CS 254 PROJECT

UNIVERSAL ASYNCHRONOUS RECEIVER AND TRANSMITTER

By:

Ghurye Sourabh Sunil (130050001)

Nikhil Vyas (130050023)

Utkarsh Mall (130050037)

Mayank Sahu (130050038)

Nitesh Dudhey (130050039)

GOAL :

To design and simulate universal asynchronous receiver/transmitter circuit (UART) in Xilinx ISE and Implement the simulated design on ATLYS board.

APPLICATION :

Transmitting and receiving UARTs must be set for the same bit speed, character length, parity, and stop bits for proper operation. The receiving UART may detect some mismatched settings and set a "framing error" flag bit for the host system; in exceptional cases the receiving UART will produce an erratic stream of mutilated characters and transfer them to the host system.

Typical serial ports used with personal computers connected to modems use eight data bits, no parity, and one stop bit; for this configuration the number of ASCII characters per second equals the bit rate divided by 10.

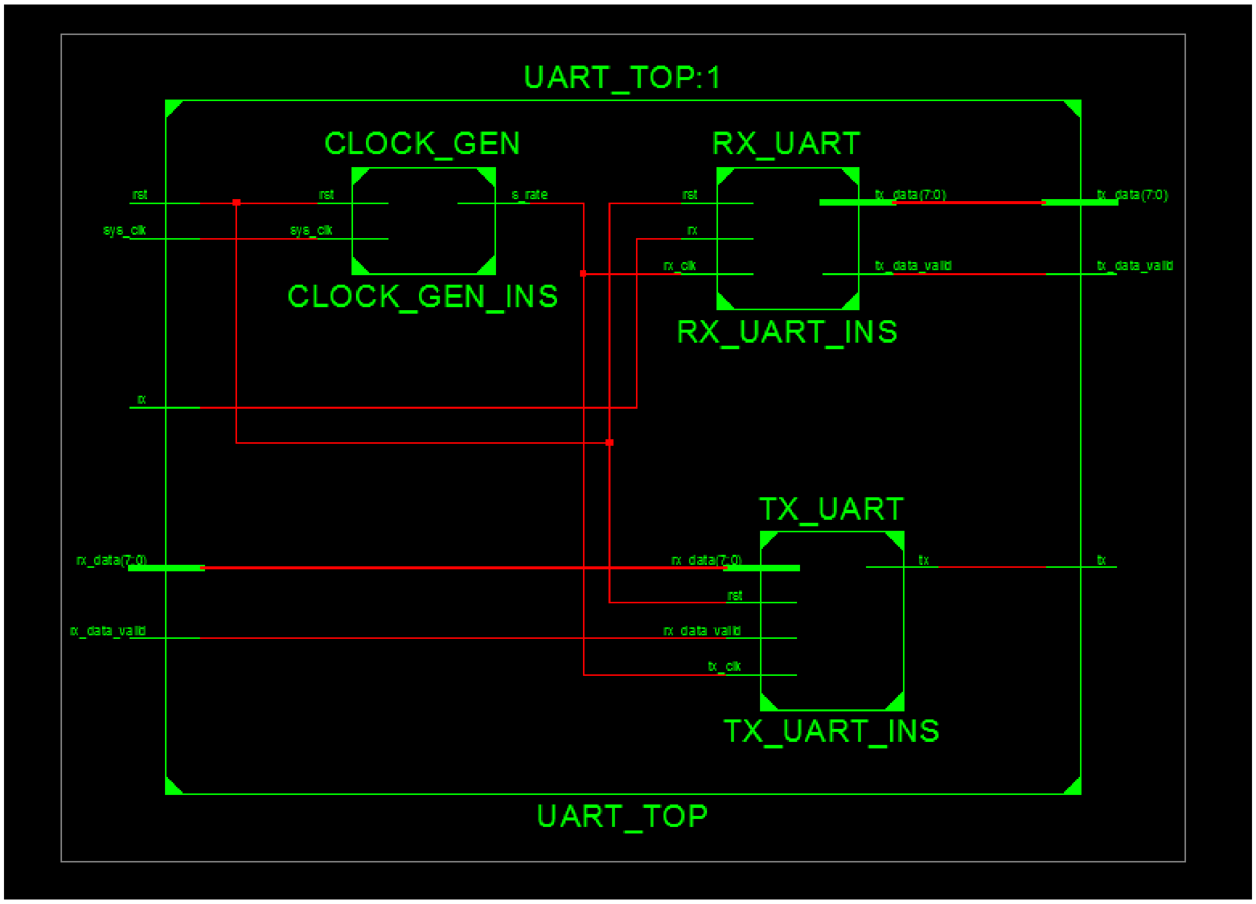
Some very low-cost home computers or embedded systems dispense with a UART and use the CPU to sample the state of an input port or directly manipulate an output port for data transmission. While very CPU-intensive (since the CPU timing is critical), the UART chip can thus be omitted, saving money and space. The technique is known as bit-banging.

