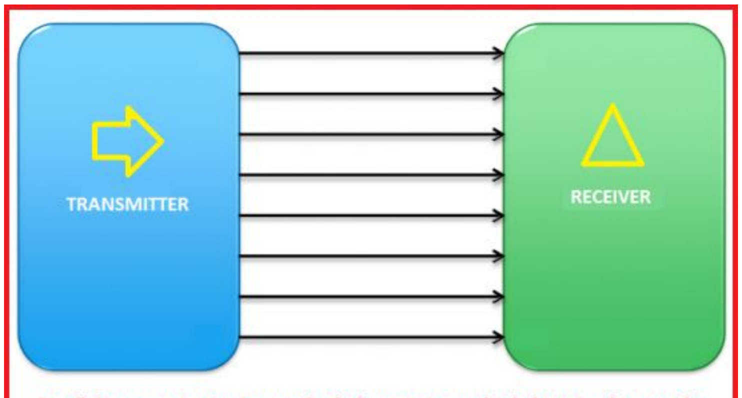
Lab 9: Asynchronous Serial Data Communication with an AVR

CompEng 3151: Digital Engineering Lab II

Last revised: July 16, 2019 (BJZ, RJS)

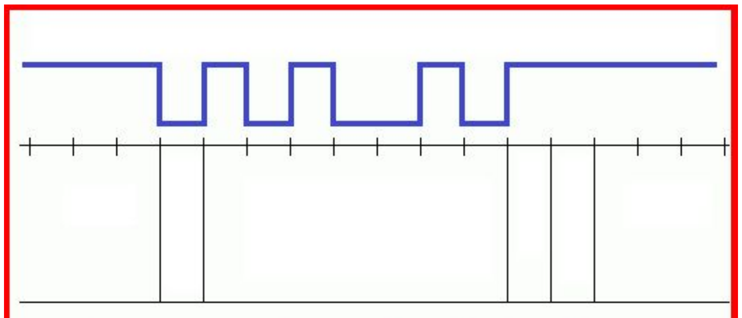
Description/Overview

It's a well-known fact that all devices in a system need some kind of communication method to interact with each other in order to maintain proper functioning of the whole system. In practice, we can divide these communication methods into two: parallel communication and serial communication. In parallel communication, multiple lines carry data bits from the source to destination. This is simple, but the cost is very high due to multiple parallel lines/wires needed to connect the devices.



Parallel communication is a method of conveying multiple bits simultaneously

In serial mode, only one line carries the data bits from source to destination and transmits them in a serial (one after one) manner [Note: serial communication may also use more than one line for synchronization]. Literally, in serial communication, a sequence of bits are traveling through the track like a train!



Serial communication is the process of sending data one bit at a time

In this lab, you will work with a serial data controller on the AVR to read and write serial data and interact with that data via a serial port terminal.

Objectives

Students will:

- Utilize the AVR's USART to read and write data over a serial port

Materials Required

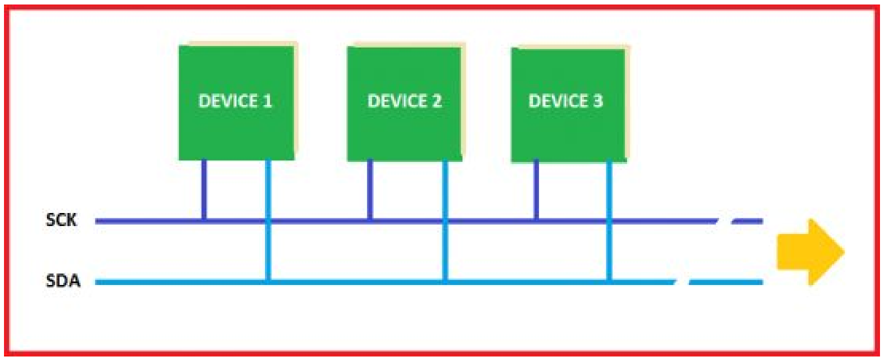
- Atmel Studio 7 Installed on PC (ECE CLC Computers will have this)
- AVR Simon Board with a USB cable
- 9-pin Serial Cable

Preparation

In order to be successful with this lab, students should review Chapter 11 in the Mazidi textbook.

Background *(excerpted from electroschematics.com - used by permission of author T.K. Hareendran)*

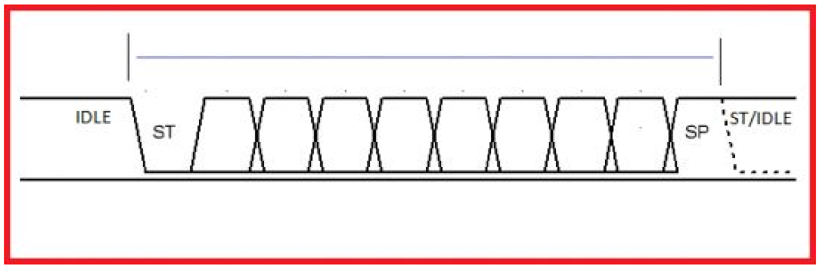
Serial communication can be of two types: asynchronous and synchronous. In asynchronous communication, the receiver has no prior intimation about the arrival of data bits/packets, and because of this receiver needs some other information (like baud/bit rate, packet size, parity type, number of stop bits, etc.) as well. In synchronous communication, besides the single data line, it has one or more other lines for synchronization. The extra lines carry either clock or any other crucial information.



