

Homework-7: UART

ECE-111 Advanced Digital Design Project

Vishal Karna

Outline

❑ **First let's learn UART Concepts**

- What is UART
- UART Tx-Rx Communication Protocol
- UART Data Synchronization and Sampling

❑ **Homework-7a : UART Receiver FSM (UART-RX)**

- FSM Design Requirements
- Simulation waveforms for reference purpose
- FSM State Transition Diagram for reference purpose

❑ **Homework-7b Requirements and Reference Results : UART Tx-Rx Communication System**

- UART TX-RX (i.e. UART Top Module) Design Requirements
- Simulation waveforms for reference purpose

❑ **Homework-7c Requirements and Reference Results : UART Control System**

- UART Control System Design Requirements
- UART Tx Control FSM and UART Rx Control FSM Design Algorithm
- Simulation waveforms and transcript for reference purpose
- FSM State Transition Diagram and State Transition Table for reference purpose

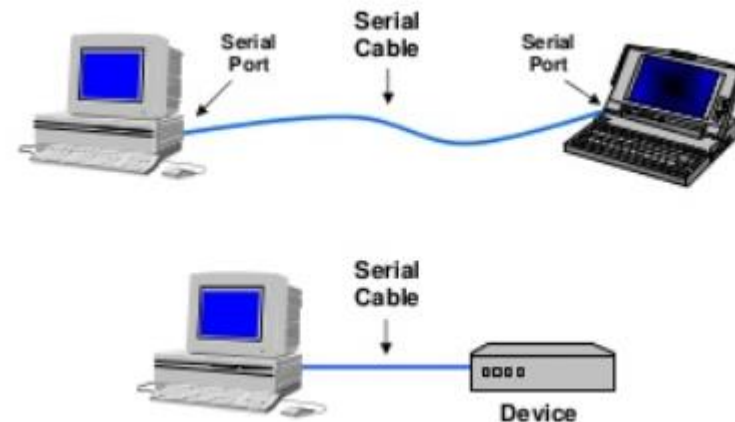
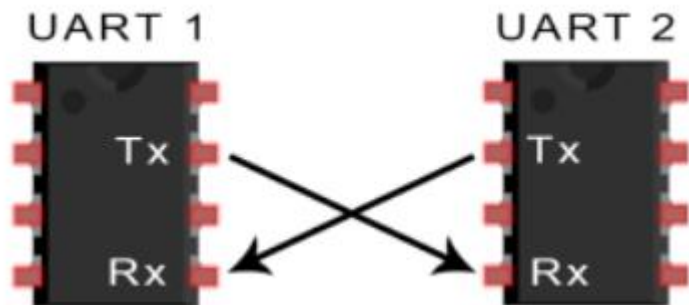
What is UART ?

❑ UART stands for Universal Asynchronous Receiver/Transmitter.

- It's not a communication protocol like SPI/I2C, but a physical circuit in a microcontroller, or a stand-alone IC.
- A UART's main purpose is to transmit and receive serial data asynchronously.
 - It is used where high speed communication is not required
 - PC Serial port is a UART, PC devices such as mice, modem (now replaced by USB)
 - UARTs being used to connect GPS, Bluetooth, RFID card reader modules to Raspberry Pi, Arduino, or other microcontrollers.

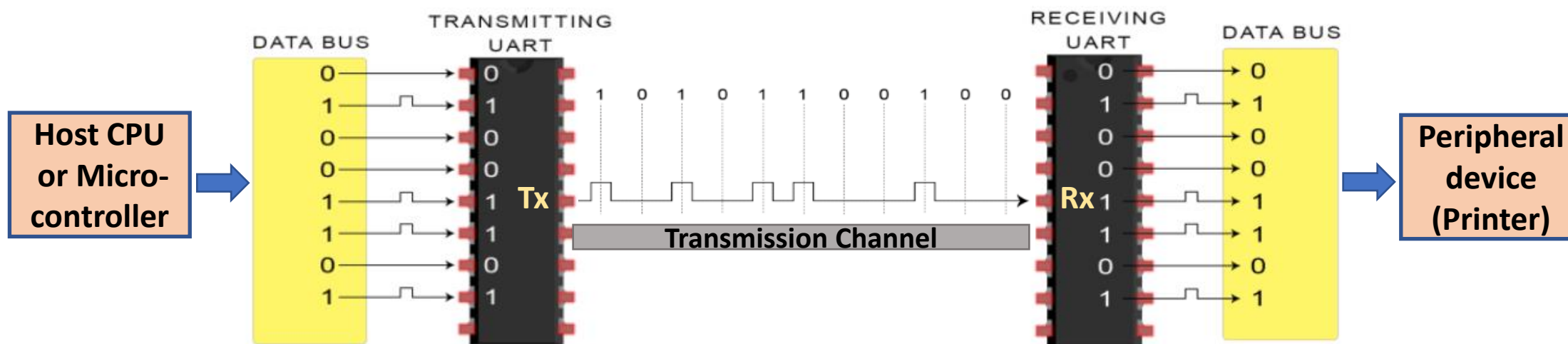
❑ UART does not require much hardware and it is simple to design.

- It only uses two wires to transmit data between devices.
- One wire for either direction (Tx to Rx and Rx to Tx) plus ground wire



UART Tx-Rx Communication

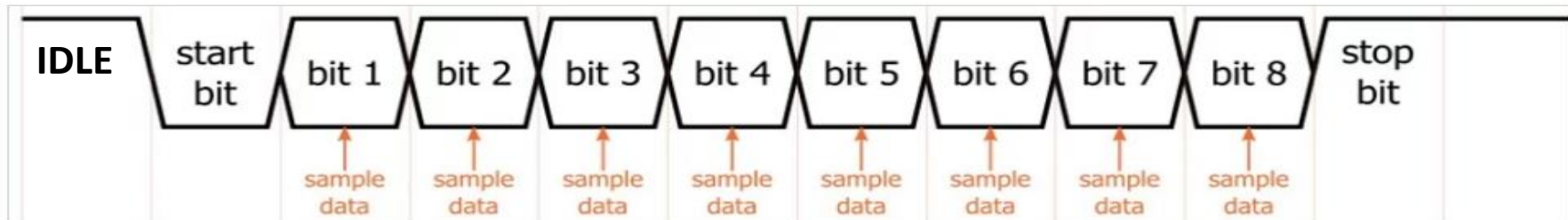
- ❑ Transmitting UART (Tx) converts parallel data received from host (such as CPU) into Serial Data
- ❑ Transmitting UART receives parallel data from host such as CPU, Memory, Microcontroller, etc
- ❑ Transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame
- ❑ Entire packet is sent serially from the transmitting UART to the receiving UART.
- ❑ Receiving UART samples the data line at the pre-configured baud rate
- ❑ Receiving UART discards the start bit, parity bit, and stop bit from the data frame
- ❑ UART converts the serial data back into parallel and transfers it to the data bus and sends to peripheral devices such as printers, scanners, etc



UART Transmitter – Receiver Simplex Communication System

UART Data Synchronization and Sampling

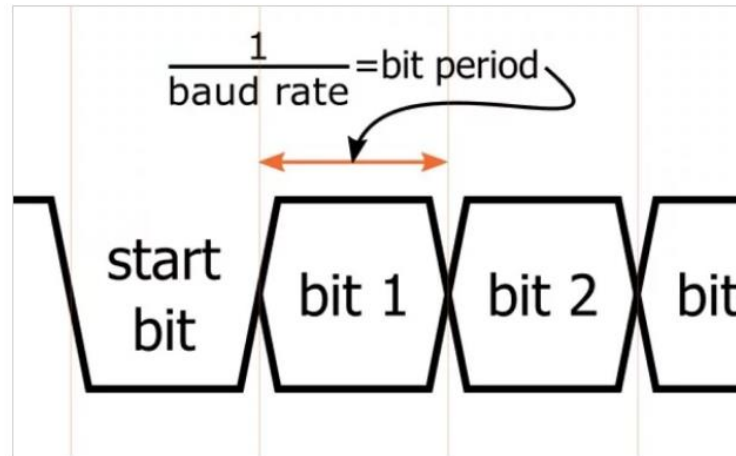
- ❑ When the system is in idle, no data to be sent, the receiver will see logic HIGH at its input
- ❑ The transmitter starts communication with the receiver by sending a start bit.
 - This is signaled by lowering the signal from logic HIGH to logic LOW
 - The start bit is followed by the serial data bits (say 8 bits) including an optional parity bit
 - After N-data bits transmitted, transmission ends with stop bits which is logic High