RTL Components :

The UART Circuit consists of 4 register level modules:

1. Clock Generator (CLOCK_GEN.vhd)
2. Transmitter (TX_UART.vhd)
3. Receiver (RX_UART.vhd)
4. (UART_TOP.vhd)

RTL schematic without Connecting TX_DATA to RX_DATA



1. Clock Generator :

Inputs:

reset (rst) : Asynchronously sets clock to 0.

clock (sys_clk) : generates clock signal at 100 MHz frequency.

Output:

s_rate : generated clock as per our requirements

The clock generator module takes system clock as input and convert this rate to sampling rate.

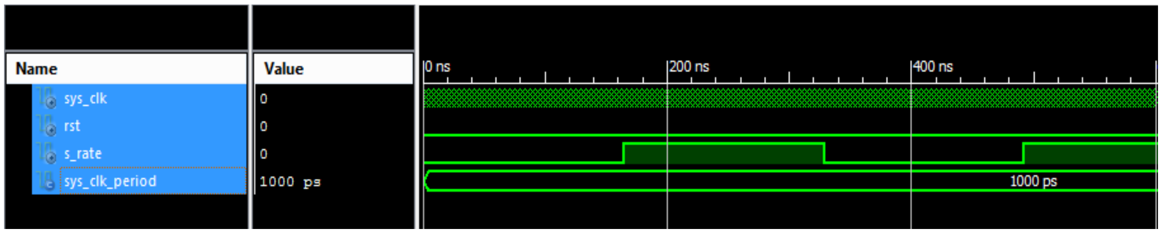
Baud rate = 19200 Hz (1)
*Sampling rate = 16 * Baud rate (2)*
clock freq = 100 MHz (3)

Baud rate is the rate at which bits are being sents .
Using the above equations we can calculate that in approximately 326 clock cycles of system clock sampling should be done once.
So we keep a counter of 8 bit. When the counter reaches 163(0b10100011) we need to change output clock signal.

In this way we can we can generate a more slower clock with our faster system clock.

'reset' signal asynchronously resets s_rate to zero.

Simulation of Clock generator :



Name	Value
sys_clk	0
rst	0
s_rate	0
sys_clk_period	1000 ps

sys_clock Time Period = 1 ns

*1/s_rate = 1 ns * 326 = 326 ns*

1. Transmitter:

Inputs:

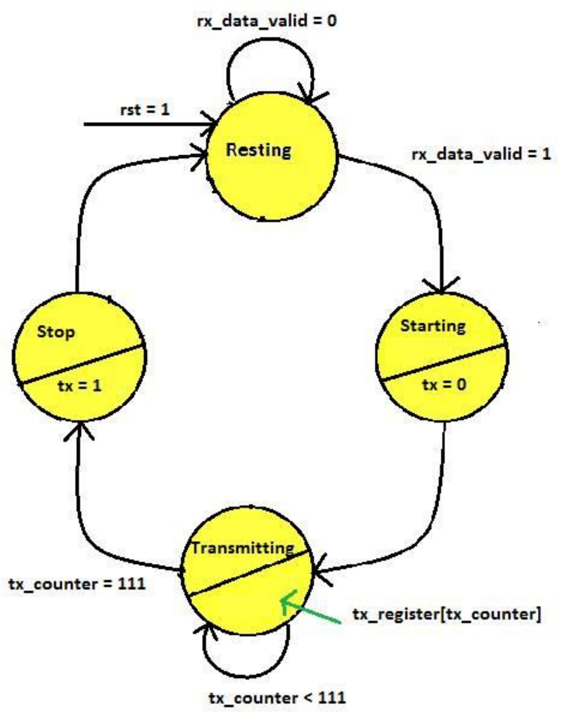
reset(rst): Resets the transmitter asynchronously.
rx_data_valid: allows the transmission of the data.
rx_data(7:0): the data input in parallel.
tx_clk: output of clock_gen module.

