Laboratory Exercise A
TCES 330, Spring 2018

Universal Asynchronous Receiver Transmitter (UART)

Background

At least one Universal Asynchronous Receiver Transmitter is found in just about every microcontroller and microprocessor found on the market today. It affords a well-known, proven, dependable means of communication between processors and between processors and sensors or other peripheral devices. In this exercise we will design and build the transmitter part of a UART.

The UART is a serial device, meaning that data is sent over a wire, one bit at a time. In the usual configuration (called full duplex), one wire is used to transmit data out of the device and another wire is use to receive data. A third wire is always present to provide a common signal ground (0 voltage reference). See Figure 1.

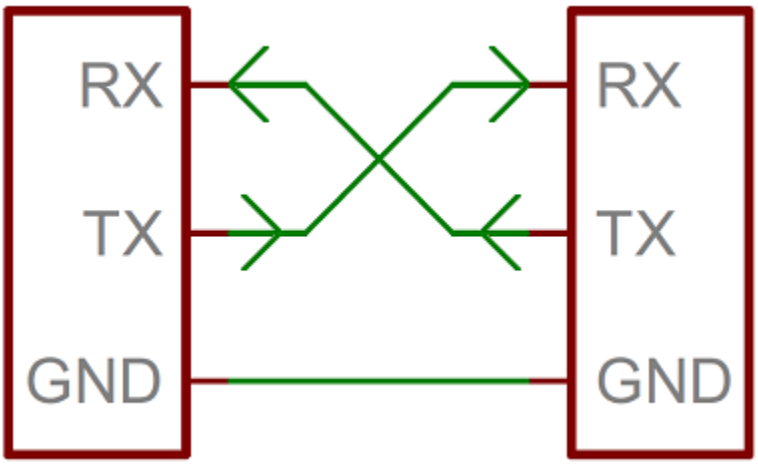

Figure 1. UART to UART Connection

For our purposes, we will assume data comes in 8-bit chunks (bytes). So for transmitting, data enters the UART in parallel and gets shifted out over the serial connection; for receiving, data comes in in serial fashion and exits in parallel (bytes). This is shown in Figure 2.

