SERIAL COMMUNICATIONS

Introduction

This section covers the hardware aspect of serial communications; how the message is sent through the wire and the general aspect of how the hardware works on the board.

Transmission

The method that serial communications uses to transmit data is called UART: Universal Asynchronous Receiver Transmitter. The serial communications that MPIDE has defined as Serial, Serial1, etc., all use UART (except for when communicating through USB).

Since UART is asynchronous, data can be sent and received at any time. In order to know to receive data, the transmit pin is kept HIGH and changes to LOW when it starts to send data. It then sends the data bits, and after sending 8 bits, it changes the transmit pin back to HIGH until the next transmission is ready. However, UART can be configured to send an extra (9th) bit or to check the parity to ensure that the data wasn't corrupted in the transmission. Figure 1 shows a UART transmission with just the start and stop bits. Figure 2 shows a UART transmission with a ninth bit and also a transmission with a parity bit.

This is because UART has four settings for transmission: eight bits with no parity, eight bits with even parity, eight bits with odd parity, and nine bits without parity.

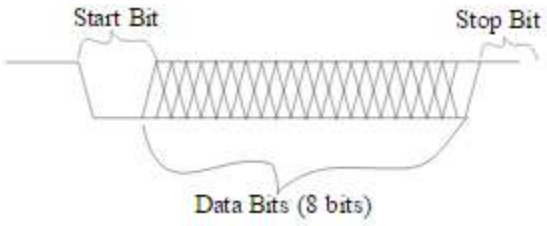

Figure 1. UART in most basic configuration.

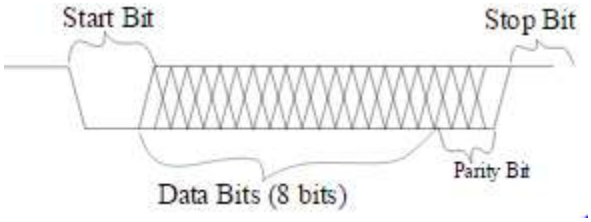

Figure 2a. UART with 8 data bits and parity.

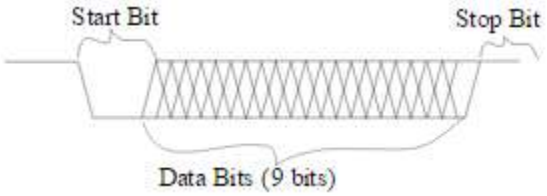

Figure 2b. UART with 9 data bits.

In order for two UART modules to send and receive without timing errors, both must be set at the right baud rate (where baud rate is basically the number of bits per second that can be sent). How the baud rate is set will be explained below in the hardware section.

Example Transmission

Lets do a simple example of sending a character through UART. To send the character 'A', first find the ASCII value. Which is 65 in decimal, 0x41 in hexadecimal, and 0b 0100 0001 in binary. Since the most basic configuration of UART is the data with a start and stop bit on the ends, Fig. 3 shows how 'A' looks when it is transmitted.

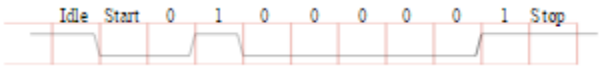

Figure 3. Sending 'A' through UART (8 bits).

There are two ways to check bit parity: by using even parity and odd parity. The quick way to determine the parity is to count the number of 1s in the data, and if there is an even number, the even parity is '0' and odd parity is '1'. If the number of 1s is an odd number, the even parity is '1' and the odd parity is '0'. Figures 4 and 5 show 'A' with both odd and even parity, respectively.

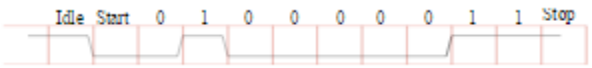

Figure 4. Sending 'A' through UART with odd parity.

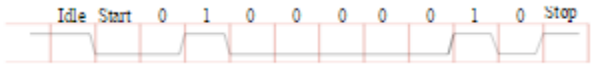

Figure 5. Sending 'A' through UART with even parity.

Hardware

The UART module is comprised of two sections: the transmitter and the receiver. In both sections, the general layout of the circuits is the same, but with data going in opposite directions. There are three primary areas of the transmitter and receiver: the FIFO (First-In-First-Out) buffer that holds the data, the shifter (either shifts the data in or out), and the configuration.

The FIFO buffer has space for 4 or 8 bytes of data (although in this case, byte can mean 8 or 9 bits). The Uno32™ has a buffer of 4 bytes and the Max32™ has a buffer of 8 bytes. Technically, we can add another byte to the size of the buffers because there is also a byte in the shifter that is being sent.

The shifter does what its name suggests; shifts. The transmitter takes the byte to be sent from the buffer, sends a start bit, starts to transmit the individual bits in the byte, and then finishes with the stop bit. What it transmits depends on the data and the configuration of the UART.

The configuration is based on three registers which control every aspect of how the UART operates. There are many different options that the UART can be set to, but most of them go beyond simply sending data. The first register is the UxMODE register, which enables or disables the UART and selects the number of data bits, parity, and stop bits. It also controls a number of other aspects.

The second register is the UxSTA register, which primarily controls the interrupt modes and indicates various status conditions. The final register is the UxBRG, which is a number that gives a certain baud rate based on the desired baud rate and frequency of the peripheral bus.

When it comes to the hardware, the UART is actually two separate circuits paired together: the transmitter and the receiver. Both the transmitter and receiver have very similar circuits, the main difference being the direction of the data.

Both the transmitter and receiver have three main sections: the data buffer, the configuration registers, and the data shifter.

The data buffer can hold up to eight bytes/characters of data to send. For the transmitter, if there is any data in the transmit buffer and the UART is enabled, then it will start sending data out, and in doing so, clear out the buffer for more data to be inputted. For the receiver, if it receives any data from the receiving pin, it stores the result into the receive buffer, which canhold up to eight bytes/characters of data. The data in the receive buffer is held until it is read.

The configuration registers (UxMODE, UxSTA, and UxBRG) are used to set the UART. UxMODE is used to enable or disable the UART, select the parity and data selection bits, and a number of other UART configurations that are explained in the manual.

UxSTA is the status and control register and is used primarily to select how the UART triggers interrupts and other functions that are further explained in the manual. UxBRG is the register that is used for generating the baud rate that the UART uses. The equation in Figure 6 gives the value to set UxBRG based on the baud rate to set the UART and the frequency of the peripheral bus clock (the clock that the UART clock is based on).

$$UxBRG = \frac{F_{PB}}{16 * BaudRate} - 1$$

Figure 6. UxBRG Calculation.

The data shifter is exactly what it sounds like, it shifts data. It is the transmitting shifter that takes data from the transmission buffer and transmits each bit according to the UART configuration. The receiving shifter, once it receives the start bit, compiles the data bits into an eight or nine bit register. If the UART is configured for parity, it will also check for parity. If the receive buffer is full and the UART receives more data, an interrupt is triggered, which can result in the loss of the entire recieve buffer if the interrupt handler does not read in the buffer.

Source: https://learn.digilentinc.com/Documents/394