Output:

tx: the serially output data one-by-one.

Data is transmitted in serial fashion. Transmitter will get 8-bit data(rx_data) in parallel which should be transmitted 1-bit as (tx) in (1/baud rate) time or over 16 times sampling cycle. The transmission character is composed of an 8-bit data byte, sent LSB first, preceded by a start bit (LOW) and followed by a stop bit (HIGH).

4 states of the Transmitter:



1. **Resting**: when rx_data_valid is 1 for (1/baud rate) time, the control changes to Starting state. Else it remains at Resting state.
2. **Starting:** when tx is set to zero for (1/baud rate) time (indicates the start low bit), then the control is moved to the Transmitting state.
3. **Transmitting:** here tx is set to rx_data[i] (i = 0 to 7) each for (1/Baud rate) time, then the control goes to the Stop state.
4. **Stop:** Here tx is set to 1 for (1/baud rate) time, indicating the stop bit, and then the control goes to the Resting state.

Implementation:

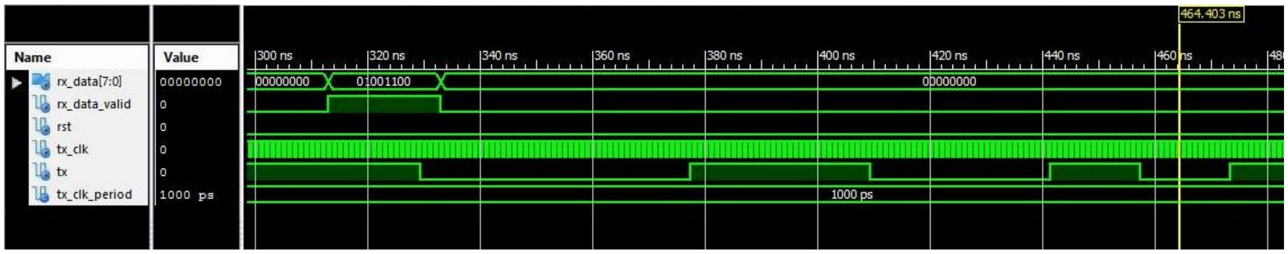
In the implementation two counters are kept:

1. One for keeping the sampling of a bit [3:0] (0 to 15).
2. One for keeping the number of bits transmitted [2:0](0 to 7).

A register(tx_register) to load the data from the input rx_data.

Reset signal asynchronously resets the circuit.

Simulation for Transmitter Module:



after rx_data_valid goes to 1 for 16 sampling cycles tx = 0 for 16 sampling cycles, then the bits are transmitted in order of LSB first, each for 16 sampling cycles, after that tx = 1 for 16 sampling cycles.

3. **Receiver :**

Inputs:

reset(rst): Resets the receiver asynchronously.

rx: the input data in serial.

rx_clk: output of clock_gen module.