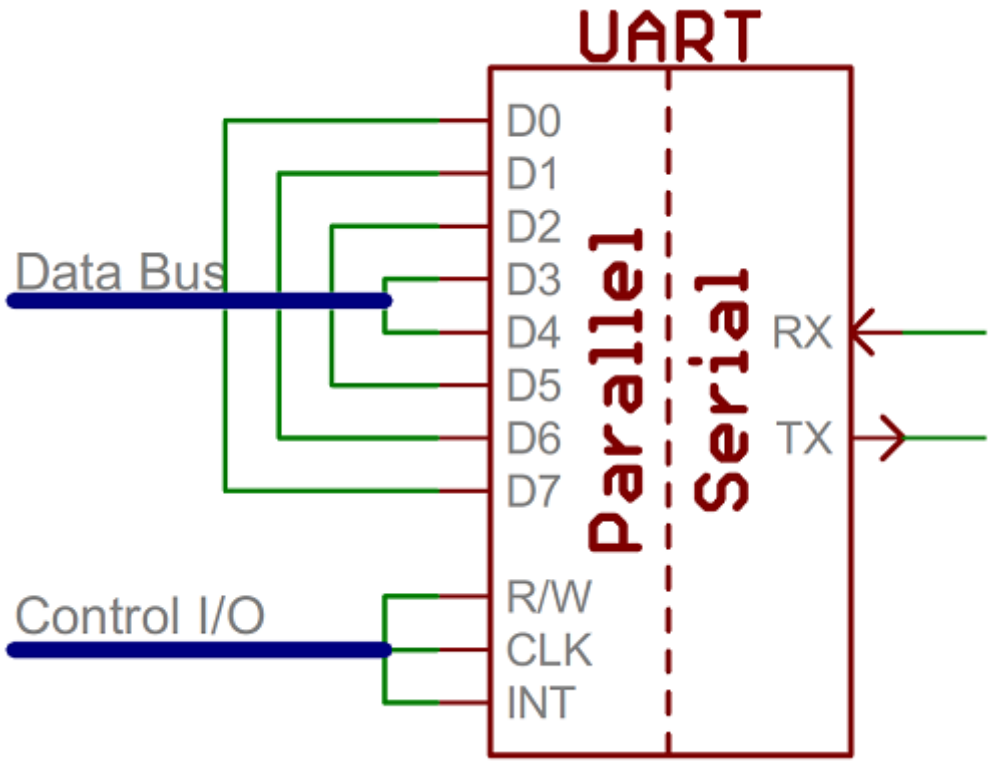

Figure 2. UART Data Input and Output

We will be sending text with our UART. We will use the standard ASCII encoding scheme to send and receive all the characters, punctuation, numbers, and control signals that we'll be using. Refer to https://en.wikipedia.org/wiki/ASCII and http://www.asciitable.com/ for a description of this coding scheme.

For a top-level description of the UART, refer to https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter or https://learn.sparkfun.com/tutorials/serial-communication We will be sending data packets (characters) that are eight bits long, no parity bits, and one stop bit (referred to as 8N1).

The anatomy of an 8N1 UART data frame is shown in Figure 3.

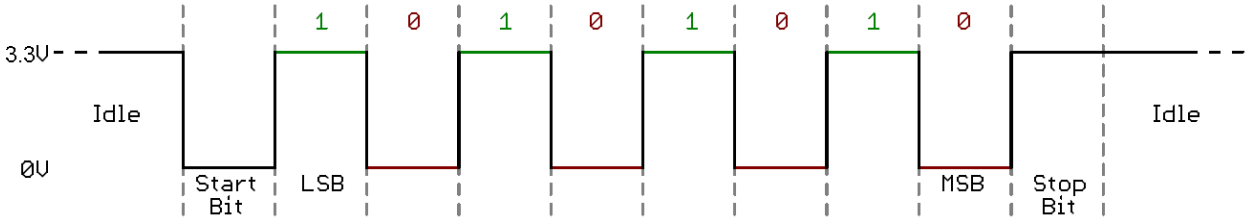

Figure 3. UART Data Frame

When not sending data, the UART transmit line (TX) is held high (3.3 V in our case because that's what the DE2-115 uses). When the UART begins to transmit a byte, it pulls the TX line low for one bit time. This is called the Start Bit. The Start Bit is followed by eight bits of data, starting with the least significant bit (LSB). In Figure 3 everything between the LSB and MSB inclusive can be either a 1 (high voltage level) or a 0 (0 voltage level), as dictated by the ASCII table (http://www.asciitable.com/). The UART holds the TX line high for one bit time following the MSB. This is called a Stop Bit. More than one Stop Bit can be specified, but we'll use only one Stop Bit. If more than one byte is being sent, the UART immediately follows the Stop Bit with the next byte's Start Bit (i.e., there would be no 'Idle' time as shown in Figure 3). Note that because of the Start and Stop Bits, the UART must transmit at least 10 bits to send 8 bits of actual data.

We are not using parity bits, but when they are included in a packet, they can help detect errors.

As an example, a string "Mom" followed by a line-feed (0X4D, 0X4F, 0X6D, and 0X0A) has been sent. Note that the LSB is sent first (right after the Start Bit).

