. Also used for an interrupt, it primarily signifies that the read transaction of received data has been completed.**

**6. USART can buffer a single byte.**

**7. True**

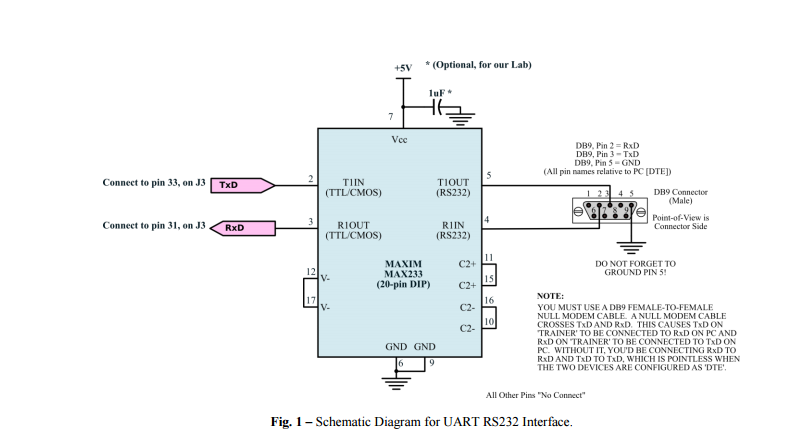
**8. 3. TxD, RxD, and Gnd**

**9. +/- 8V**

**10.+/- 30V**

**Conclusion**

This lab was intended to be a one week lab which could be completed in two weeks. For no justifiable reason were we unable to finish the lab within one week. We were prepared, had code written, and were earnest in our work. Despite this we had hiccups along the way that made us backtrack and debug for hours. Our biggest issue was the fact that we didn’t have the ‘\r’ box checked off on our terminal. This prevented us from finding any errors in our code as well as from moving forward since we had code that would complete the task asked of us in Activity 2. In the end we completed the lab within the first hour of the second week, including the time it took to set up the circuit. In the future it will be important to not only familiarize ourselves with the lab itself, but also the equipment being used.

**Appendix**

Appendix 1:

Appendix 2:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* USARTS: Serial Ports \*//

//\* Class: 1060-003, Spring 2015 \*//

//\* Description: using USARTS to \*//

//\* communicate serially \*//

//\* Lab 10, Activity 1 \*//

//\* Names: Austin Koch, Riley Hester, \*//

//\* Collin Peterson \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

#include<avr/io.h>

#include<stdint.h>

#include<avr/interrupt.h>

#include<util/delay.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void USART\_config(void);

void USART\_set(void);

void USART\_transmit(uint8\_t info);

uint8\_t USART\_receive(void);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void USART\_config(void)

{

UCSR1A = (1<<U2X1); // Doubles speed to reduce error

UCSR1B = 0x00; // Disabled interrupts

UCSR1C = 0x06; // No parity, character size, stop bit

UBRR1H = 0x00; // Set Baud Rate

UBRR1L = 0x0C;

}

void USART\_set(void)

{

UCSR1B |= (1<<TXEN1) | (1<<RXEN1); // Enables transmit and recieve

}

void USART\_transmit(uint8\_t info)

{

while(!(UCSR1A & (1<<UDRE1))); // waits the data register is empty

UDR1 = info; // loads UDR1 with passed info

}

uint8\_t USART\_receive(void)

{

uint8\_t info;

while(!(UCSR1A & (1<<RXC1))); // Waits until receive is complete

info = UDR1; // loads info from UDR1

return info; // returns info

}

void delay\_ms(unsigned int time)

{

\_delay\_ms(time); // Delays in milliseconds

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main (void)

{

USART\_config(); // configures USART

USART\_set(); // Sets TX & RX

unsigned char command[7] = "Collin\r", count = 0;

while(1)

{

USART\_transmit(command[count]);

if(count == 6)

{

count = 0;

delay\_ms(1000);

}

else

count++;

}

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* USARTS: Serial Ports \*//

//\* Class: 1060-003, Spring 2015 \*//

//\* Description: using USARTS to \*//

//\* communicate serially \*//

//\* Lab 11, Activity 2 \*//

//\* Names: Austin Koch, Riley Hester, \*//

//\* Collin Peterson \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

#include<avr/io.h>

#include<stdint.h>

#include<avr/interrupt.h>

#include<util/delay.h>

#define LCD\_DTport PORTB //\* Variables

#define LCD\_DTdirection DDRB //\*

#define LCD\_DTpin PINB //\*

#define LCD\_CMport PORTD //\*

#define LCD\_CMdirection DDRD //\*

#define LCD\_CMpin PIND //\*

#define LCD\_RW 4 //\*

#define LCD\_EN 5 //\*

#define LCD\_RS 6 //\* done defining variables

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void USART\_config(void);

void USART\_set(void);

void USART\_transmit(uint8\_t info);

uint8\_t USART\_receive(void);

void LCD\_init(void);

void LCD\_cmd(unsigned char cmd);

void LCD\_data(unsigned char data);

void LCD\_clear(void);

void delay\_ms(unsigned int time);

void delay\_us(unsigned int time);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void USART\_config(void)