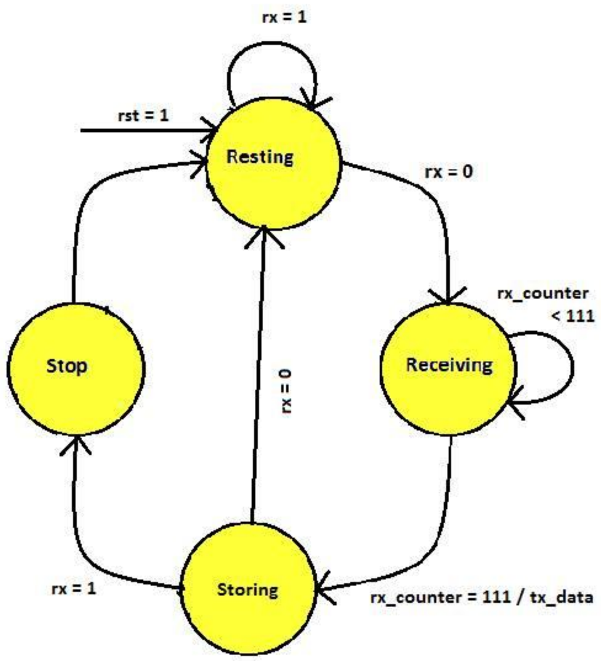
Outputs:

tx_data[7:0]: the output in parallel.

tx_data_valid: output 1 when all the 8 bits are received successfully for one rx_clk cycle.

The data is received in serial fashion. The receiver will receive 8-bit data serially(rx) which should be given as 1-byte (tx_data) in (1/baud rate) time or over 16 times sampling cycle. When no character is being transmitted, the line remains idle (HIGH).

Receiver consists of 4 states:



1. **Resting:** Idle state, tx_data_valid is 0. When rx is 0 for (1/baud rate) time, the start of the data input is indicated and hence the control moves to the Receiving state.
2. **Receiving:** As rx comes, tx_data[i] (i 0 to 7) is set to rx value each one of the Baud cycle for 8 cycles and then the control moves to the Storing state.
3. **Storing:** tx_data_valid is set to 1 for (1/Baud rate) time, indicating successful receive of data. Then the control goes to Stop state.
4. **Stop:** The control waits for (1/baud rate) time, then goes to Resting state.

Implementation:

Two counters are kept:

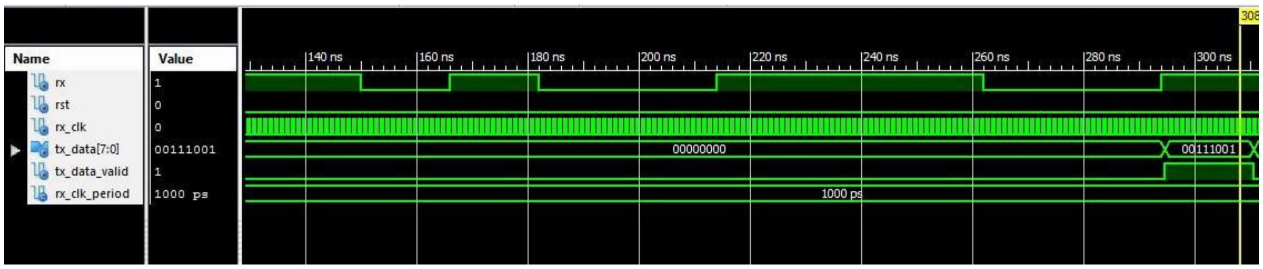
1. For keeping the sampling of a bit [3:0] (0 to 15). The 8th bit of this sample is taken as rx.

2. For keeping the number of bits received [2:0](0 to 7).

A register(rx_register) is maintained to store the temporarily received bits.

Reset signal asynchronously resets the circuit.

Simulation for Receiver Module:



after rx goes to 0 for the first time receiving procedure starts and the bits are received in LSB first order and the 8 bits received are stored as tx_data.