UART Transmitter Communication

UART Data Synchronization and Sampling

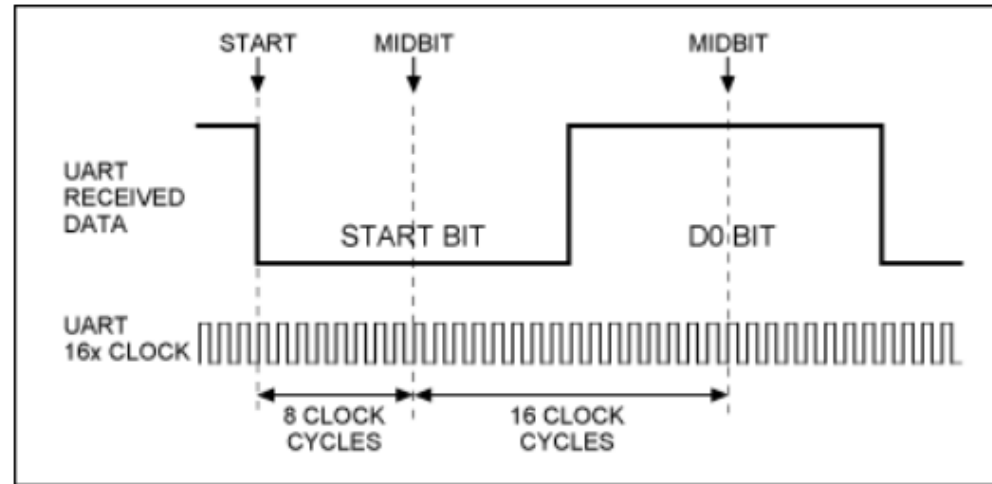
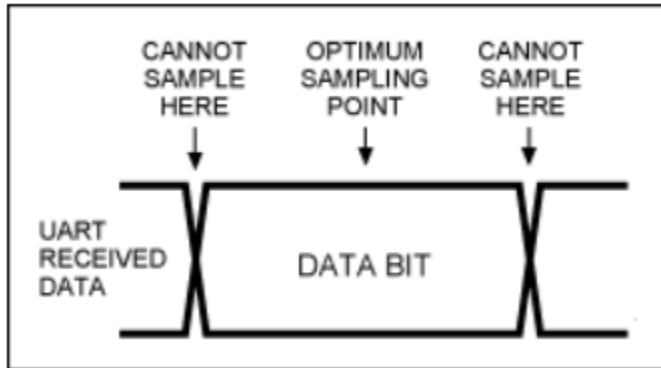
- ❑ In UART there is no clock synchronization between the transmitter(Tx) and Receiver(Rx).
 - Clock frequency is agreed beforehand. This is done by setting the **baud rate** for Rx and Tx module.
 - Both Transmitter and Receiver will have its own independently running clock generation circuit
 - UART communication doesn't work if Tx and Rx configured for different data-transmission frequencies (baud rate)
- ❑ **Baud Rate** : Time (in seconds) required to transmit one bit of digital data.
 - For example, with a 9600-baud system, one bit requires $1/(9600 \text{ bps}) \approx 104.2 \mu\text{s}$ to be transmitted
 - **Baudrates** can be 300, 600, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps
 - **Note : bps** (bits per second)



UART Baud Rate

UART Data Synchronization and Sampling

- ❑ Due to no clock synchronization, receiver needs to ensure incoming bits are sampled when it stable and when they are not changing

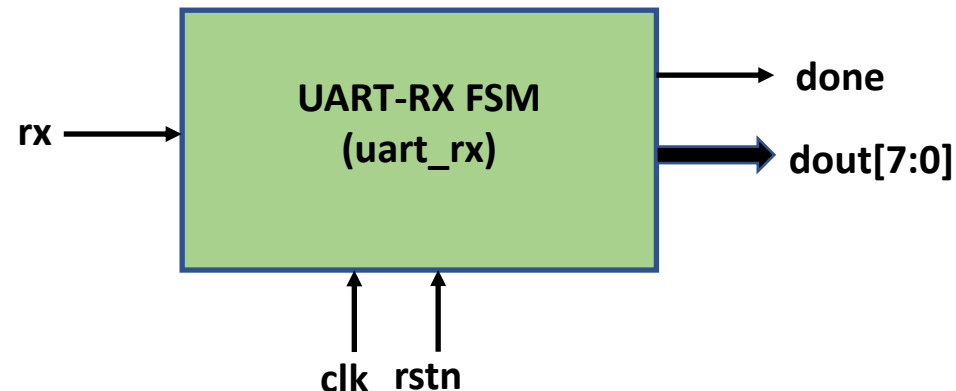


- Oversampling of bits is used to ensure accurate sampling of data. Example, bit will be sampled 16 times.
- Receive process is initiated by the falling edge of the start bit.
- Receiver waits for 8 clock cycles, establish a sampling point near the middle of the start bit period
- Receiver then waits 16 clock cycles, which brings it to the middle of the first data-bit period.
- The first data bit is sampled and stored in the receive register
- Rx module waits another 16 clock cycles before sampling the second data bit
- This process repeats until all data bits have been sampled and stored, and
- Then the rising edge of the stop bit returns the UART interface to its idle state.

Homework-7a : UART Receiver

□ Design UART Receiver (UART-Rx) Module using Finite State Machine in SystemVerilog :

- Develop FSM for UART receiver (uart_rx). Use 1 always_ff block FSM implementation with non-blocking assignment statement
- Develop state transition diagram. Use **binary state encoding** for states
- Testbench will send 8-bit serial data to UART RX.
- Uart RX will convert 8-bit serial data (**rx**) and generate 8-bit parallel data (**dout**)
- done will be set to '1' when 8-bit dout parallel data is available otherwise set to '0'
- Baud rate requirement for **UART-RX** is **115200 bps**. Which means set NUM_CLKS_PER_BIT=434 for **UART RX** clock period = **20ns**.
 - **Bit period** = $(1 / 1152000 \text{ bps}) = 8.68\mu\text{s}$, **NUM_CLKS_PER_BIT** = $8.68\mu\text{s} / 20\text{ns} = 434$
- However for faster simulation results design **UART RX** with NUM_CLKS_PER_BIT=16 instead of 434
- Use testbench provided for **uart_rx** receiver and simulate **FSM** to confirm its behavior.
- Assume below mentioned primary port list for uart_rx module :
 - **input** clk, rstn : posedge clk and synchronous active low reset
 - **input** rx : 1-bit serial data input
 - **output** [7:0] dout, **output** done.



Homework-7a : UART Receiver

❑ How many states are required to design UART-RX FSM ?

