Serial communication system

Embedded Software Design

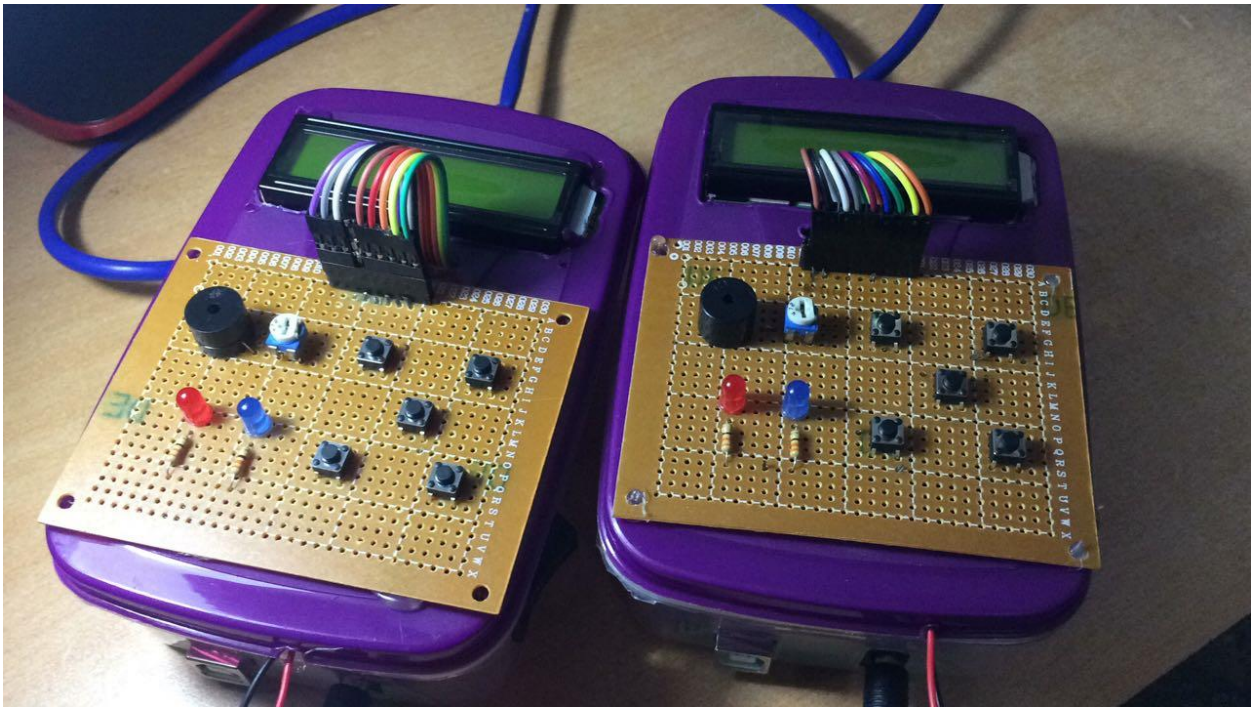
Project Report

Year 2 – Semester 2

Name	: S.M.A.M.Manchanayake
Reg. No	: IT 16058088
Name	: L.P.J.Perera
Reg. No	: IT 16084896
Date of Submission	: 2017/11/20

Introduction

In this project we are expected to design and build a serial communication system using ATmega328P microcontroller. The ATmega328P microcontroller include serial communications capability.
On the ATmega328P it's called the Universal Synchronous and Asynchronous serial Receiver and Transmitter. (USART)
There are two stations with Arduino UNO boards running identical code.



Theory

Serial Transmission

In telecommunication and data transmission, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus.

Serial Ports

- Used to transfer data to devices such as modems, terminals and various peripherals.
- Uses standard DB9 connector.
- A serial interface needs just two wires: one for sending data and another for receiving.