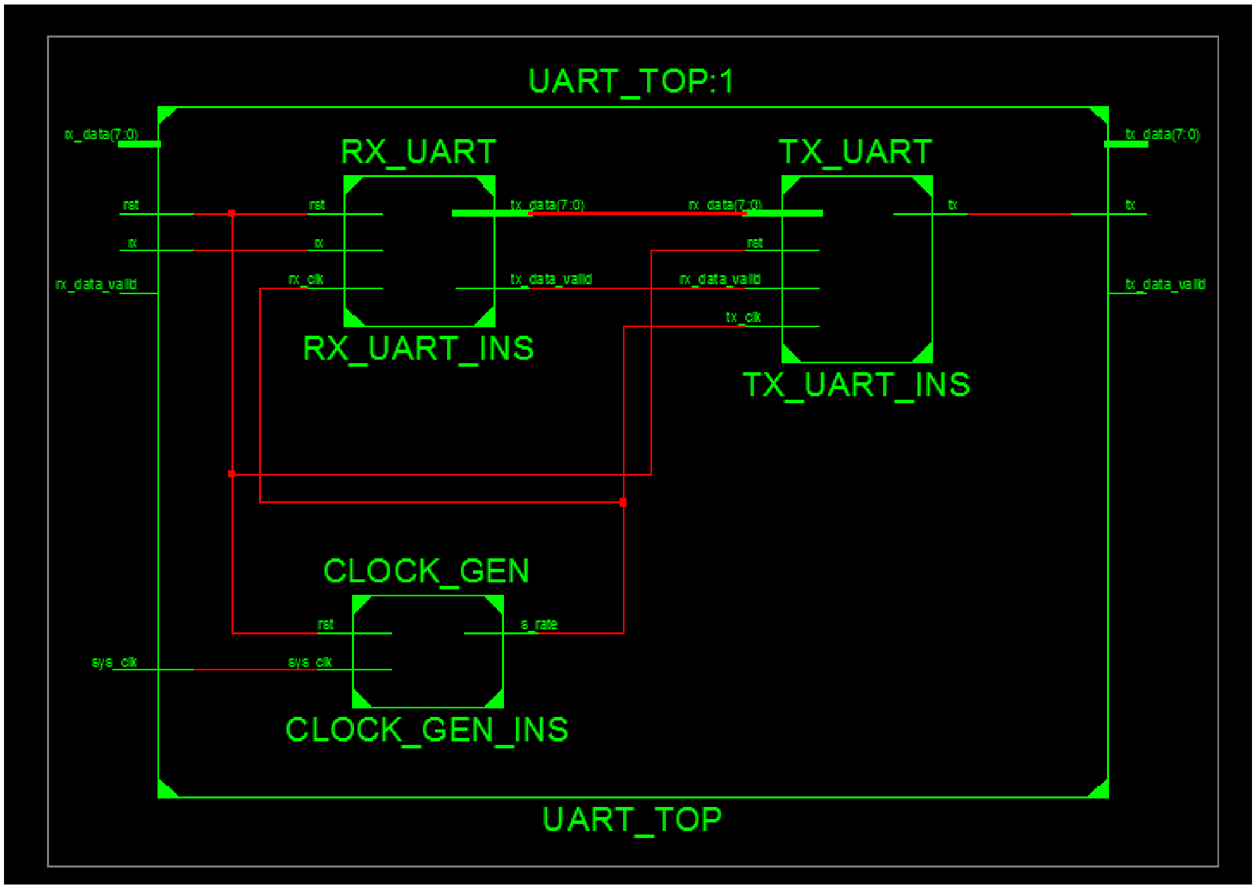
4. UART_TOP Module :

It assemble all the modules together and forms the loopback between output of receiver and input of transmitter.

Testing on Board:

While testing on board, the output of receiver is sent to the transmitter, i.e. tx_data is connected to rx_data and tx_data_valid to rx_data_valid for loopback purposes.

RTL while testing on board:



In this case, TX_DATA and TX_DATA_VALID has no use as the outputs of receiver are connected directly to inputs of transmitter.

Similarly RX_DATA and RX_DATA_VALID.
Connecting this to tera term with input/output UART ports in UCF files will give the correct result.

Note: the above may give warning. But for actual purpose comment and uncomment the line properly as shown in UART_TOP.vhd. To get working UART circuit.

Using TERA TERM :

Tera term converts ASCII characters into bits and vice-versa. It transmits and receives the bits serially. Tera term interacts with the FPGA port via a16 and b16 ports.

The baud rate of tera term has to be changed to 19200 Hz.

Driver installed for enabling the transfer via a USB-interface is XR21V1410.

Interesting observations :

- Because of presence of noise the bits are altered sometimes which lead to error in the received ASCII character.
- Since 100 MHz is not exactly divisible by 19.2 KHz and since circuit works on discrete cycles therefore the baud rate seen by tera term is not same as UART circuit, hence if reset is not done for long time some errors may creep in.