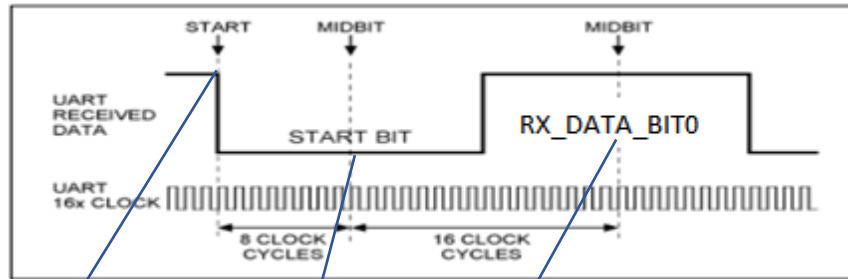
- 1 state for IDLE
- 1 state bit to receive start bit (wait for rx == 0). "rx" is input serial port of uart_rx FSM.
- 8 states to receive 8 data bits serially (i.e. 1 data bit state to receive 1-bit of data)
- 1 state bit to receiver stop bit (wait for rx == 1)
- **Total number of states = 1 IDLE state + start bit state + 8 data bit states + 1 stop bit state = 10 states for FSM**

❑ State encoding variables :

- RX_IDLE = 4'b0000 → IDLE State
- RX_START_BIT = 4'b0001 → wait for "rx == 0" which indicates start bit. "rx" is input serial port of uart_rx FSM
- RX_DATA_BIT0 = 4'b0010 → 1st data bit received at "rx" input port of uart_rx FSM
- RX_DATA_BIT1 = 4'b0011 → 2nd data bit received at "rx" input port of uart_rx FSM
- RX_DATA_BIT2 = 4'b0100 → 3rd data bit received at "rx" input port of uart_rx FSM
- RX_DATA_BIT3 = 4'b0101 → 4th data bit received at "rx" input port of uart_rx FSM
- RX_DATA_BIT4 = 4'b0110 → 5th data bit received at "rx" input port of uart_rx FSM
- RX_DATA_BIT5 = 4'b0111 → 6th data bit received at "rx" input port of uart_rx FSM
- RX_DATA_BIT6 = 4'b1000 → 7th data bit received at "rx" input port of uart_rx FSM
- RX_DATA_BIT7 = 4'b1001 → 8th data bit received at "rx" input port of uart_rx FSM
- RX_STOP_BIT = 4'b1010 → wait for "rx == 1" which indicates stop bit

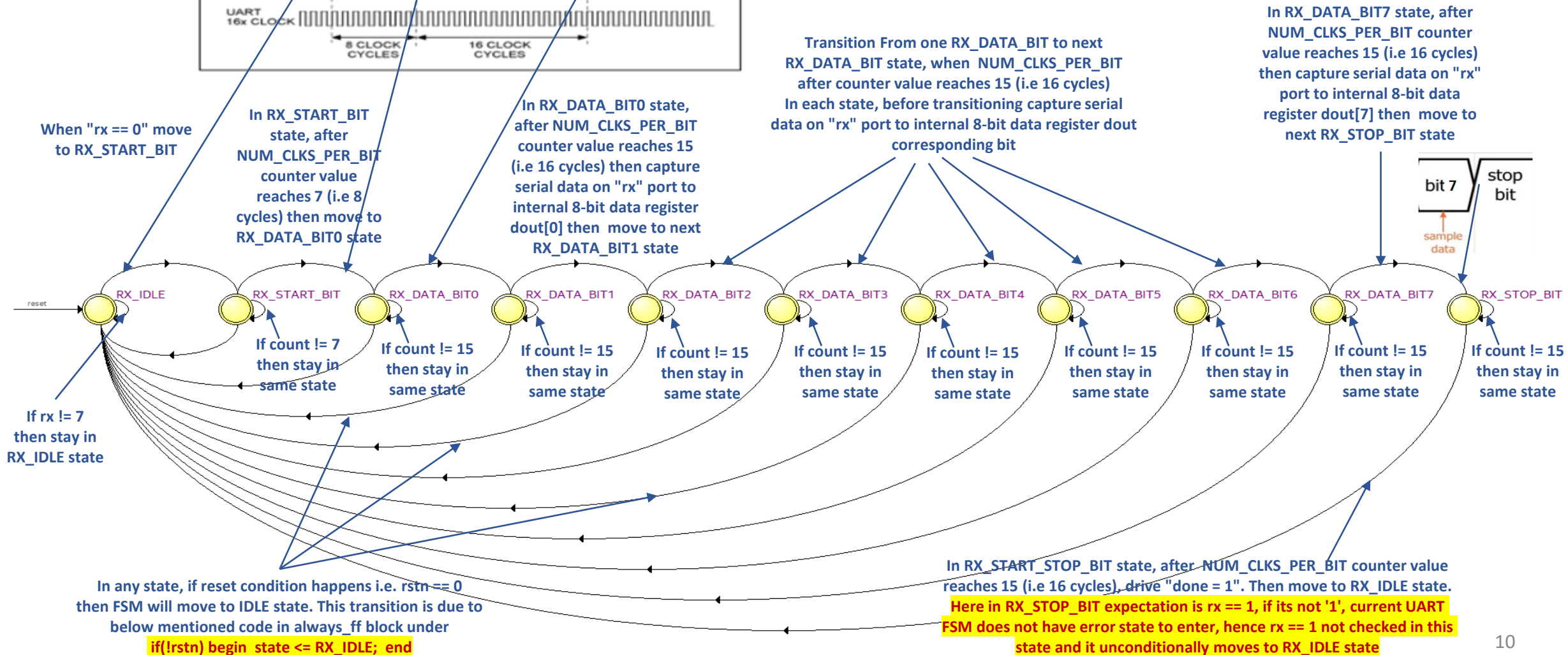
Homework-7a : UART Receiver

❑ UART RX FSM State Transition Diagram For Reference Purpose



In RX_DATA_BIT 0 when counter reaches value "15" capture data i.e. `dout[0] <= rx;`
 In RX_DATA_BIT 1 when counter reaches value "15" capture data i.e. `dout[1] <= rx;`

 In RX_DATA_BIT 7 when counter reaches value "15" capture data i.e. `dout[7] <= rx;`
 End of RX_DATA_BIT7 state, dout register has 8-bit parallel data available

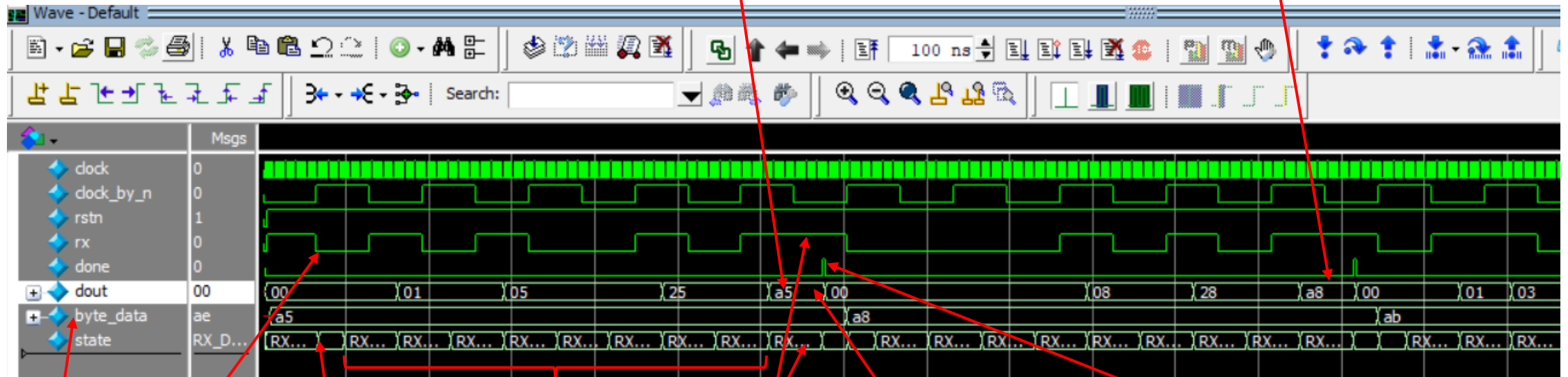


Homework-7a : UART Receiver

❑ UART Receiver (UART RX) Simulation Waveform View-1

First Byte
Received by
uart_rx : 8'ha5

Second Byte
Received by
uart_rx : 8'ha8



rx=1 to 0
indicates
start bit

RX_START_BIT
state

RX_DATA_BIT0
to 7 states

Rx=1 indicates
stop bit and
RX_STOP_BIT
state

done is set to '1' once all start
bit, 8 data bits and stop bit are
serially transmitted