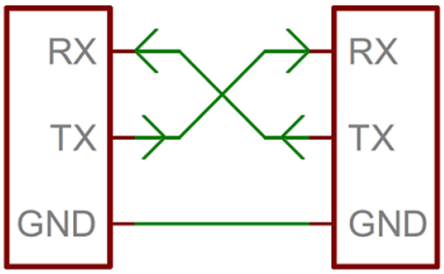


Figure 4.0

- Two serial pins: the receiver, RX, and the transmitter, TX.
- Single wise these two lines are Uni-directional.
- Devices are divided into two categories:
    1. Data terminal equipment (DTE)
    2. Data circuit-terminating equipment (DCE).
- A DCE device can be connected to a DTE device with a straight wired cable.
- Computers and terminals are DTE while modems and peripherals are DCE.
- If it is necessary to connect two DTE devices (or two DCE devices) , a cross-over null modem, in the form of either an adapter or a cable, must be used.

Serial communication in the microcontroller

The serial port on the microcontrollers uses two pins on the chip, one for sending data (transmitter) and the other for receiving data. The interface is called an "asynchronous" interface since it does not have any separate clock signal. Only the data is sent on the lines. The transmitter must send the data at an agreed upon rate and in a defined manner. Ones the receiver sees the start of incoming data, it samples the incoming signal at fixed intervals to determine whether a zero or one is present.
If the two devices are not set to use the same signaling rate, or one does not follow the same communications protocol as other, the signal will not be received properly.

Baud Rate

"Baud rate" is a historical term for the data rate in bits/second used in the communications link. Serial devices can usually communicate their data at any rate up to the maximum the hardware will allow as long as both are using the same rate. However many serial devices only support a small number of traditional baud rates when communicating with these it is necessary to use one of the rates they support. Some of the more common baud rates are 300,1200,2400,4800,9600,14400,19200,28800,38400,57600,115200.

However in many cases the rate will be determined by the other device. For example an LCD display with a serial interface may specify that it communicates at 4800 baud. In that case that rate would have to be used by the microcontroller.

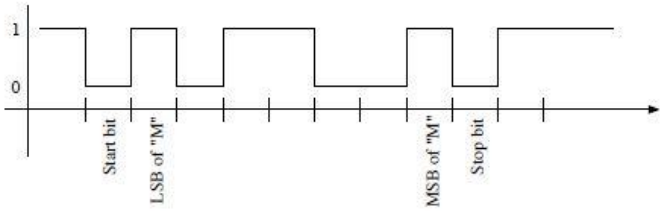


Figure 1: Sending the seven bit ASCII code for "M" (1001101)

Asynchronous Serial Protocol

The protocol used by the microcontrollers to send bits over an asynchronous link is well defined and must be followed by the devices on both ends of the link. In order to correctly communicate the data, the following things must be agreed upon by both devices.

Baud rate, Number of data bits, Parity bits, Number of stop bits, Start bit, Data bit, Parity bit, Stop bits.

Choosing a serial interface to use depends on several factors,

- What interface is available on the device you need to talk to.
- Speed
- Distance between devices
- Cost of wiring and connectors
- Complexity of software
- Reliability

Serial interfaces

1. I^2c Interface (Inter-Integrated Circuit Interface)

    - Also known as the “Two Wire Interface” (TWI)
      Clock is generated by the master device.
      Data line is bi-directional.

    - Most commonly used on a single PC board to transfer data between two or more ICs.
    - Data rates are relatively slow (usually < 100kb/sec)
    - Half duplex

2. SPI Interface (Serial Peripheral Interface Bus)

    - Users four wires. (three in many cases)
    - Full duplex

3. 1-Wire Interface

        - Uses a single wire to send data in both directions.
        - No clock. Timing based on length of the high and low states of the data line.
        - Used for communicating to low speed devices.

4. RS-232 Interface

    - A “legacy” serial interface developed in the 1960’s.
    - Still in wide-spread use due to it’s simplicity.
    - Uses minimum of three wires
    - It is an Asynchronous interface because it does not use a clock signal.

