Lab 7: AVR Serial Communication

EE222: Microprocessor Systems

Contents

1 Administrivia

1.1 Learning Outcomes

By the end of this lab you will be able to:

1. Setup ATmega16A microcontroller for UART communication.
2. Send and receive serial data from ATmega16A.
3. Setup serial link between your PC and ATmega16A microcontroller through UART interface.

1.2 Deliverables

You are required to submit:

- Code
- Observations and experiences

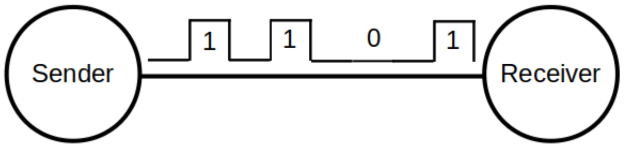
in the beginning of next lab.

2 Hardware Resources

- ATmega16A Microcontroller Unit
- Universal Programmer
- Seven Segment Display
- Resistance 47Ω
- LEDs (may use from trainer kit)
- USB-TTL Converter.

3 Serial Communication

Computers either communicate in a serial manner or parallel manner. Each mode has its own merits and demerits. In serial communication, data is sent bit by bit through a single wire.



Serial communication can be "Synchronous" and "Asynchronous". In synchronous communication, the sender sends a clock along-with the data (through another wire) to tell the receiver that valid data bit will be available on each positive/negative edge of the clock. In asynchronous communication, the sender and receiver agrees upon a time unit after which the bit transferred or received is considered valid. In this lab we will learn about asynchronous serial communication.

3.1 Universal Asynchronous Receiver Transmitter (UART)

UART is a block of circuitry in microcontrollers responsible for implementing asynchronous serial communication. In UART there are two pins "Rx" and "Tx". Rx is responsible for data reception while Tx is responsible for data transmission. This mode of serial communication is also called **"TTL serial"** because in this mode data is transferred on `0-5V or 0-3.3V` voltage levels (Transistor-Transistor Logic).

3.1.1 Baud Rate

In digital communication, number of bits per second is termed as baud rate. In UART communication, both, the sender and the receiver agrees upon a common baud rate at which data will be transferred between them.

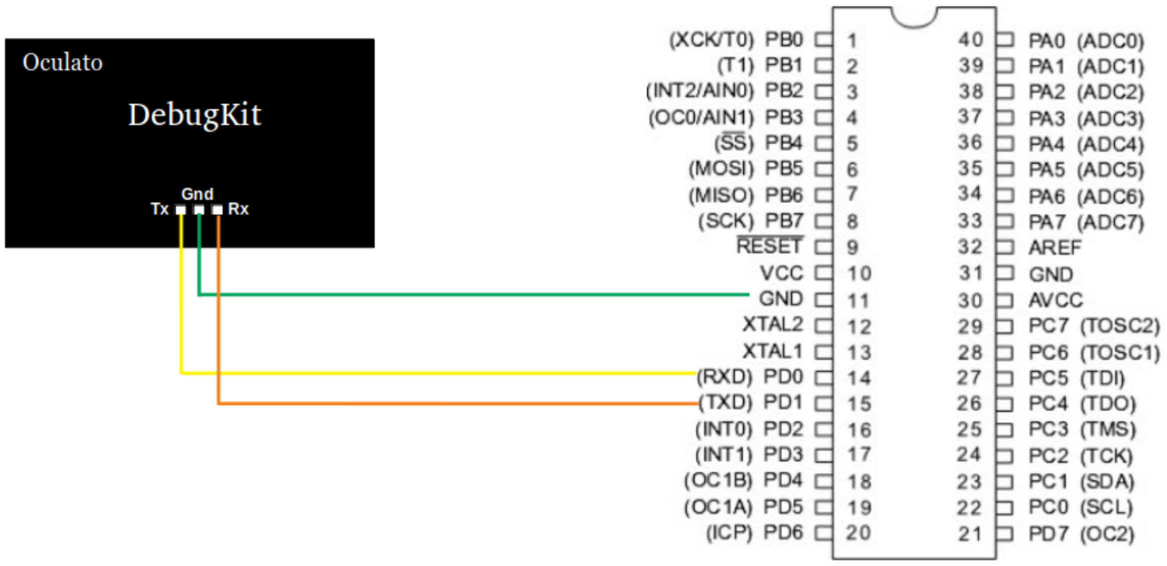
3.2 PC Communication

In modern computers, serial communication is standardized by industry through **USB (Universal Serial Bus)** standard for computers to communicate with their peripheral devices.

3.2.1 Connecting PC with Microcontroller

The Tx and Rx pins on ATmega16A side uses TTL volatge levels while the interface on PC side is USB. Therefore, we need an intermediate device that converts TTL standards to USB.

In our lab, Oculato UART Debug kit is available which in fact is a USB-to-TTL converter. The following pin configuration should be followed while using this equipment.



3.2.2 Software

TeraTerm is an open-source terminal emulator for serial communication. You can download the software from here[1]. As you launch teraterm, a black screen will appear on your display. Configure teraterm using this tutorial[2]. Whatever your microcontroller sends through UART will appear on this screen and whatever you'll type on this screen will be sent by teraterm to your microcontroller.