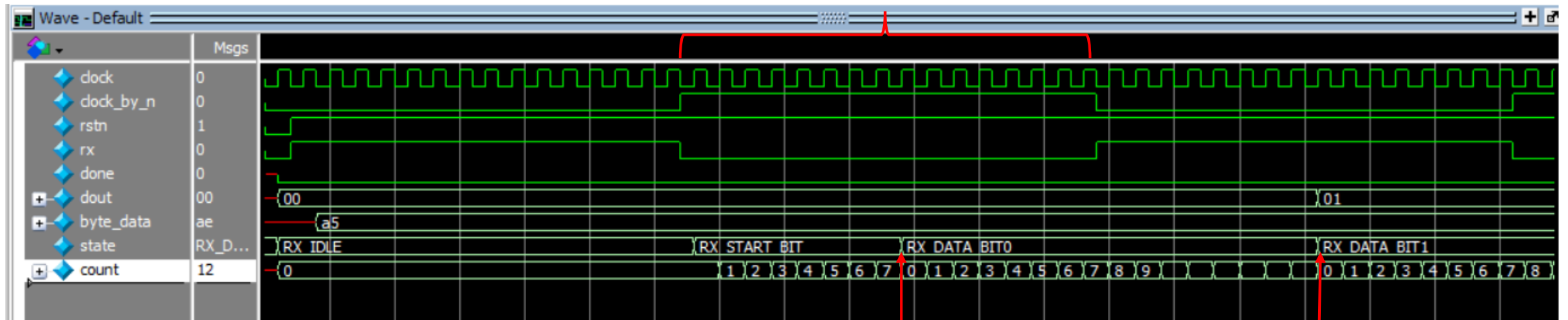
byte_data is testbench signal
which should match dout signal
value when done is set to '1'

parallel 8-bit dout output data

Homework-7a : UART Receiver

❑ UART Receiver (UART RX) Simulation Waveform View-2

If NUM_CLKS_PER_BIT=16, then there will be 16 clocks per rx serial bit

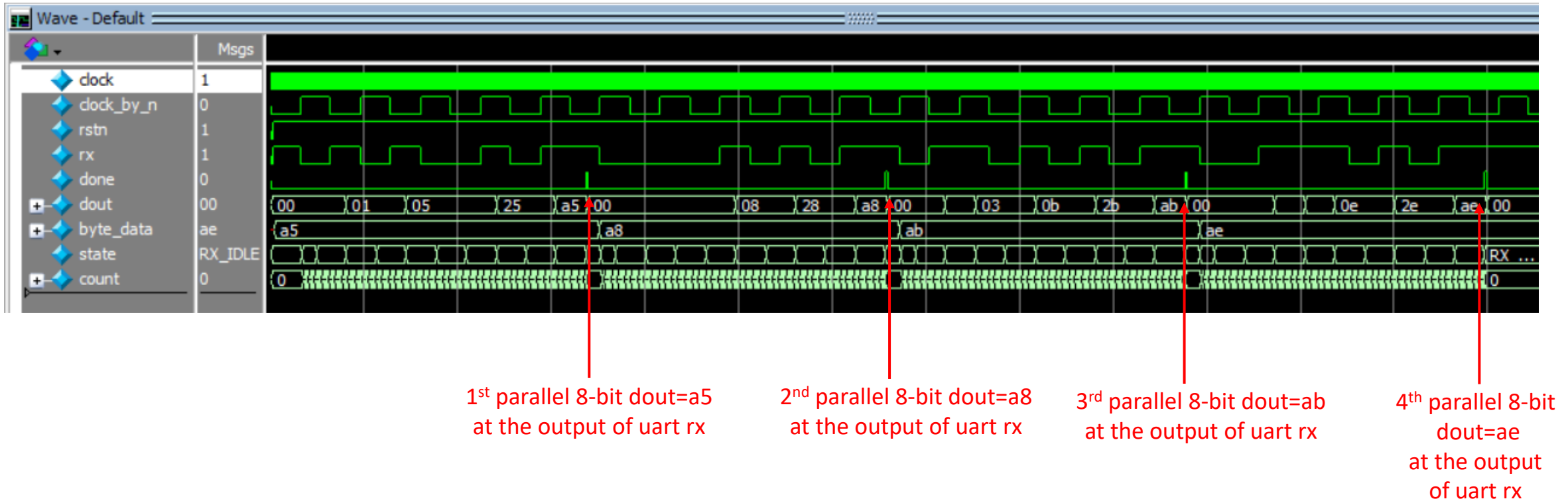


When count == 7, sample rx=0 start bit at midpoint

When count == 15, sample first data bit at midpoint

Homework-7a : UART Receiver

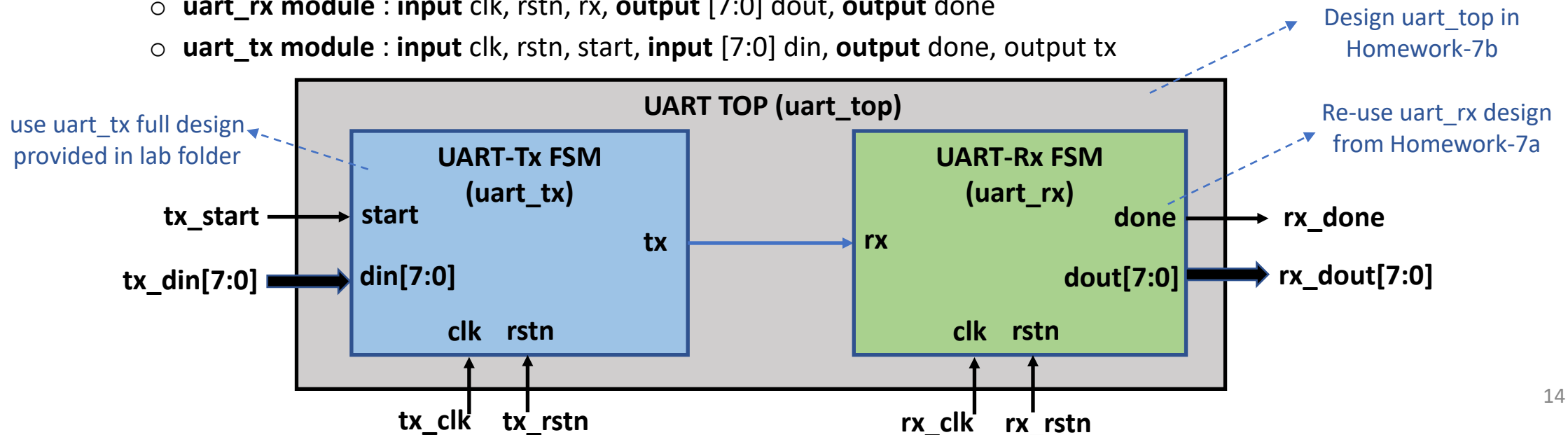
❑ UART Receiver (UART RX) Simulation Waveform View-3



Homework-7b : UART TX-RX Communication System

❑ Requirements for UART Tx-Rx communication system (uart_top)