Equipment used:

- Two Arduino ATmega328P microcontrollers.
- Two 16*2 LCD screens
- Two buzzers
- Four LEDs
- Jumper cables (Male-to-Male, Female-to-Female, Male-to-Female)
- Ten buttons
- Two 5k Variable Resistors
- Two 2k Variable Resistors
- Two switches
- Two RJ-45 adapters
- Two 9V batteries
- One LAN cable

Main operations and the functions of the project

We have set our Baud Rate on both sides to 9600 bits/second.

1. Programs starts with the Main()……

```
int main(void){
   USART_Init(MYUBRR);
        init_lcd();
        _delay_ms(15);
        writecommand(0x01);
        initKeypad();
        sei();

   while(1){
                keyPress();

                if(flag==1){
                        recieveIndication();
                        flag=0;
                        k=0;
                        writecommand(0x01);
                        stringout(msg);
                        clearbuff();
                        SendString("?");
                }
                else if(flag==2){//react for ack
                        flag=0;
                        k=0;
                        deliverIndication();
                }
        }
}
```

- USART_Init(MYUBRR);

```
void USART_Init( unsigned int ubrr) /* SETUP UART */
{
UBRR0 = MYUBRR;                         /*Set baud rate */
UCSR0B |= (1 << RXCIE0);                //enable interrupt when start receiving data
UCSR0B |= (1 << TXEN0 | 1 << RXEN0);/*Enable receiver and transmitter */
UCSR0C = (3 << UCSZ00);                  //Asyn, No parity,1 Stop bit,8 Data bits
}
```

In USART_Init(MYUBRR) we first sets our Baud rate by assigning the MYUBRR value to the Register UBRRO.
Then in the UCSROB register we are enabling an interrupt that will occur when data is starts to receive.
Then we are enabling the transmitter and the receiver in the UCSROB register.
Finally we are setting the Frame format which is Asynchronous, No parity bits, 1 Stop bits and 8 Data bits.

- init_lcd();

This function is used to initialize the LCD.
A register select (RS) signal will determines which register is the destination of the data we send.

IF RS = 1
    Info goes to the data register
IF RS = 0
    Info goes to the command register

Command register (Clear LCD(0x01), move cursor, turn display on or off) write command code to the command register.

Data register (display characters on the screen) write the ASCII code for the character to the data register.

- initKeypad();

Then we have initialize our keyboard consist of **five buttons** when we power on our Arduino board.
We are using five pins from the PORTB as inputs so we are assigning them as inputs by using the **data direction register (DDRB),** and as we are using **inputs** we are also have to **enable the internal pull up resistors.**

- Sei();

We are enabling the **global interrupts** using the command Sei().

After all these are initialize once at the beginning of the code execution, While loop will start to run.

Inside the while loop first function that we are invoking is,

- keyPress();

First we see whether any of the five buttons connected to the PORTB is press or not (if ((PINB & 0x1f) != 0x1f)).
Move to the second row because what we are typing should be printed in the second row. (moveto (1,0))
PB1 = 'a'
PB2 = 'b'
PB3 = 'c'
PB0 = backspace
PB 4 = Send Button

When sending (PB4) button is used, we are adding a '.' to the end of the message that will indicate the end of the word to the receiving party.

When data starts to receive it will jump to the ISR(USART_RX_vect) interrupt service routine because we have enable it at the very beginning.
It will store the receiving message to the msg[] array.

Flag = 0 means still the message is coming.

Flag = 1 means it is the end of the word, and it is captured by the '.' mark we have added when sending to identify the end of the word.

After message have been successfully received we will send an acknowledgement to the transmitter, that is done by '?' mark so,
Flag == 2 means react for acknowledgement by lighting up the LED in the transmitters side.

Overall these are the main functions that are executing in the Main.

Extra Features Used:

1. recieveIndication();

At first when the receiver sees the '.' Sign that means the end of the word and the Flag will be set to 1, which will call the receive indication function which will ring the buzzer two times so we know that has been received successfully.