{

UCSR1A = (1<<U2X1); // Doubles speed to reduce error

UCSR1B = 0x00; // Disabled interrupts

UCSR1C = 0x06; // No parity, character size, stop bit

UBRR1H = 0x00; // Set Baud Rate

UBRR1L = 0x0C;

}

void USART\_set(void)

{

UCSR1B |= (1<<TXEN1) | (1<<RXEN1); // Enables transmit and recieve

}

void USART\_transmit(uint8\_t info)

{

while(!(UCSR1A & (1<<UDRE1))); // waits the data register is empty

UDR1 = info; // loads UDR1 with passed info

}

uint8\_t USART\_receive(void)

{

uint8\_t info;

while(!(UCSR1A & (1<<RXC1))); // Waits until receive is complete

info = UDR1; // loads info from UDR1

return info; // returns info

}

void delay\_ms(unsigned int time)

{

\_delay\_ms(time); // Delays in milliseconds

}

void delay\_us(unsigned int time)

{

\_delay\_us(time); // Delays in microseconds

}

void LCD\_init() // Initializes LCD

{

LCD\_DTdirection = 0xFF;

LCD\_CMdirection = 0xFF;

LCD\_cmd(0x30);

delay\_ms(5);

LCD\_cmd(0x30);

delay\_ms(5);

LCD\_cmd(0x30);

delay\_ms(5);

LCD\_cmd(0x38);

delay\_ms(1);

LCD\_cmd(0x08);

delay\_ms(1);

LCD\_clear();

LCD\_cmd(0x06);

delay\_ms(1);

LCD\_cmd(0x0F);

delay\_ms(1);

}

void LCD\_cmd(unsigned char cmd) // Sends Commands

{

LCD\_DTport = cmd;

LCD\_CMport &= ~(1<<LCD\_RS);

LCD\_CMport &= ~(1<<LCD\_RW);

delay\_us(1);

LCD\_CMport |= (1<<LCD\_EN);

delay\_us(1);

LCD\_CMport &= ~(1<<LCD\_EN);

delay\_ms(1);

}

void LCD\_data(unsigned char data) // Sends data

{

LCD\_DTport = data;

LCD\_CMport |= (1<<LCD\_RS);

LCD\_CMport &= ~(1<<LCD\_RW);

delay\_us(1);

LCD\_CMport |= (1<<LCD\_EN);

delay\_us(1);

LCD\_CMport &= ~(1<<LCD\_EN);

delay\_ms(1);

}

void LCD\_clear () // Clears LCD

{

LCD\_cmd(0x01);

delay\_ms(15);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main (void)

{

LCD\_init();

USART\_config();

USART\_set();

unsigned char toWrite[16], count = 0x00;

while(1)

{

unsigned char toWrite[16], count1 = 0x00, count2 = 0x00;

while(1)

{

toWrite[count1] = USART\_receive();

if(toWrite[count1] == '\r')

{

LCD\_clear();

while(count2 < count1)

{

LCD\_data(toWrite[count2++]);

}

count2 = 0;

while(count2 < count1)

{

toWrite[count2++] = 0;

}

count2 = 0;

count1 = -1;

}

count1++;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* USARTS: Serial Ports \*//

//\* Class: 1060-003, Spring 2015 \*//

//\* Description: using USARTS to \*//

//\* communicate serially \*//

//\* Lab 10, Activity 2 \*//

//\* Names: Austin Koch, Riley Hester, \*//

//\* Collin Peterson \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

#include<avr/io.h>

#include<stdint.h>

#include<avr/interrupt.h>

#include<util/delay.h>

#define LCD\_DTprt PORTB //\* Variables

#define LCD\_DTdir DDRB //\*

#define LCD\_CMprt PORTD //\*

#define LCD\_CMdir DDRD //\*

#define SW\_prt PORTE //\*

#define SW\_dir DDRE //\*

#define LCD\_RW 4 //\*

#define LCD\_EN 5 //\*

#define LCD\_RS 6 //\* done defining variables

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void USART\_config(void);

void USART\_set(void);

void USART\_transmit(uint8\_t info);

uint8\_t USART\_receive(void);

void LCD\_init(void);

void LCD\_cmd(unsigned char cmd);

void LCD\_data(unsigned char data);

void LCD\_clear(void);

void delay\_ms(unsigned int time);

void delay\_us(unsigned int time);

uint8\_t EEPROM\_read(unsigned char address);

void EEPROM\_write(unsigned char address, unsigned char data);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ISR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