- Create **uart_top** module with both **uart_tx** and **uart_rx** modules instantiated and make connections between these two modules
- Re-use **uart_rx** module from Homework-7a and use **uart_tx** module full implementation provided in Lab folder
- Review **uart_tx** design code provided in lab folder to understand UART transmitter design
- Review resource utilization including total flipflops and combination ALUT's in report
- Simulate UART Top including UART TX and RX implementation with testbench provided
- Review simulation waveform and explain results for UART Top module
- Primary port list for **uart_tx**, **uart_rx** and **uart_top** modules :
 - **uart_top** : **input** tx_clk, tx_rstn, rx_clk, rx_rstn, tx_start, **input** [7:0] tx_din, **output** rx_done, **output** [7:0] rx_dout
 - **uart_rx module** : **input** clk, rstn, rx, **output** [7:0] dout, **output** done
 - **uart_tx module** : **input** clk, rstn, start, **input** [7:0] din, **output** done, **output** tx



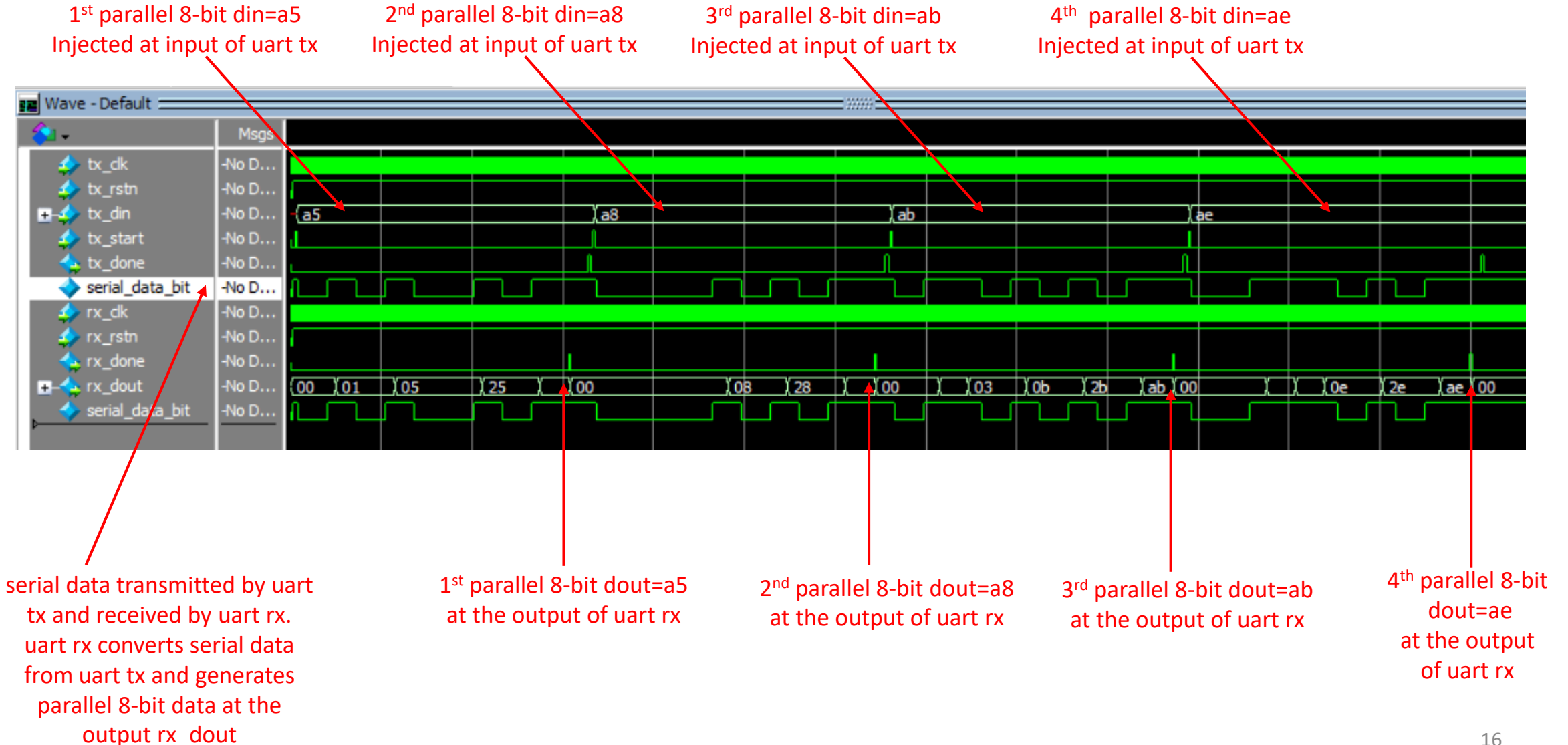
Homework-7b : UART TX-RX Communication System

❑ Requirements (Continued ...)

- Review testbench code provided for uart_top module which does below mentioned :
 - Transmit 8-bit characters tx_din with tx_start asserted (say : 8'hA5, 8'hA8, 8'AB, 8'hAE, etc is generated on tx_din)
 - Receive 8-bit characters rx_dout with rx_done asserted (and same 8'hA5, 8'hA8, 8'AB, 8'hAE in same order is expected to receive on rx_dout). Confirm this in waveform by reviewing it.
 - In testbench tx_clk and rx_clk are generated with 20ns clock period
 - **Optional** : In testbench there is self-checking mechanism where 8bit input tx_din is compared and 8bit output rx_dout whenever it is transmitted and received.

Homework-7b : UART TX-RX Communication System

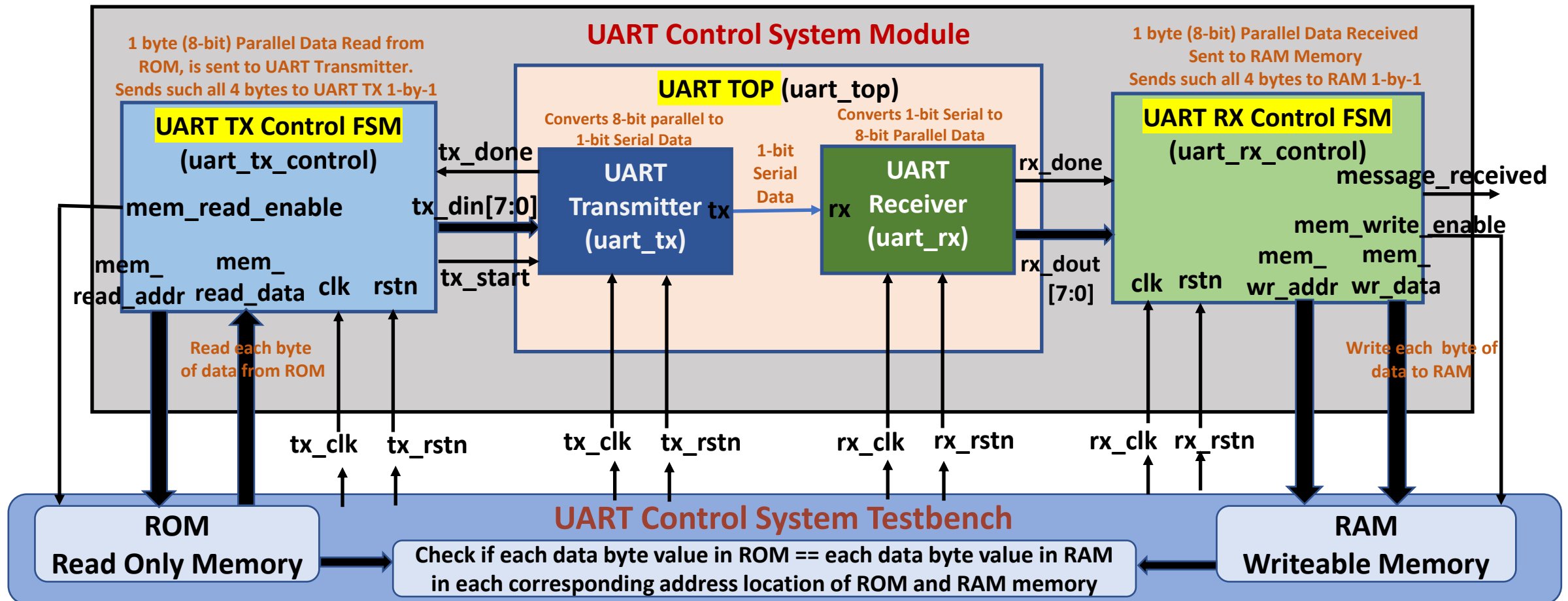
❑ UART Top (i.e. UART TX-RX) Simulation Waveform