2. deliverIndication();

We will send an acknowledgment to the transmitter using '?'. So when Flag ==2 that means we have to react for the acknowledgment, and it will call the deliver Indication function which will blink the LED two times, so we know that acknowledgment also has been received successfully.

3. We have used a LAN cable to connect the two stations, and also two RJ-45 adapters.

Identical code used for two stations which are having two arduino boards,

```
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <string.h>

/* UART SERIAL DEFINES */
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1
#define FOSC 16000000 // Clock frequency

void USART_Init( unsigned int ubrr);
void USART_Transmit( unsigned char data );
unsigned char USART_Receive( void );
void SendString(char *StringPtr);
void RecieveString(void);
void clearbuff(void);
void init_lcd();
void moveto(unsigned char row, unsigned char col);
void stringout(char *str);
void writecommand(unsigned char cmd);
void writedata(unsigned char dat);
void writenibble(unsigned char lcdbits);
void initKeypad(void);
void keyPress();
void recieveIndication();
void deliverIndication();
```

```
char input[16];
char output[15];
char msg[16];
int counter;
int k=0;
int outputPointer = 0;
int flag=0;


void timer0_init()
{
  //TCCR0 |= (1 << CS00);// set up timer with no prescaling
  //TCNT0 = 0;// initialize counter
}
void init_timer1(unsigned short m)
{
        TCCR1B |= (0b00001101);
        TIMSK1 |= (0X02);
        OCR1A = m;
 }


void USART_Init( unsigned int ubrr) /* SETUP UART */
{
        UBRR0 = MYUBRR; /*Set baud rate */
        UCSR0B |= (1 << RXCIE0); //enable interrupt when start receiving data
        UCSR0B |= (1 << TXEN0 | 1 << RXEN0); /*Enable receiver and transmitter */
        UCSR0C = (3 << UCSZ00); /* Set frame format */ //Asyn,No parity,1 Stop
        bit,8 Data bits
}
void USART_Transmit( unsigned char data )
{
        // Wait for transmitter data register empty
        while ((UCSR0A & (1<<UDRE0)) == 0) {}
        UDR0 = data;
}
```

```
unsigned char USART_Receive( void )
{
	// Wait for receive complete flag to go high
	while ( !(UCSR0A & (1 << RXC0)) ) {}
	return UDR0;
}
void SendString(char *StringPtr)
{
	while(*StringPtr != 0x00)
	{
		USART_Transmit(*StringPtr);
		StringPtr++;
	}
}
void RecieveString(void)
{
	int i;
	for(i=0;i<16;i++)
	{
		input[i]=USART_Receive();
	}
}
void clearbuff(void)
{
	int i;
	for(i=0;i<16;i++)
	{
		msg[i]=' ';
	}
}
```

```
void init_lcd()
{
   // Set the DDR register bits for ports B and D
        DDRC|=0x0f;
        DDRD|=0x0c;
   _delay_ms(15);// Delay at least 15ms

         // Use writenibble to send 0011
        writecommand(0x03);
   _delay_ms(5);// Delay at least 4msec

        // Use writenibble to send 0011
        writecommand(0x03);
   _delay_us(105);// Delay at least 100usec

    // Use writenibble to send 0011, no delay needed
        writecommand(0x03);

        // Use writenibble to send 0010
        writecommand(0x02);
   _delay_ms(2); // Delay at least 2ms

   writecommand(0x28);        // Function Set: 4-bit interface, 2 lines
        _delay_ms(2);

        writecommand(0x0f);        // Display and cursor on
        _delay_ms(25);

        writecommand(0x01);
}
```

```
