**What is a UART?**

AKA Serial Port, RS-232, COM Port, RS-485

This type of functionality has been referred to by many different names: Serial Port, RS-232 Interface, COM Port, but the correct name is actually **UART (Universal Asynchronous Receiver Transmitter)**. A UART is one of the simplest methods of talking to your FPGA. It can be used to send commands from a computer to an FPGA and vice versa.

A UART is an interface that sends out usually a byte at a time over a single wire. It does not forward along a clock with the data, which is why it is called *asynchronous* as opposed to synchronous. UARTs can operate in either Half-Duplex (two transmitters sharing a line) or [Full-Duplex](http://www.nandland.com/articles/tri-state-buffer-half-full-duplex.html) (two transmitters each with their own line). UARTs have several parameters that can be set by the user. These are:

Baud Rate (9600, 19200, 115200, others)

Number of Data Bits (7, 8)

Parity Bit (On, Off)

Stop Bits (0, 1, 2)

Flow Control (None, On, Hardware)

These settings need to be the same on both sides of the interface (the receiver and transmitter) for communication to work correctly. When the settings are incorrect, strange and unusual characters can appear on the screen. Let's look at each of these settings individually.

**Baud Rate** is the rate at which the serial data is transmitted. 9600 Baud means 9600 bits per second. **Number of Data Bits** is almost always set to eight. A **Parity Bit** can be appended after the data is sent. Parity is always computed by doing an [XOR Operation](https://www.nandland.com/articles/boolean-algebra-truth-table-and-gates.html) on all of the data bits. A **Stop Bit** always set to 1, and there can be 0, 1, or 2 Stop Bits. **Flow Control** is not typically used in present day applications and will likely be set to None.

As mentioned previously, there is no clock that gets sent along with the data. In any interface that does not have a clock, the data must be sampled to recover it correctly. It needs to be sampled at least eight times faster than the rate of the data bits. This means that for an 115200 baud UART, the data needs to be sampled at at least 921.6 KHz (115200 baud \* 8). A faster sampling clock can be used.

[**Click Here to see a completed UART in VHDL and Verilog!**](https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html)

**A Trick For Debugging UARTs**

Often time when debugging UARTs it is difficult to know if your computer serial port is not working or if it's your FPGA code. Here's a trick: Take your 9-pin cable and jumper pins 2 and 3 together. You can use a paperclip or whatever you have handy. Pins 2 and 3 are TX and RX. If you connect them together, any command you send from the computer will be received by the computer. You have created serial loopback!

Open your terminal program, try to connect to your serial cable. Hit any key on your keyboard. Baud rate and things don't matter because it's a loopback. If you see the character you sent out received on the terminal, your serial cable works! If not, you have a problem with your cable or with your serial program. I have had good luck using [Tera Term](http://en.sourceforge.jp/projects/ttssh2/releases/) in the past and recommend that. If you are having trouble with your serial cable, try this [USB to RS-232 Cable](http://www.amazon.com/gp/product/B0007T27H8/ref=as_li_tf_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=B0007T27H8&linkCode=as2&tag=nandland-20)http://ir-na.amazon-adsystem.com/e/ir?t=nandland-20&l=as2&o=1&a=B0007T27H8 at Amazon. I've used it and it works great with Windows.

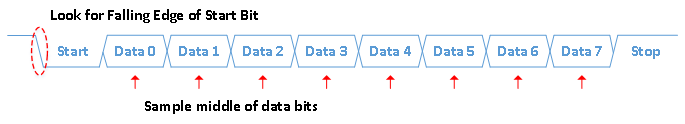
# UART, Serial Port, RS-232 Interface

## Code in both VHDL and Verilog for FPGA Implementation

Do you know how a UART works? If not, first brush up on the [basics of UARTs](https://www.nandland.com/articles/what-is-a-uart-rs232-serial.html) before continuing on. Have you considered how you might sample data with an FPGA? Think about data coming into your FPGA. Data can arrive by itself or it can arrive with a clock. When it arrives with a clock, it is call synchronous. When it arrives without a clock, it is called asynchronous. A UART is an asynchronous interface.

In any asynchronous interface, the first thing you need to know is when in time you should sample (look at) the data. If you do not sample the data at the right time, you might see the wrong data. In order to receive your data correctly, the transmitter and receiver must agree on the **baud rate.** The baud rate is the rate at which the data is transmitted. For example, 9600 baud means 9600 bits per second. The code below uses a generic in VHDL or a parameter in Verilog to determine how many clock cycles there are in each bit. This is how the baud rate gets determined.

The FPGA is continuously sampling the line. Once it sees the line transition from high to low, it knows that a UART data word is coming. This first transition indicates the start bit. Once the beginning of the start bit is found, the FPGA waits for one half of a bit period. This ensures that the middle of the data bit gets sampled. From then on, the FPGA just needs to wait one bit period (as specified by the baud rate) and sample the rest of the data. The figure below shows how the UART receiver works inside of the FPGA. First a falling edge is detected on the serial data line. This represents the start bit. The FPGA then waits until the middle of the first data bit and samples the data. It does this for all eight data bits.



**UART Serial Data Stream**

The above data stream shows how the code below is structured. The code below uses one Start Bit, one Stop Bit, eight Data Bits, and no parity. Note that the transmitter modules below both have a signal o\_tx\_active. This is used to infer a [tri-state buffer for half-duplex communication](https://www.nandland.com/articles/tri-state-buffer-half-full-duplex.html). It is up your specific project requirements if you want to create a half-duplex UART or a full-duplex UART. The code below will work for both!

If you want to simulate your code (and you should) you need to use a [testbench](https://www.nandland.com/articles/what-is-a-testbench-fpga.html). Luckily there is a test bench already created for you! This testbench below exercises both the Transmitter and the Receiver code. It is programmed to work at 115200 baud. Note that this test bench is for simulation only and can not be synthesized into functional FPGA code.