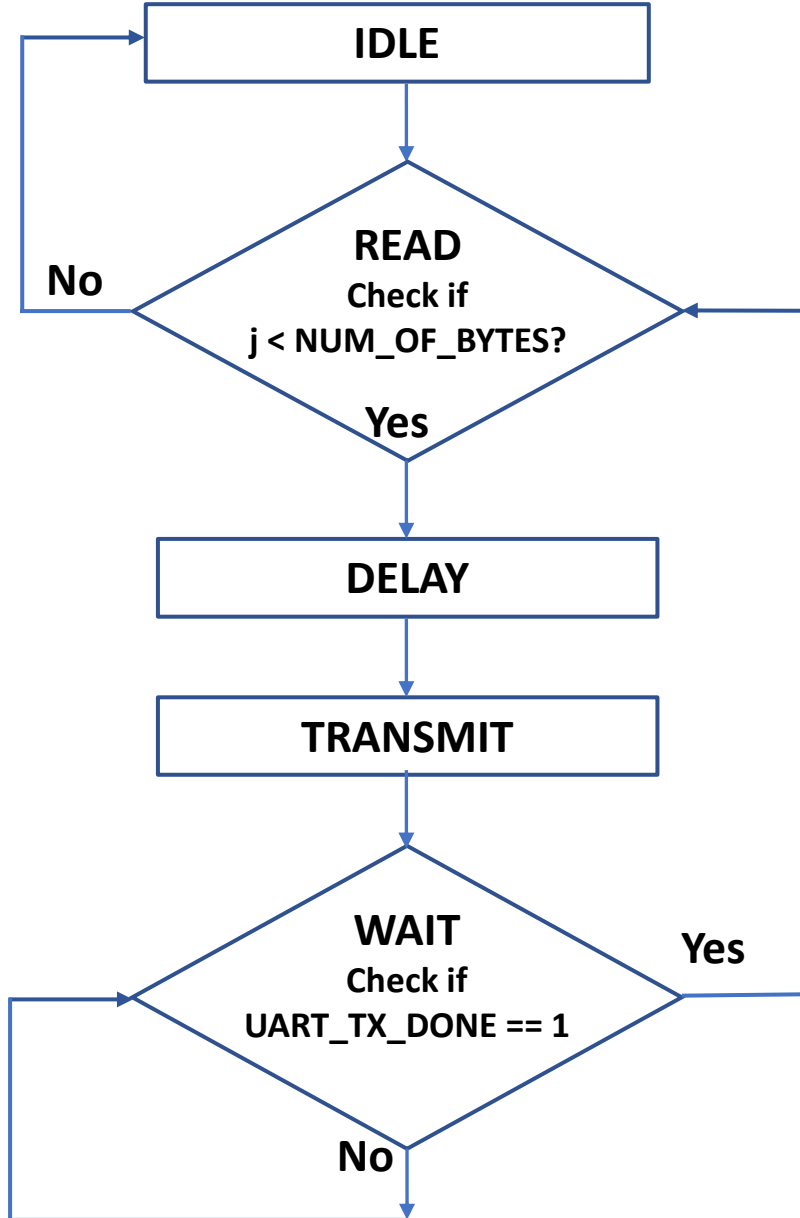
Homework-7c : UART Control System

□ Design UART Control System Module Including below mentioned :

- Design UART TX Control FSM which will read 4 data bytes from **ROM** in testbench and send it to UART-Tx one by one
- Design UART RX Control FSM which will receive 4 data bytes one by one and writes each byte in **RAM** in testbench
- Instantiate UART_TOP module, UART TX Control module and UART RX Control module within UART Control System Module
- Re-use `uart_top` module and `uart_tx`, `uart_rx` from homework-7a and homework-7b



Homework-7c : UART Control System



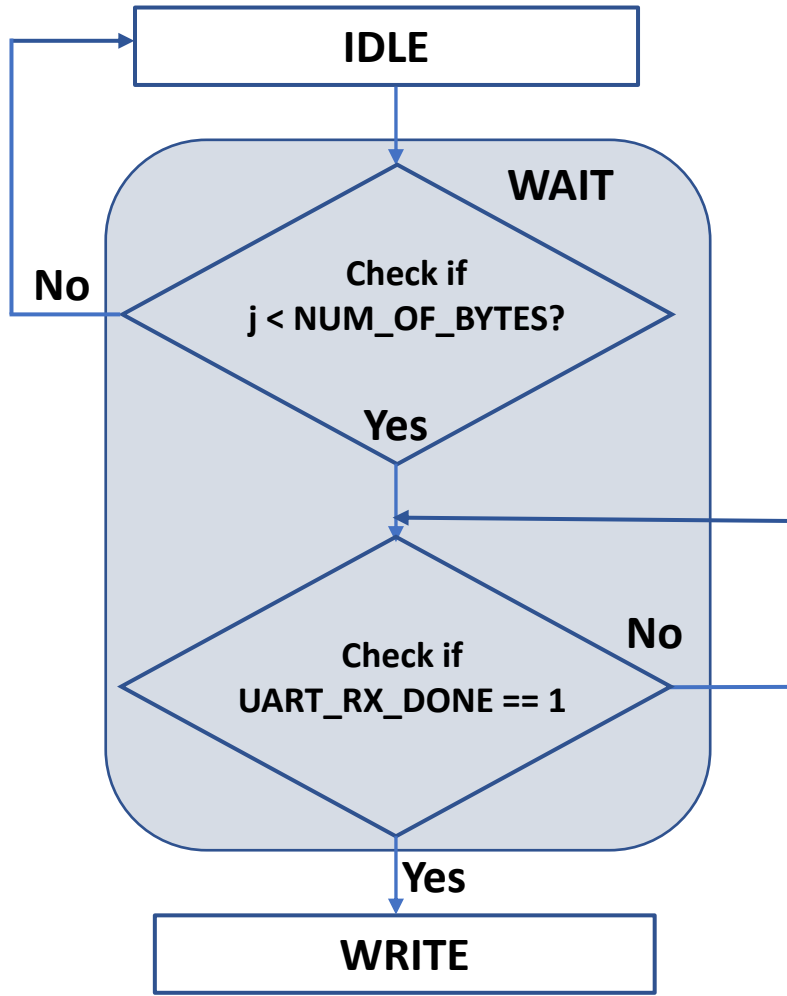
❑ UART TX Control FSM Purpose and Design Algorithm

- **Purpose** : Forms an interface between UART-TX and **ROM memory**. Responsible to receive 4 bytes of data, 1-by-1 from ROM memory and controls UART Transmitter to send each byte of data serially to UART Receiver.
- **In READ FSM State** :
 - Checks if all 4 bytes of data have been read and sent to UART Transmitter.
 - If not, then reads 1 byte (i.e. 8-bit) of data from Read Only Memory (**ROM**) by generating memory read enable '1' and memory address signals to **ROM** and then moves to **DELAY** state. **ROM** resides in testbench
 - If all 4 bytes are sent to UART Transmitter, then goes back to **IDLE** state
- **In DELAY FSM State** :
 - Stays in this state for 1 clock cycle for read data to be made available from **ROM** (**Note** : data will be available on `mem_read_data` bus, 1 clock cycle after `mem_read_enable = 1`). And then move to Transmit state.
- **In TRANSMIT State** :
 - Sends 1 byte of data which is now available from **ROM** on `mem_read_data` to UART Transmitter's `din` port.
 - Generates `uart_tx_start = 1` to UART transmitter to start parallel to serial conversion and start serial transmission of each of the data bits to UART receiver. And then moves to **WAIT** state
- **In WAIT State** :
 - Waits for UART Transmitter to indicate that it has completed transmission of 1 byte of data to UART receiver and then moves to **READ** state after increment 'j' which is total byte counter.