[1]https://tera-term.en.lo4d.com/

[2]https://github.com/Uthmanhere/EE222/blob/master/teraterm$_t$*utorial.pdf*

3.3 Transmitting data over UART

The following code continuously sends “OK” to PC over UART.

```c
1  #define F_CPU 1000000
2  #include <avr/io.h>
3  #include <util/delay.h>
4  
5  //function to initialize UART communication
6  void UART_initializer()
7  {
8    UBRRL = 0xC;           // set baud rate to 4800
9    UCSRB |= (1<<TXEN);     // enable transmitter
10   UCSRC |= (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1); // set data size
11 }
12 //function to transmit data over UART
13 void UART_transmitter()
14 {
15   char msg[] = {'\n', '\r', 'O', 'K', '\n', '\r'};
16   char size_msg = 6;
17   unsigned char i;
18   for(i = 0; i < size_msg; i=i+1)
19   {
20     while( !( UCSRA & (1<<UDRE) ) );
21     UDR = msg[i];
22   }
23 }
24 int main(void)
25 {
26   UART_initializer(); //initialize UART
27   while(1)
28   {
29     UART_transmitter(); //transmit "OK"
30     _delay_ms(500);
31   }
32 }
```

Explanation

- Configure UART Communication protocol
  - Line 8: Baud rate is being set to 4800 bps as per following calculations

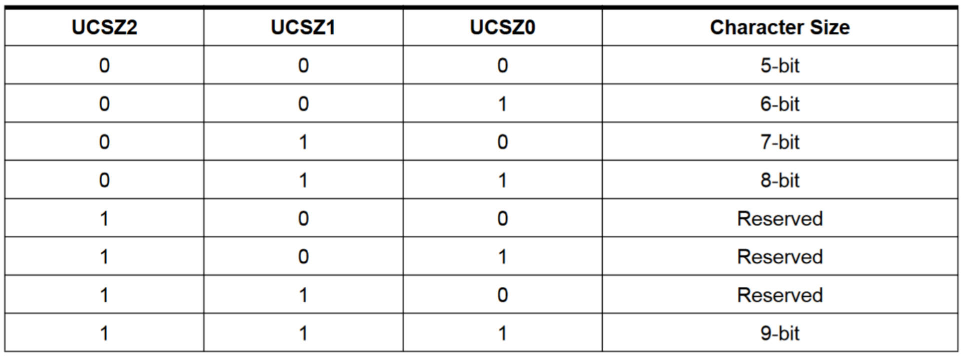
$$UBRR = \frac{f_{OSC}}{16 \times BAUD} - 1$$

for us, $f_{OSC} = 1MHz$ and assuming the required BAUD rate is $4800bps$, the value of `UBRR` equates to

$$UBRR = \frac{10^6}{16 \times 4800} - 1 \simeq 12$$

where the hexadecimal equivalent of 12 is C_{16}

  - Line 9: Enable transmission by raising `TXEN` bit in `UCSRB` (UART Control Status Register B).
  - Line 10: In `UCSRC` raising `URSEL` to high while setting up `UCSZ2:UCSZ0` to 011_2 which leads to 8 bit data chunk size for data transfer as the following table referred from datasheet suggests.



UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- Transmit over UART communication protocol
  - Line 15: Initialize array.
    - **\r** Carriage return: Move the cursor to beginning of the line.
    - **\n** Newline: Move the cursor to next line.
  - Line 20: Poll over UART Data Register Empty flag `(UDRE)` in `UCSRA` to prepare UART Data Register `(UDR)` for transferring next data packet.
- Main: The over-all functionality of this code is to initialize UART and send data to PC every half a second.

3.4 Receiving data over UART

The following code assumes LEDs are connected to each pin of PORTB. It receives data from UART and turns on the respective LED. For example, if character '5' is sent, the fifth LED will lit up. The program will ignore characters sent other than digits from `0-7`.

```
1  #define F_CPU 1000000
2  #include <avr/io.h>
3  #include <util/delay.h>
4  //function to initialize UART communication
5  void UART_initializer()
6  {
7    UBRRL = 0xC;   // set Baud Rate to 4800
8    UCSRB |= (1<<RXEN); // enable receiver
9    UCSRC |= (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1); // set data size
10 }
11 //function to receive data over UART
12 void UART_receive()
13 {
14   while( !( UCSRA & (1<<UDRE) ) ); //poll UDRE and wait to receive data
15   char val = UDR;
16   if(val >= '0' && val < '8') //check the ASCII sent and
17       PORTB = 1 << (val - '0'); //turn on the respective LED
18 }
19 int main(void)
20 {
21   DDRB = 0XFF;    //initial configuration for PORTB
22   PORTB = 0xFF;
23   UART_initializer(); //initialize UART
24   while(1)
25   {
26       UART_receive(); //receive data
27   }
28 }
```

Explanation This piece of code is not much different from the one we implemented in Task A. Following differences have been highlighted.

- Line 8: Receive `RXEN` is enabled instead of transfer.
- Line 16,17: Evaluates digit out of ASCII using expression `(val - ‘0’)` and set the particular bit high.

4 Lab Tasks

4.1 Task A

Estabilish connection between your ATmega16 and PC through UART-to-TTL converter. Run the programs in section 3.3 and 3.4 and explain your observations through screenshots.

4.2 Task B

Connect a 7-segment-display with your ATmega16. Estabilish connection between your ATmega16 and PC through UART-to-TTL converter. Write a program in C such that;

- Initially, ZERO should be displayed on the 7-segment and the program should wait for the next instruction from PC.
- If a 'u' is sent from PC to ATmega16, the next digit on seven segment should be displayed in ascending order.
  Ascending order (`0,1,2,3,...,8,9,0,1,...`)
- If a 'd' is sent from PC to ATmega16, the next digit on seven segment should be displayed in descending order.
  Descending order (`9,8,7,...,2,1,0,9,...`)
- If any character other than 'u' or 'd' is sent, do nothing with 7-segment-display and send "ERR" back to PC in order to notify error.

Implement the above functionality using **AVR SERIAL INTERRUPTS**.