ISR(USART1\_RX\_vect)

{

LCD\_data(UDR1);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void USART\_config(void)

{

UCSR1A = (1<<U2X1); // Doubles speed to reduce error

UCSR1B = 0x00; // Disabled interrupts

UCSR1C = 0x06; // No parity, character size, stop bit

UBRR1H = 0x00; // Set Baud Rate

UBRR1L = 0x0C;

}

void USART\_set(void)

{

UCSR1B |= (1<<TXEN1) | (1<<RXEN1); // Enables transmit and recieve

}

void USART\_transmit(uint8\_t info)

{

while(!(UCSR1A & (1<<UDRE1))); // waits the data register is empty

UDR1 = info; // loads UDR1 with passed info

}

uint8\_t USART\_receive(void)

{

uint8\_t info;

while(!(UCSR1A & (1<<RXC1))); // Waits until receive is complete

info = UDR1; // loads info from UDR1

return info; // returns info

}

void delay\_ms(unsigned int time)

{

\_delay\_ms(time); // Delays in milliseconds

}

void delay\_us(unsigned int time)

{

\_delay\_us(time); // Delays in microseconds

}

void LCD\_init() // Initializes LCD

{

LCD\_DTdir = 0xFF;

LCD\_CMdir = 0xFF;

LCD\_cmd(0x30);

delay\_ms(5);

LCD\_cmd(0x30);

delay\_ms(5);

LCD\_cmd(0x30);

delay\_ms(5);

LCD\_cmd(0x38);

delay\_ms(1);

LCD\_cmd(0x08);

delay\_ms(1);

LCD\_clear();

LCD\_cmd(0x06);

delay\_ms(1);

LCD\_cmd(0x0F);

delay\_ms(1);

}

void LCD\_cmd(unsigned char cmd) // Sends Commands

{

LCD\_DTprt = cmd;

LCD\_CMprt &= ~(1<<LCD\_RS);

LCD\_CMprt &= ~(1<<LCD\_RW);

delay\_us(1);

LCD\_CMprt |= (1<<LCD\_EN);

delay\_us(1);

LCD\_CMprt &= ~(1<<LCD\_EN);

delay\_ms(1);

}

void LCD\_data(unsigned char data) // Sends data

{

LCD\_DTprt = data;

LCD\_CMprt |= (1<<LCD\_RS);

LCD\_CMprt &= ~(1<<LCD\_RW);

delay\_us(1);

LCD\_CMprt |= (1<<LCD\_EN);

delay\_us(1);

LCD\_CMprt &= ~(1<<LCD\_EN);

delay\_ms(1);

}

void LCD\_clear () // Clears LCD

{

LCD\_cmd(0x01);

delay\_ms(15);

}

void to\_Computer(unsigned char command[])

{

unsigned char count = 0;

while(command[count] != '\r')

{

USART\_transmit(command[count++]);

delay\_ms(1);

}

}

unsigned char EEPROM\_read(unsigned char address)

{

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

EEAR = address;

EECR |= (1<<EERE);

return EEDR;

}

void EEPROM\_write(unsigned char address, unsigned char data)

{

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

EEAR = address;

EEDR = data;

EECR |= (1<<EEMPE);

EECR |= (1<<EEPE);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main (void)

{

LCD\_init();

USART\_config();

USART\_set();

DDRE = 0;

unsigned char check = 0, c = 1;

unsigned char test = EEPROM\_read(0);

do

{

check = EEPROM\_read(c++);

LCD\_data(check);

delay\_ms(1);

}

while(c != test); //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* possible infinite read from EEPROM

while(1)

{

if(!(PINE & 1))

{

unsigned char command[16] = "Enter string!\r";

to\_Computer(command);

unsigned char c1 = 0, toWrite[16];

do

{

toWrite[c1] = USART\_receive(); // Stores USART data in toWrite

}

while(toWrite[c1++] != '\r');

EEPROM\_write(0, c1);

LCD\_clear();

unsigned char c2 = 0; // Clears LCD

for(; c2 < c1 - 1; c2++) // <+ ?

{

LCD\_data(toWrite[c2]); // Writes toWrite to LCD

EEPROM\_write(c2 + 1, toWrite[c2]);

toWrite[c2] = 0;

delay\_ms(1);

}

// EEPROM\_write(c2, 0x0D);

delay\_ms(500);

while(!(PINE & 1));

}

}

}