Homework-7c : UART Control System

□ UART RX Control FSM Purpose and Design Algorithm

- **Purpose** : Forms a interface between UART-RX and **RAM memory**. Responsible to receive 4 bytes of data, 1-by-1 from UART Receiver and then write each byte of data to **RAM** memory.
- **In WAIT FSM State :**
 - Checks if all 4 bytes of data have been received from UART Receiver
 - If not, then waits for next byte of data from UART Receiver
 - Once 1-byte (i.e. 8 bit) of data in received from UART Receiver, then it stores the received data in a local variable and then moves to **WRITE** state
 - If all 4 bytes of data have been received from UART Receiver, it then generates **message_received** = 1 to indicate all 4-bytes have been received and then moves to **IDLE** state
- **In WRITE FSM State :**
 - Writes 1 byte (i.e. 8-bit) of data to **RAM memory** by generating memory write enable '1' and memory address signals to **RAM** and then moves to **WAIT** state after incrementing 'j' number of byte received counter.
 - **Note** : **RAM** resides inside testbench.

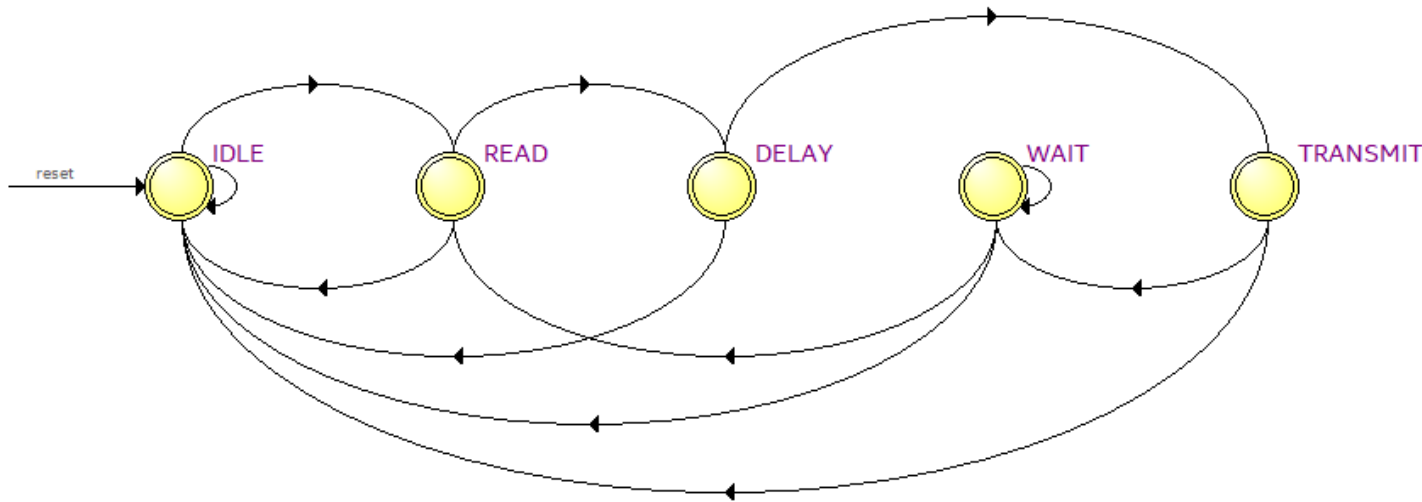


Homework-7c : UART Control System

UART TX Control FSM Reference FSM State Transition Diagram and State Transition Table For Reference Purpose

- Note :** Student FSM can have variations if there is alternate algorithm implemented. It will be accepted as long simulation has passed (see messages in modelsim transcript window)

UART TX Control FSM



	Source State	Destination State	Condition
1	DELAY	IDLE	(!rstn)
2	DELAY	TRANSMIT	(rstn)
3	IDLE	IDLE	(!rstn)
4	IDLE	READ	(rstn)
5	READ	DELAY	(LessThan0).(rstn)
6	READ	IDLE	(!LessThan0) + (LessThan0).(!rstn)
7	TRANSMIT	IDLE	(!rstn)
8	TRANSMIT	WAIT	(rstn)
9	WAIT	IDLE	(!rstn)
10	WAIT	READ	(uart_tx_done).(rstn)
11	WAIT	WAIT	(!uart_tx_done).(rstn)

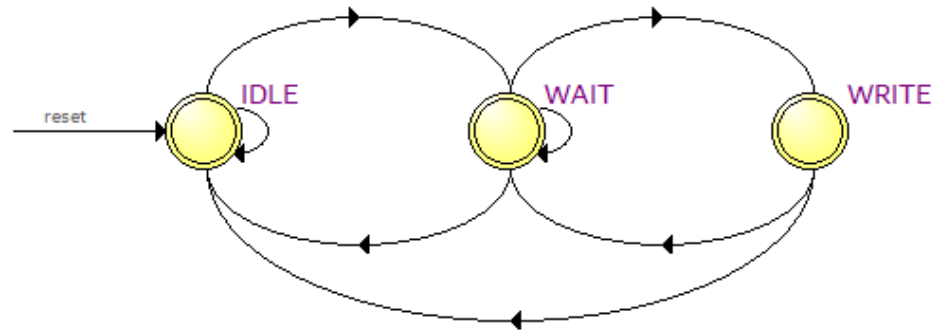
Total : 5 states required for UART TX Control FSM

- IDLE = 3'b000, READ = 3'b001, DELAY = 3'b010, WAIT = 3'b011, TRANSMIT
- For encoding 5 FSM state : 3 bits are required for state encoding variables (i.e. 3 flipflops in hardware for state bits)

Homework-7c : UART Control System

- ❑ **UART RX Control FSM Reference FSM State Transition Diagram and State Transition Table For Reference Purpose**
- **Note :** Student FSM can have variations if there is alternate algorithm implemented. It will be accepted as long simulation has passed (see messages in modelsim transcript window)

UART RX Control FSM



	Source State	Destination State	Condition
1	IDLE	WAIT	(rstn)
2	IDLE	IDLE	(!rstn)
3	WAIT	WRITE	(uart_rx_done),(LessThan0),(rstn)
4	WAIT	WAIT	(!uart_rx_done),(LessThan0),(rstn)
5	WAIT	IDLE	(!LessThan0) + (LessThan0),(!rstn)
6	WRITE	WAIT	(rstn)
7	WRITE	IDLE	(!rstn)

