CS226 Project : UART DESIGN
KATURI SAI KIRAN (130050051)
LINGALA RAHUL (130050056)
ADDANKI RAVI CHANDRA (130050061)
VEGULLA KRANTHI (130050062)
THATIPAMULA AKHIL (130050068)

1 Objective

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

2 What is a UART?

An UART is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port. Pronounced u-art, and short for universal asynchronous receiver-transmitter, the UART is a computer component that handles asynchronous serial communication.

3 Components of UART

It mainly contains two components namely Receiver and Transmitter. Receiver receives the bits in series and sends them as output in parallel where as Transmitter receives the bits in parallel from the receiver and sends it in series. In-order to make our clock sync with the system clock we generally do sampling in UARTs by making the clock run faster than the system clock.

4 Modules inside our UART design

Apart from the top module, UART_TOP, we had used used three modules under it namely RX_UART, TX_UART and CLOCK_GEN. Here goes the description of each module.

CLOCK_GEN :

We made this module for generating the clock with frequency 307200Hz (16*19200) that we need in-order to make the clock that can sample the clock of the system. This clock is 326 times slower than normal clock whose frequency is 100 MHz and it is 16 times faster than baud-rate that we use whose frequency is 19200 Hz. This module takes the Clk as input and gives sampling_rate as output which will act as a virtual clock for modules RX_UART and TX_UART modules. This uses a counter in-order to generate the virtual clock.

RX_UART :

This is the module that is used to receive the data in series and convert it into parallel data using the virtual clock by sampling on the baud rate. It
takes the input RX in series and outputs RX_DATA and RX_DATA VALID which are a length 8 vector and a valid output which says that the output is valid. The valid will be on for one virtual clock period. We are using an FSM for this and two counters one to count for the sampling purpose and another to count the index of the vector to store in it.

**TX_UART:**

This is the module that is used to receive the data in parallel and convert it into series data using the virtual clock and send it one by one. In order to make sure that we sync with the clock of tera-term we send each bit in 16 virtual clock cycles. It takes the input TX_DATA and TX_DATA VALID which are a 8 length vector and a boolean variable which is made 1 when TX_DATA is valid. It outputs TX which is single bit signal generated using the input vector.

In idle case we always send TX as 1 which means the idle state. When ever we receive a valid vector in parallel as input we start sending each bit of the vector one by one using the output signal TX. We will first send the start bit of 0 as TX and then move to the state data for sending data. After sending data successfully we send a stop bit 1 as TX and then return to the idle state. We are using an FSM for this and two counters one to count for the sampling purpose and another to count the index of the vector to store in it.

**UART.TOP:**

This is the top module of our project that combines all the modules at a place. This module takes the RX and clk as input and sends TX as output. In this we will loop-back the RX_UART and TX_UART modules by connecting RX_DATA with TX_DATA and RX_DATA VALID with TX_DATA VALID.

5 A walk through our algorithm

According to the protocol given in the idle case the system will be sending 1 only. Hence our receiver(RX_UART) which is always listening on system will be initially in idle state. When ever it receives a 0 which says that the system is going to send some data since the protocol says that the start bit is 0. Hence once it receives the start bit it starts sampling over the baud-rate and once it receives the entire data and along with stop bit and storing it in RX_DATA it then enables the RX_DATA VALID signal.

Then since we connected the receiver and transmitter in the top module TX_UART VALID becomes 1 and hence TX_DATA reads out the value from RX_DATA as they are also connected in top module. Then it starts sending the received vector bits one by one through the TX output. This is again done using a FSM which again consists of 4 states. In idle state it sends only 1 as it is given as idle bit in the protocol. Then once it receives a valid vector as input it goes into the start state to send the start bit and after sending it it goes into the data state to send the 8-bit data that it had received. After this it goes into the stop state to send the stop bit and after this it returns to its normal idle state. As mentioned earlier it will be sending each bit for 16 clock cycles of its clock since
its clock is 16 times faster as mentioned earlier.