Synchronous communication can be divided into two types: Serial Peripheral Interface (SPI), and Inter-Integrated Circuit Communication (I2C). The I2C bus is just a Two Wire Interface (TWI) with two wires, called SCL/SCK (Serial Clock) and SDA (Serial Data). The clock line (SCK) is used to synchronize all data transfers over the I2C bus. The SCL/SCK & SDA lines are connected to all devices on the I2C bus. There needs to be a third wire which is just the ground (0V). There may also be a 5-volt wire distributing power to the devices. The I2C bus is a topic for further discussion elsewhere.

The AVR UART
Universal Asynchronous Receiver Transmitter (UART) is a popular example of serial asynchronous communication. In UART, the arrival of the packet is indicated by a "Start bit" appended by the beginning of every packet, and the end of packet is indicated by a (one or more) "Stop bit". UART can be defined as a programmable piece of hardware that can communicate with other asynchronous serial devices in an asynchronous serial mode. UART works in "Full Duplex" mode because receiving and transmitting is done on independent pins named as RXD and TXD respectively.

The AVR UART transmits one character pack at a time. Each of these packets contains one Start bit followed by 5 -9 Data bits followed by (optional) Parity bits, which is finally followed by one (or two) Stop bits. When there is nothing to transmit the transmission line remains in High (H) state which is the idle state of the line. When the transmitter gets some packet to transmit, it pulls the line to Low (L) level to transmit the Start Bit (ST).



After receiving this, the receiver prepares itself to handle the upcoming Data bits. As receiver knows the baud rate (bit rate), it starts sampling the line after every bit delay (also known as Bit length/Bit duration, which is the inversion of Bit rate). Bit delay indicates the amount of time the line represents one (1) bit. After the completion of reception of the reception of total Data bits, receiver expects one Parity (either odd, even or no parity) bit. After this, the receiver looks for one (or more) logic-high (H) state Stop bits (SP). The whole process is followed by either the next Start bit or by the IDLE signal. The frame structure is shown.

AVR UART Registers

The configuration of AVR UART requires access to some registers, which are:

- USART Band Rate Register – UBRRH & UBRRL
- USART Control and Status Register A – UCSRA
- USART Control and Status Register B – UCSRB
- USART Control and Status Register C – UCSRC
- USART Data Buffer Register – UDA

Here, note that many UART comes with Synchronous communication facility, and this hardware block is known as “Universal Synchronous/Asynchronous Receiver Transmitter” (USART). AVR USART is fully compatible with the AVR UART for transmitter operation, receiver operation, band rate generation, etc.

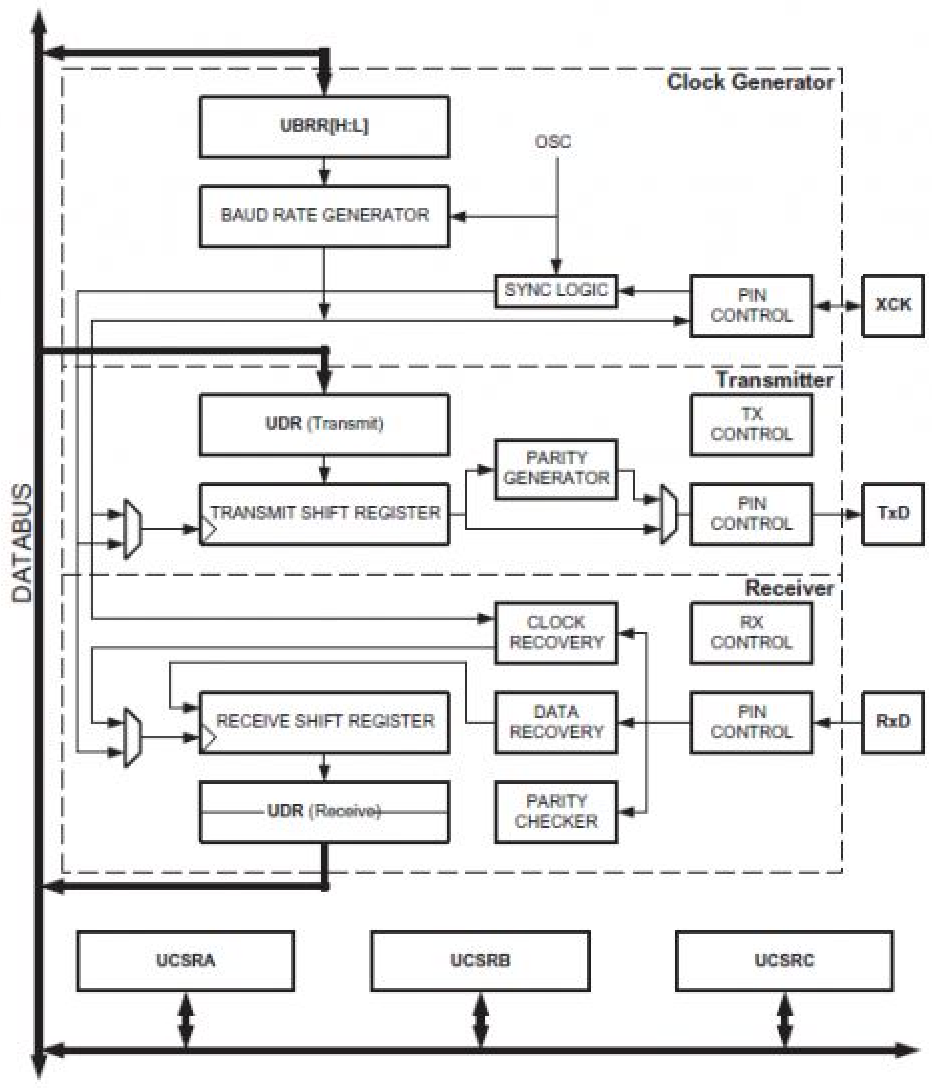
When it comes to AVR UART configuration, it is required to define the packet format a transmitter is going to transmit, and packet format is defined by character size, parity bits and stop bits. Another one needs configuration is the baud rate by putting up some egresses in the UBRR (USART Band Rate Register). We know from the previous discussion that the USART band registers are responsible for configuring baud rate of UART, and the value in this register will define the exact baud rate.