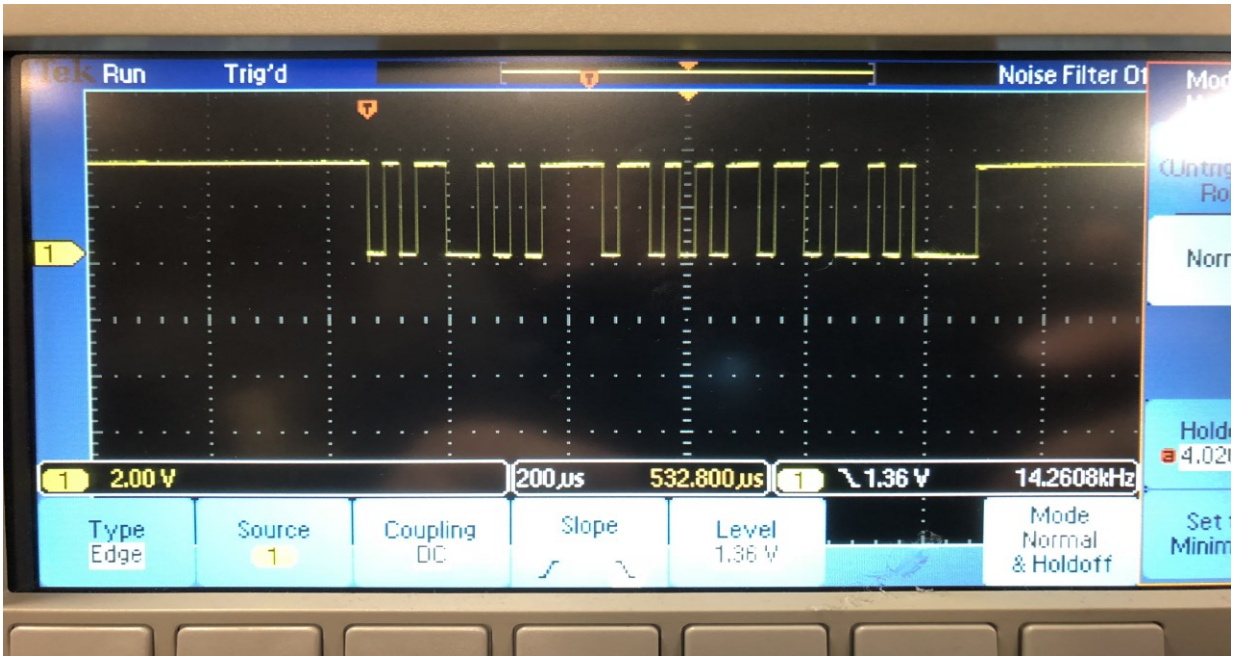

Figure 4. The Letter 'A' Sent Repeatedly

Note Figure 4 is captured from an oscilloscope in CP206D.

The transmit (TX) side is responsible for sending waveforms such as shown above and the receive (RX) side is responsible for receiving these waveforms and figuring

out what part of the received signal is a Start Bit, Stop Bit, Data Bits, and Parity Bits, if used. The receive part is more complex than the transmit part.

How long is a bit time? This can vary, but the common bit rates (called baud rate) are shown on
https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
We will be using a fixed baud rate of 38,400 bits/sec. Note, the sender and receiver must both agree on a baud rate and a data format (8N1) before communication can take place. With 38,400 bits/sec, we have 1/38,400 sec/bit = 26.042 μsec/bit.

The Problem

We will build the transmit side of a UART and repeatedly transmit the exact phrase "Hello World!" followed by a linefeed character (ASCII 10d) to an Oscilloscope. This implies that we have two major modules to build: a UART to transmit the data and another circuit that feeds the UART data, one byte at a time. You are to use state machines to build both of these circuits. A block diagram is shown in Figure 5.

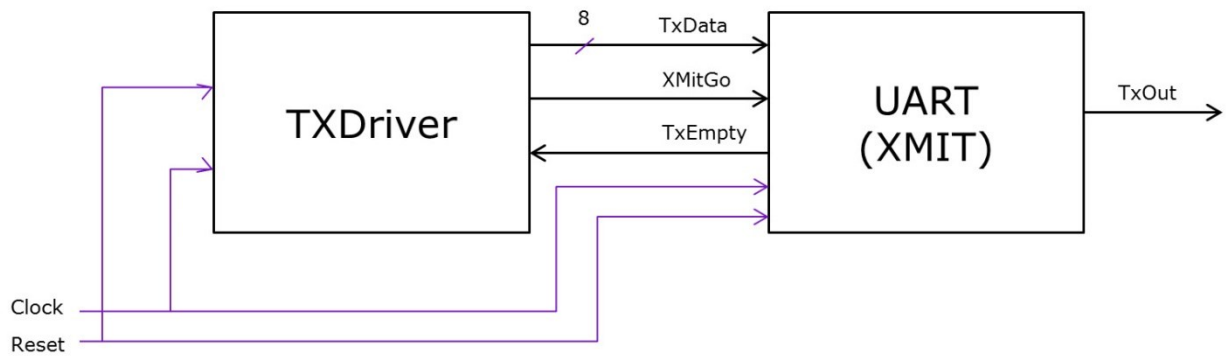

Figure 5. Block Diagram.