/*moveto - Move the cursor to the row (0 or 1) and column (0 to 15) specified*/
void moveto(unsigned char row, unsigned char col)
{
	if(row==0)
	{
		writecommand(0x80+col);
	}
	if(row==1)
	{
		writecommand(0xc0+col);
	}
}
/*stringout - Write the string pointed to by "str" at the current position*/
void stringout(char *str)
{
	do{
		writedata(*str);
		str++;
	}while(*str!= '\0');
}
/*writecommand - Send the 8-bit byte "cmd" to the LCD command register*/

void writecommand(unsigned char cmd)
{
	unsigned char temp;

	PORTD&=~(0x04);
	temp=cmd&0xF0;
	temp=temp>>4;
	writenibble(temp);
	temp=cmd&0x0F;
	writenibble(temp);
	_delay_ms(3);
}
```

```
/*writedata - Send the 8-bit byte "dat" to the LCD data register*/


void writedata(unsigned char dat)
{
	unsigned char temp;

	PORTD|=0x04;
	temp=dat&0xF0;
	temp=temp>>4;
	writenibble(temp);
	temp=dat&0x0F;
	writenibble(temp);
	_delay_ms(3);
}
/*writenibble - Send four bits of the byte "lcdbits" to the LCD*/

void writenibble(unsigned char lcdbits)
{
	PORTC = lcdbits;
	PORTD &= ~(0x08);
	PORTD |= 0x08;
	PORTD &= ~(0x08);
}

void initKeypad(void)
{
	DDRD |= 0xe0;
	DDRB &= ~(0x1f);
	PORTB |= 0x1f;
	PORTD |= 0x40;
}
```

```
void keyPress()
{
	if((PINB & 0x1f)!=0x1f){
		_delay_ms(5);
		moveto(1,0);

		if(!(PINB & (1<<PB1))){
			_delay_ms(5);
			while(!(PINB & (1<<PB1))){}
			output[outputPointer]='a';
		}
		else if (!(PINB & (1<<PB2))){
			_delay_ms(5);
			while(!(PINB & (1<<PB2))){}
			output[outputPointer]='b';
		}
		else if (!(PINB & (1<<PB3))){
			_delay_ms(5);
			while(!(PINB & (1<<PB3))){}
			output[outputPointer]='c';
		}
		else if (!(PINB & (1<<PB0))){
			_delay_ms(5);
			while(!(PINB & (1<<PB0))){}
			outputPointer--;
			output[outputPointer]=' ';
			if(outputPointer>=0)
				outputPointer--;
		}
		else if (!(PINB & (1<<PB4))){
			_delay_ms(5);
			while(!(PINB & (1<<PB4))){}
			output[outputPointer]='.';
			SendString(output);
		}
		outputPointer++;
		stringout(output);}}
```

```
void recieveIndication()
{
	PORTD |= 0x20;//small buzzer indication
	_delay_ms(70);
	PORTD &= ~(0x20);
	_delay_ms(70);
	PORTD |= 0x20;//small buzzer indication
	_delay_ms(70);
	PORTD &= ~(0x20);
}

void deliverIndication()
{
	PORTD |= 0x80;
	_delay_ms(70);
	PORTD &= ~(0x80);
	_delay_ms(70);
	PORTD |= 0x80;
	_delay_ms(70);
	PORTD &= ~(0x80);
}
```

```
int main(void)
{
   USART_Init(MYUBRR);
        init_lcd();
        _delay_ms(15);
        writecommand(0x01);
        initKeypad();
        sei();

   while(1){
                keyPress();

                if(flag==1){
                        recieveIndication();
                        flag=0;
                        k=0;
                        writecommand(0x01);
                        stringout(msg);
                        clearbuff();
                        SendString("?");
                }
                else if(flag==2){//react for ack
                        flag=0;
                        k=0;
                        deliverIndication();
                }
        }
}
```

```
ISR(USART_RX_vect)
{

        msg[k++]=USART_Receive();
        if(msg[k-1]=='?')//check for ack
                flag=2;
        else if((msg[k-1]=='.')||(k==15))
                flag=1; // . is the end of the word. one word has been received
        else
                flag=0; // still the msg is coming
}
```