Department of Computer Engineering
Faculty of Engineering, University of Peradeniya

CO321 Embedded Systems - 2020

Lab 6 - Serial Communication

AVR Serial Port Programming in C Language

Basics of serial communication

When a microprocessor communicates with the outside world, it provides the data in byte-sized chunks. For some devices, such as printers, the information is simply grabbed from the 8-bit data bus and presented to the 8-bit data bus of the device. This can work only if the cable is not too long, because long cables diminish and even distort the signals. Furthermore, an 8-bit data path is expensive. For these reasons, serial communication is used for transferring data between two systems located at distances of hundreds of feet to millions miles apart.

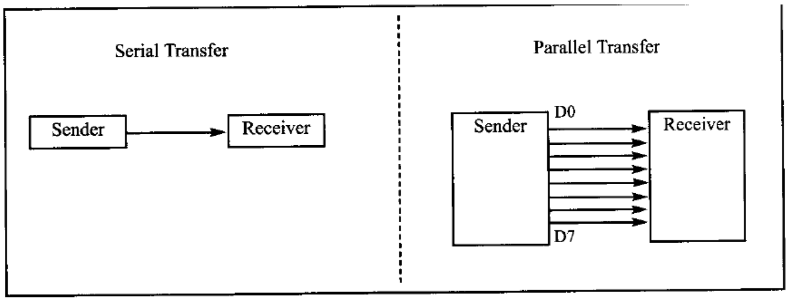

Figure 01: Serial versus Parallel data transfer

For the serial data communication to work, the byte of data must be converted to serial bits using a parallel-in-serial-out shift register; then it can be transmitted over a single data line. This also means that at the receiving end there must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte.

Serial data communication uses two methods, asynchronous and synchronous. The synchronous method needs to transmit the clock as a separate wire while it is not so in asynchronous transmission. It is possible to write software to use either of these methods, but the programs can be tedious and long. For this reason, special IC chips are made by many manufacturers for serial data communications. These chips are commonly referred to as USART (Universal Synchronous-Asynchronous Receiver-Transmitter).

Data transfer rate

The rate of data transfer in serial data communication is stated in bps. Another widely used terminology for bps is baud rate. Baud rate is defined as the number of signal changes per second. The data transfer rate of a given computer system depends on communication ports incorporated into that system

USART in ATmega328P

USART(universal synchronous asynchronous receiver/transmitter) in Atmega328P can work in both synchronous mode and asynchronous mode. The asynchronous mode is the one we will use to connect the AVR-based system to the PC serial portfor the purpose of full-duplex serial data transfer.

RX and TX pins in the ATmega328P

The ATmega32 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TX and RX and are part of the Port D group (PD0 and PD1). These pins are TTL compatible; therefore, they require a line driver such as MAX232 to make them RS232 compatible if you are connecting to the RS232 serial port of a computer. But we are going to connect them via a USB to RS232 converter to the USB port of the computer. The converter will convert the signal to/from RS232 protocols to USB protocol.

UBRR (USART Baud rate Register) and baud rate in the AVR

The baud rate in the AVR is programmable. This is done with the help of the 16-bit register (two 8 bit registers) called UBRR that decides the baud rate. Only 12 bits are used to set the baud rate.

UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

The relation between the value loaded into UBBR and the Fosc (frequency of oscillator connected to the XTAL1 and XTAL2 pins) is dictated by the following formula where X is the value to be loaded to UVRR register.

$$\text{Desired baud Rate} = \frac{\text{Fosc}}{(16\,(X+1))}$$

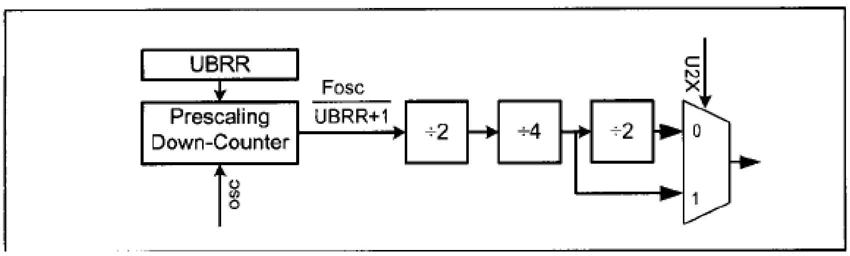

Figure 02: Baud rate generation block diagram

As shown in Fig 02 UBRR is connected to a down-counter, which functions as a programmable prescaler to generate baud rate. The system clock (Fosc) is the clock input to the down-counter. The down-counter is loaded with the UBRR value each time it counts down to zero. When the counter reaches zero, a clock is generated. This makes a frequency divider that divides the OSC frequency by UBRR + 1. Then the frequency is divided by 2, 4, and 2.