6 Algorithmic State Machine (ASM) of RX_UART
Algorithmic State Machine (ASM) of TX_UART
8 State diagram of RX_UART

**Explanation:**

1. It contains 4 states in total.
2. At start the machine will be in the idle state which implies that it is receiving the idle input from system. The output `RX_DATAVALID` on this state will always be 0. When ever we receive a 0 as RX we will transfer to the start state.
3. In the start state we will receive the start bit from system and we will wait for 16 clock cycles since our clock is 16 times faster than baud rate. Then after this counter wraps around we will move to the receiving state.
4. In the receiving state we will read out the data values sent by system and store in a vector by sampling. After reading out all 8 data bits the machine goes to the stop state.
5. In stop state it listens for the stop bit and then returns to the idle state.
9 State diagram of TX_UART

Explaination:

(1) It contains 4 states in total.
(2) At start the machine will be in the idle state which implies that it had not received any valid input vector and hence it sends the idle bit i.e; 1 as TX. When ever we receive a valid input vector i.e; when TX_DATA_VALID becomes one it makes a transition to the start state.
(3) In the start state it sends TX as 0 for 16 clock cycles since its clock is 16 times faster as mentioned earlier. After this counter wraps around the machine makes a transition to the send state.
(4) In the send state it sends the bits from the received vector TX_DATA one after other in series each bit again in 16 clock cycles. After sending all the bits successfully it then makes a transition to the stop state.
(5) In the stop state it sends a stop bit i.e; 1 as TX for again 16 clock cycles and then goes to the idle state.

10 Coding constrains fulfilled

(1) Must implement two FSMs:- We had done our project using two FSMs one in RX_UART and other in TX_UART. The state diagram of corresponding are given above. (2) Sampling rate must be 16 times baud rate:- We implemented this using the module CLOCK_GEN. (3) Must contain three counters:- We used one counter in CLOCK_GEN module and two counters in RX_UART and two more
counters in TX_UART. Hence in total we had used five counters.

11 Interesting inferences

(1) How to solve different phase problem of the clocks on receiver and sender? One way is obviously to have a wire that provides a common clock implementation and both the sender and receiver share this clock signal through this wire. But obviously this method is inefficient and also unnecessary as can be understood by the following trick.

We make the clock on receiver run faster (in our case 16 times) and sample the inputs at the middle of 16 cycles which approximately corresponds to the one cycle of the sender.

(2) How to achieve synchronization between two asynchronous systems? In many applications which involve data transfer or other communication formats, it is essential to have a clock synchrony between the receiver and sender. For example for sending the network packets in the link layer, the sender and the receiver must be sending and receiving data at the same rate and also they must be synchronized at the rising edges. For this sake we use a preamble of 32 bits at the beginning of the header which has alternating 1’s and 0’s.

(3) However in our case since we only have 1 byte of data to send it is unnecessary to have a preamble of that size. Instead we use a single 1-0 transition to indicate the beginning of the message. In fact if the receiver is listening on the line at every rising edge of a faster clock (in our case 16 times faster) we can fairly achieve synchrony with an error of about 1/16 in the phase which corresponds to around 22.5 degrees. Since we sample at the middle of every 16 clock periods we don’t get any error due to the asynchrony.

12 RTL Schematic
(1) Note that the external inputs to the entire circuit are rst, Clk, rx, tx.
(2) The inputs for the component RX_UART are rst, Clk, rx and its outputs are rx_data and rx_data_valid.
(3) The inputs for the component TX_UART are rst, Clk, TX_DATA, tx_data_valid and its outputs are tx.
(4) The rx_data and tx_data are HARDWIRED. Similarly rx_data_valid and tx_data_valid.
(5) Note that both the components take their Clk and the rst inputs commonly. This is because they are mapped this way in top_uart.
(6) Going further deeper we find the following complicated design for the generation of serial and combinational elements that help in simulating the vhdl code written. It is as shown in the image below.

13 Simulations

(1) We first made a testbench sending a sequence of data and the first waveform depicts this.
(2) Secondly, we made a testbench for just the receiver module and then for only the transmitter module.
(3) Finally we made a testbench to see how the sampling rate varies with the Clk signal.
14 Top sim

15 Rx sim

16 Tx sim
17 Clock sim

18 Technology Schematic

19 CONTRIBUTIONS

Code: All the parts of our projects were written and developed in the presence of all of the team members. So, we all contributed for the source code part.
20 References Used

(1) http://en.wikipedia.org/wiki/8250_UART
(2) http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
(3) http://en.wikipedia.org/wiki/16550_UART
(4) http://en.wikipedia.org/wiki/Tera_Term
(5) http://download.cnet.com/Tera-Term/3000-20432_4-75766675.html