UART Implementation Workshop

Lab 03

Table of contents

01
UART Protocol Theory
How's the UART protocol works

02
Implement Receive State
Implement Receive State for receiving data from uart

03
Implement Transmit State
Implement Transmit State for transmitting data from uart

04
Exercise
Tasks For Exercise

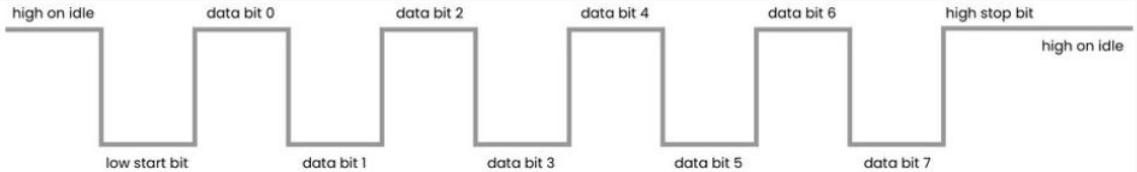
01

UART Protocol Theory

UAR Protocol theory

UART Protocol Theory

With UART you send data from one device to another over a single wire. You send 1 start bit of data, then an agreed upon amount of data bits (usually 8) and finally a stop bit.
We will be looking at the simple case of 1 start bit, 8 data bits and a stop bit. The data is transferred the least significant bit first.



Example sending the byte 01010101 or 85 in decimal

There is no clock signal to synchronize both sides like in some other common protocols where both sides need to agree in advance on a frequency or "**baud rate**" which is the amount of bits per second and then each side needs to manage their own clocks to meet the desired frequency.

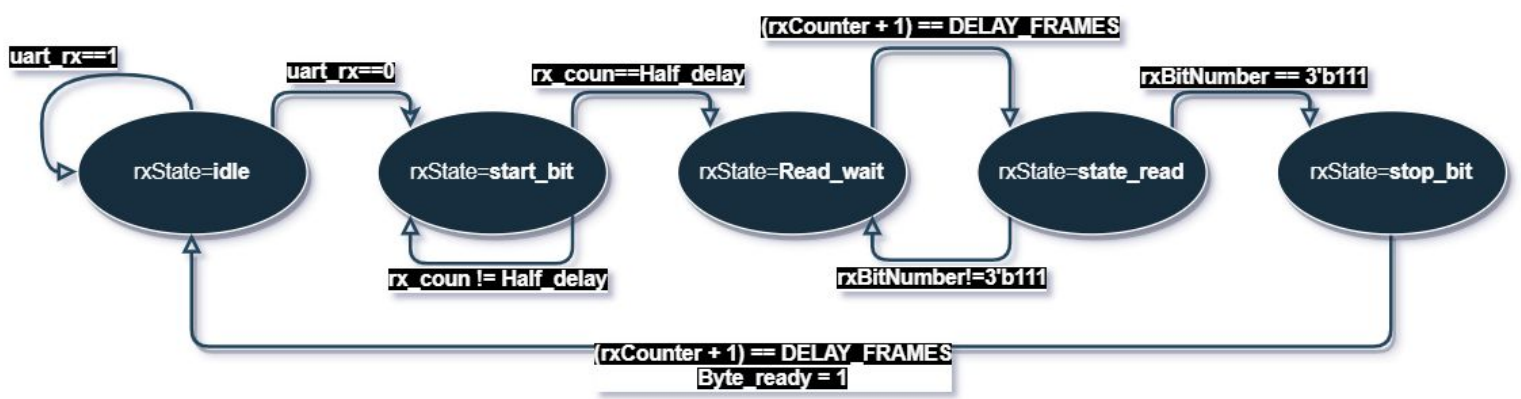
02
Receive Logic Of UART

Receiving Data from UART

Receive State

We can start in an "**idle**" state when we see the "**start bit**" we can start receiving data and go to the "**read data**" state and then once we finish the bit we can go to the "**stop bit**" state finally returning back to "**idle**" ready to receive the next communication.