The AVR datasheet contains the table of values for a particular baud rate at a particular crystal frequency. The relation between baud rate and UBBR values is given by the formula: UBRR = Fosc/16xBaud – 1. For instance, with 16 MHz crystal frequency and UBRR contains rounded off the value of 103 in it, the baud rate will be 9600 bps. In the formula, UBRR is the UBRR register value in integer, Fosc is the crystal frequency in Hertz, and the baud rate is the required baud rate in bps. Note that only 12 bits of UBRR are used for defining baud rates (i.e., 8 bits of UBRRL and 4 low order bits of UBRRH).

The Atmega controller has a so-called Universal Asynchronous Receiver Transmitter (UART) which can be used to communicate with any other RS-232 device like a PC. A simplified block diagram of the USART Transmitter (from the Atmega8L datasheet) is shown here. CPU accessible I/O Registers and I/O pins are shown in bold lines.

The dashed boxes in the figure separate the three main parts of the USART: A clock generator, Transmitter, and Receiver. Control Registers are shared by all units. The clock generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCK (transfer clock) pin is only used by synchronous transfer mode.

The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator, and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a parity checker, control logic, a shift register and a two-level receive buffer (UDR). The Receiver supports the same frame formats as the Transmitter and can detect Frame Error, Data Over Run and Parity Errors.



For handling UART, first, the baud has to be specified in UART. After the initialization, UART can perform reading and writing tasks. Here, both reading and writing data buffer registers share the same input-output (I/O) address referred to as UDR (USART Data Request). The transmit data buffer register (TXB) will be the destination for data written to the UDR register location. Reading the UDR register location will return the contents of the receive data buffer register (RXB).

Your textbook has some sample code blocks which demonstrate how to set up and utilize the AVR's UART which may be helpful during this lab.

Procedure

1) Create a *UART_Init()* function that sets up the UART for use with a PC serial port. Standard parameters for this tend to include a baud rate of 9600, 8 start bits, no parity and 1 stop bit.
2) In the *main()* part of your program, create C code that lights an LED when a particular character is received. For example, if the letter "A" is received, LED1 might light up for 1 second. Your program should light different LEDs for four different characters. This routine should be written using interrupts (as opposed to polling).
3) Modify the main function to output any arbitrary letter to the serial port when one of four buttons is pressed on the AVR Simon Board. For example, when SW1 is pressed, a letter "G" could be transmitted.

Deliverables

This lab requires you to submit a formal lab report. Your TA will review what goes into a formal lab report. As you have in the past, your code submission should be thoroughly documented and demonstrated to your TA. Make sure your code also includes a documented configuration of how the hardware is connected (i.e. what pins are connected to what devices and what they are used for). A picture of the hardware setup/wiring would also be helpful.

Questions/Observations

1. What parameters should the transmitter and receiver agree on before starting a serial data transmission?
2. Explain the difference between polling and using interrupts in relation to receiving data using the USART.
3. Assuming we are receiving a lot of serial data, what kinds of data structures would be convenient to help store and access all of the received characters?

References

- Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi. 2010. *AVR Microcontroller and Embedded Systems: Using Assembly and C (1st ed.).* Prentice Hall Press, Upper Saddle River, NJ, USA.
- Hareendran, T.K. *ATmega8 Advanced Guide: VR Analog to Digital Conversion (ADC) – Tutorial #13.* Accessed from https://www.electroschematics.com/10053/avr-adc/ Accessed on January 18, 2019.