UDR0(USART Data Register) registers and USART data I/O in the AVR

In the AVR, to provide a full-duplex serial communication, there are two shift registers referred to as Transmit Shift Register and Receive Shift Register. Each shift register has a buffer that is connected to it directly. These buffers are called Transmit Data Buffer register and Receive Data Buffer Register. The USART Transmit Data Register and USART Receive Data Buffer Register share the same I/O address, which is called USART Data Register or UDR. When you write data to UDR, it will be transferred to the Transmit Data Buffer Register (TXB), and when you read data from UDR, it will return the contents of the Receive Data Buffer Register (RXB). See Fig 03.

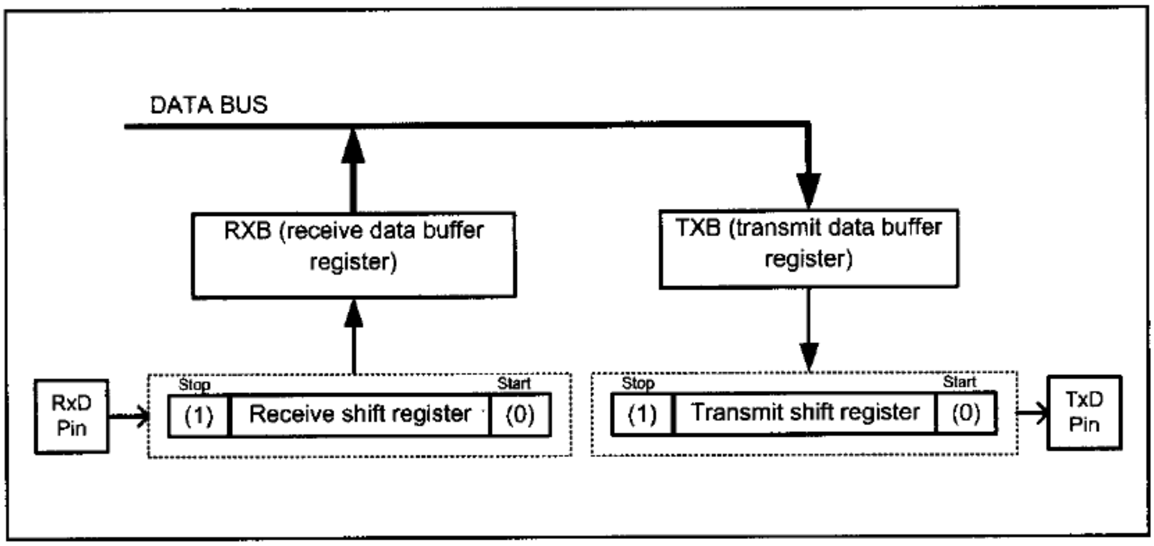

Fig 03: Simplified USART block diagram

UCSR0 registers (USART Control Status Registers) and USART configurations in the AVR

UCSR0s are 8-bit control registers used for controlling serial communication in the AVR. There are three USART Control Status Registers in the AVR. They are UCSR0A, UCSR0B, and UCSR0C.

Programming the AVR to transfer data serially (polling method)

1. Enable the USART transmitter (See TXEN0 bit in UCSR0B). The transmitter will override normal port operation for the TxD pin when enabled.
2. Select asynchronous mode. We will use 8-bit data frame, no parity, and one stop bit. (See UCSR0C register)
3. The UBRR0 is loaded to set the baud rate for serial data transfer. (Select a baud rate supported in the serial tool you use on your PC and accordingly calculate the value)
4. The character byte to be transmitted serially is written into the UDR0 register.
5. Monitor the UDRE0 bit of the UCSR0A register to make sure UDR0 is ready for the next byte.
6. To transmit the next character, go to step 4.

Programming the AVR to receive data serially (polling method)

1. Enable the USART receiver (See RXEN0 bit in UCSR0B). The receiver will override normal port operation for the RxD pin when enabled.
2. Select asynchronous mode. We will use 8-bit data frame, no parity, and one stop bit. (See UCSR0C register)
3. The UBRR0 is loaded to set the baud rate for serial data transfer. (Select a baud rate supported in the serial tool you use on your PC and accordingly calculate the value)
4. The RXC0 flag bit of the UCSR0A register is monitored for a HIGH to see if an entire character has been received yet.
5. When RXC0 is raised, the UDR register has the byte.
6. To receive the next character, go to step 5.

Exercises:

1) Write a C function called `usart_init` to initialize the USART to work at 9600 baud, 8-bit data, no parity and 1 stop bit.
   Write a function called `usart_send` to transmit a character given as the argument.
   Write a function called `usart_receive` that returns a received character from the USART.
   (No need to show these, but this will make your life easy in the next exercises)

2) Program the AVR in C to transmit the names and E-numbers of your group mates.

3) Write a program that repeatedly does the following
   a. Receives a sentence only having ASCII characters from the PC. You can use the carriage return character to find the end of the sentence.
   b. Apply Caesar cipher (key is 3) to the sentence on the microcontroller. Only encrypt alphabetic characters (both upper and lower case). Keep all other characters unchanged.
   c. Transmit the encrypted sentence back to the PC.