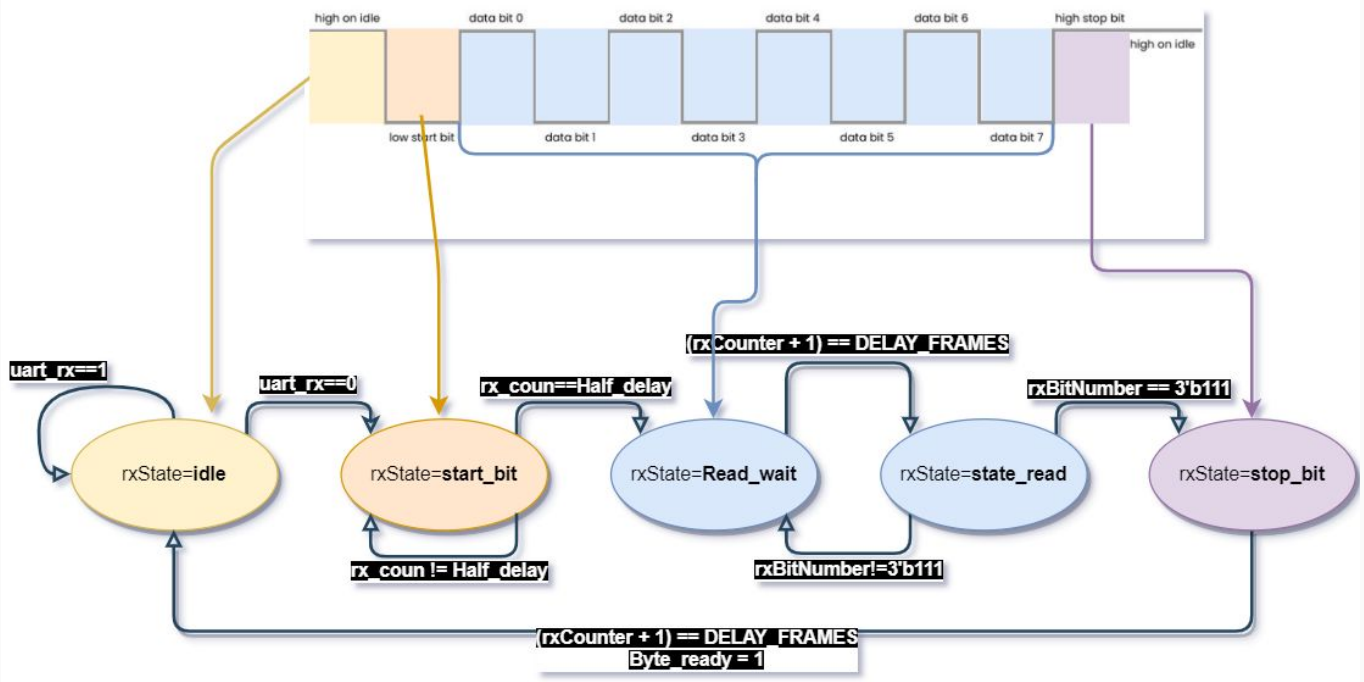
State Machine:



- The **rxState** register can hold in-which state we currently are in
- rxCounter for counting clock pulses because **baud rates** of **115200** bits per second, dividing the **27Mhz** by this number = **DELAY_FRAME** = **234**. So we saw 234 clock pulses is 1 UART bit frame
- **rxBitNumber** which can keep track of how many bits we have read.

Receive State

Receive State Waveform analysis



03

Transmit Logic Of UART

Transmit data from UART

Transmit State

The sending part works a lot like how we set up the receiver earlier, but with a key difference: instead of starting to count from the middle of the pulse, the sending side changes the line right at the start of each bit frame.

Similar to the receiver, the sending side uses a few registers. One keeps track of the transmission process, another counts clock cycles, dataOut holds the current byte being sent, and txPinRegister stores the value for the uart_tx pin. The last two registers help keep tabs on which bit and byte are being sent.

In our example, we're sending a message stored in memory, so we need to know which byte is currently being sent. The "assign" statement links the uart_tx wire to our register, and the last two lines create a memory structure where each cell holds 8 bits, with our example having 12 cells in total.

Moving on, let's define the different stages of our sending process. Unlike the receiver, we don't include an extra "wait" stage because we're not dealing with a middle frame offset. However, we add an extra stage at the end to debounce the button, which we'll use to decide when to send data.

Note.: **First, try creating your own state machine diagram and waveform analysis for transmitting data. After that, you can use the source code as a reference on Github ABDUL MUHEET GHANI**

04

Exercise

Exercise

Transmit State

- As an exercise, now you'll receive data by pressing buttons on the keyboard, and that data should be received by the FPGA, displaying an ASCII pattern on the LED.

Testimonial

Author:

Abdul Muheet Ghani, Research Assistant at MERL-UITU.

Under The Supervision Of:

Dr.Ali Ahmed: Team Lead MERL.
Sajjad Ahmed: Research Associate.

Thanks: Lushay Lab, Slidesgo.

Future Work

This is version 1.0 of our course. We will continue this training and very soon release further versions. For any questions or to stay connected with us

Github: Abdul Muheet Ghani.
Gmail: Dr.Ali Ahmed. Sajjad Ahmed. Abdul Muheet Ghani.