The signals are:

- Clock is the system clock (50 MHz)
- Reset is a system reset, active high
- **TxData is a byte** for the UART to transmit
- XMitGo is the signal that TxData is present and should be transmitted
- TxEmpty is a signal that means the UART is ready for a new TxData
- **TxOut is the serial output**

TXDriver

TXDriver should hold the data-to-be-transmitted in a memory block. It should hold an address into the memory block and fetch data out of the memory, one byte at a time to send to the UART. A new address should be generated and new data should be sent when TxEmpty is true. XMitGo should be true when new data is ready to be loaded into the UART.

For this exercise, TXDriver should pause for one second after the required phrase has been transmitted and prior to transmitting the phrase again.

UART (XMIT)

UART (XMIT) should read new data in and save (register) it when XMitGo is true. It should add the Start Bit and Stop Bit and transmit the entire data frame (TxOut). It should set TxEmpty when it's ready to accept new data.

Testing

You will find timing to be a critical issue in this exercise. The only way you will be able to gain insight into timing issues is to simulate using ModelSim. Develop testbenches for each module you write and for the entire system shown in Figure 5, above.

Only after your simulations work as expected, write a top-level module called LabA to connect your system to the DE2-115 board as follows:

- Use CLOCK_50 as the system clock.
- Use KEY[0] as a system reset
- Output TxOut to GPIO[1]. Refer to Figure 6 below. Be very careful not to short any of the GPIO pins to ground and don't connect them to any pin on a device that uses 5 V (TTL) logic levels. This includes most all of the Arduino family of devices.

Take care of unused pins, and unwanted warnings. Make an sdc file.

Now connect the output to an oscilloscope.

If you are not seeing what you expect, make sure the baud rate is set to 38,400 (Rate: 38400).

GND pins. Figure 4-15 shows the I/O distribution of the GPIO connector. The maximum power consumption of the daughter card that connects to GPIO port is shown in Table 4-10.

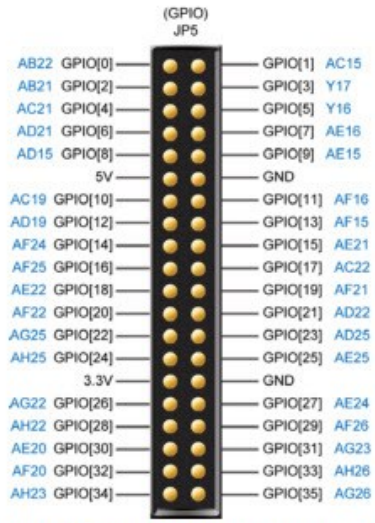

Figure 4-15 GPIO Pin Arrangement

Figure 6. 40-Pin GPIO Header (from DE2-115 User's Manual)

The Report

When you have succeeded with your tests, write a formal laboratory report using the sample lab report as a guide.

- Pay special attention to the section on testing.
- Don't include your code in the report. You are turning in your files, so don't clutter up your report. If you need to, you may use snippets of code to make a particular point.
- This is the multi-media generation, so use lots of figures, pictures, screen shots, block diagrams, and whatever else it takes to make your report convincing and interesting.
- Don't assume your audience has access to this document or knows any more about UARTs than you did before you started this project.
- Do assume that your audience knows SystemVerilog, MultiSim tcl, and the basics of Quartus at the same level you know these things now.

Turn in a zip file with your project files just like you did with your homework assignments. Turn in your report at the designated place on Canvas for the Lab A report.

Extra Credit

Use the Quartus RS-232 UART LPM to instantiate a UART. Bring out the UART receive pin to GPIO[3]. Connect your UART (XMIT) output (GPIO[1]) to the UART LPM receive pin (GPIO[3]) and send a message that can be displayed on the Hex displays – like 'HELLO', but you can be more inventive than that! You want to slow things down to send one character per second and have it scroll across the displays.