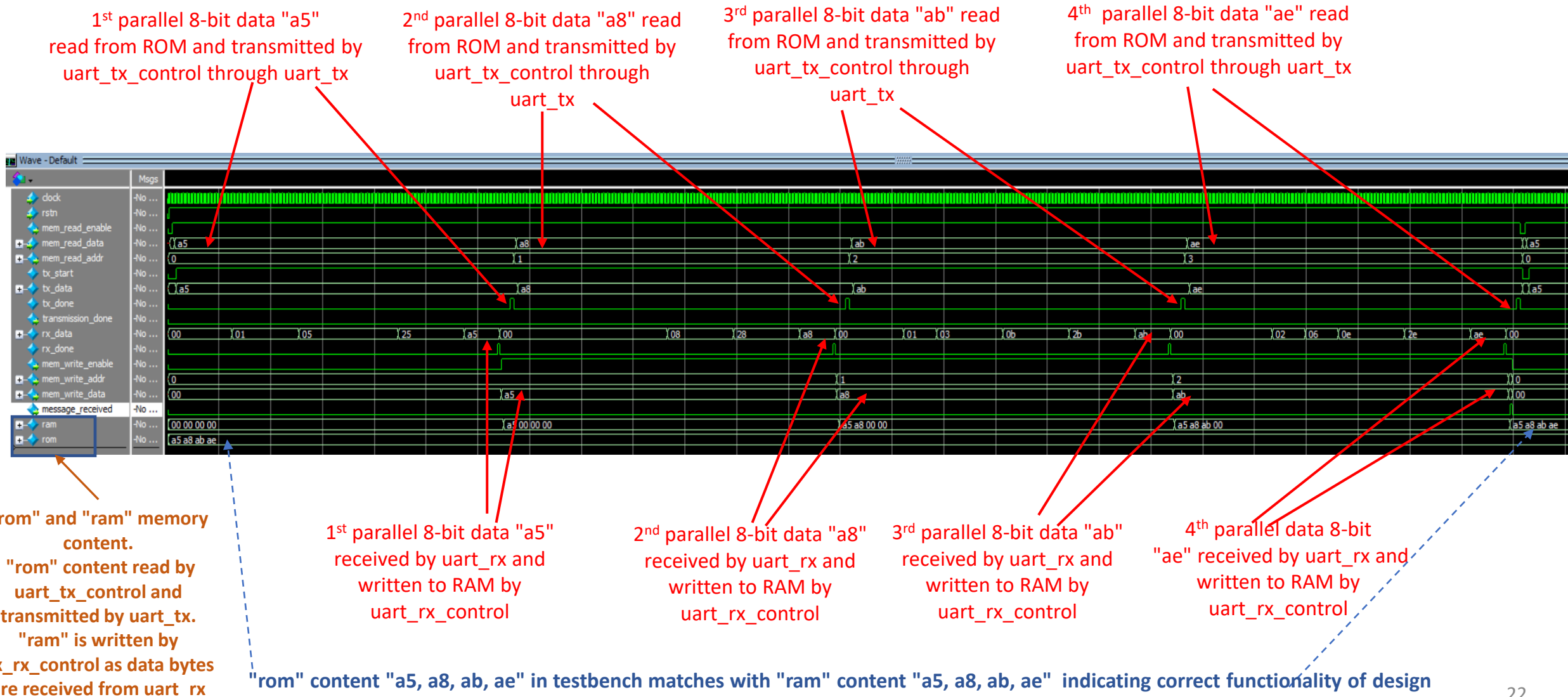
Total : 3 states required for UART RX Control FSM

- IDLE = 2'b00, WAIT = 2'b01, WRITE = 2'b10
- For encoding 5 FSM state : 2 bits are required for state encoding variables (i.e. 2 flipflops in hardware for state bits)

Homework-7c : UART Control System

uart_control_system module simulation snapshot for reference purpose

- All signals shown below in waveform snapshot are at uart_control_system level with the exception of two unpacked arrays : "ram" and "rom"
- "ram" and "rom" are part of testbench module are displaying it's memory content.



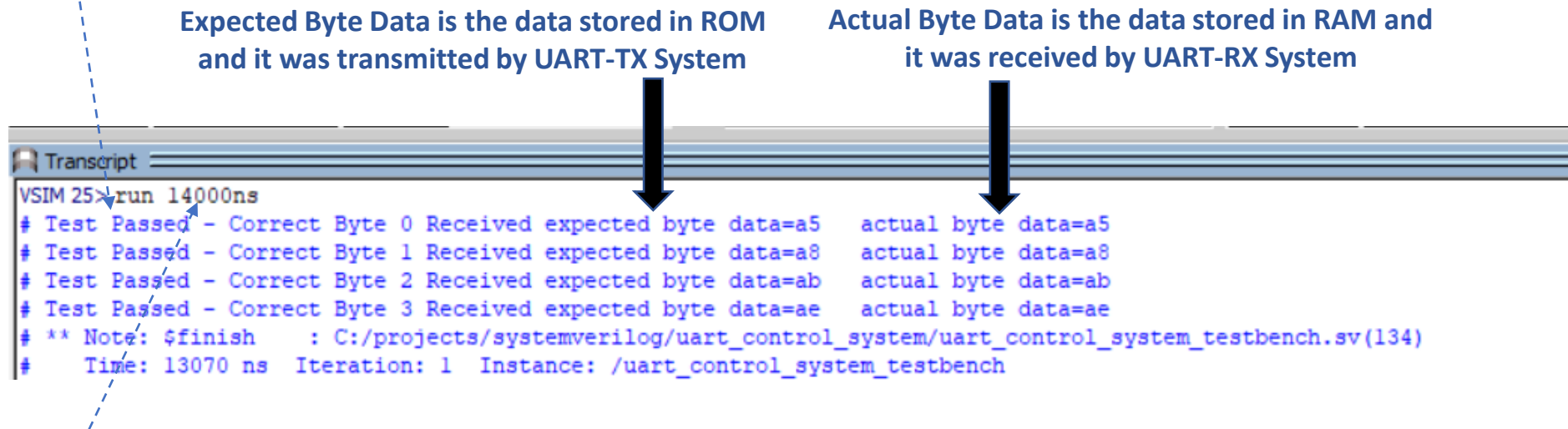
Homework-7c : UART Control System

❑ UART Control System Modelsim Simulation Transcript For Reference Purpose

- There are 4 bytes of data store in ROM memory in testbench. These are :
 - Byte-1 : 8'ha5
 - Byte-2 : 8'ha8
 - Byte-3 : 8'hab
 - Byte-4 : 8'hae
- At the end of simulation, check if byte values in each corresponding location in RAM are matching with ROM
 - To achieve this see Modelsim simulation transcript window which will print RAM and ROM content from testbench which has checking mechanism code
 - If "**Test Passed**" message is printed for all 4 bytes comparison, then it indicates that UART Control System Design is correctly implemented in SystemVerilog.

Expected Byte Data is the data stored in ROM and it was transmitted by UART-TX System

Actual Byte Data is the data stored in RAM and it was received by UART-RX System



```
Transcript
VSIM 25>run 14000ns
# Test Passed - Correct Byte 0 Received expected byte data=a5    actual byte data=a5
# Test Passed - Correct Byte 1 Received expected byte data=a8    actual byte data=a8
# Test Passed - Correct Byte 2 Received expected byte data=ab    actual byte data=ab
# Test Passed - Correct Byte 3 Received expected byte data=ae    actual byte data=ae
# ** Note: $finish      : C:/projects/systemverilog/uart_control_system/uart_control_system_testbench.sv(134)
#   Time: 13070 ns  Iteration: 1  Instance: /uart_control_system_testbench
```

- **Note** : UART Control System Tests needs to be run longer for around **14000ns** to **14600ns** to receive all 4 data bytes at the UART Receiving control system

Homework-7c : UART TX-RX Communication System

❑ Requirements :

- Create `uart_tx_control` and `uart_rx_control` design modules using template code provided
- Create `uart_control_system` top level module with `uart_tx_control`, `uart_rx_control` and `uart_top` modules instantiated and make connections between these modules
 - Re-use `uart_top`, `uart_tx` and `uart_rx` code from Homework7a and 7b
- Synthesis `uart_control_system` module and review source utilization including total flipflops/registers and combination ALUT's
- Simulate `uart_control_system` including `uart_tx_control` and `uart_rx_control` designs using testbench provided. Review simulation result.
 - No separate testbench code provided for `uart_tx_control` and `uart_rx_control` designs. These will be verified as part of `uart_control_system` design simulation
- When creating `uart_control_system` project in Quartus and when simulating using Modelsim, add all these files :
 - `uart_rx.sv`, `uart_tx.sv`, `uart_top.sv`, `uart_tx_control.sv`, `uart_rx_control.sv` and `uart_control_system.sv`
- Primary port list for **`uart_tx_control` module** :
 - input logic clock, `rstn` : posedge clock and synchronous active low reset
 - output logic[3:0] `mem_write_addr` : memory write address generated to write RAM in testbench
 - output logic[7:0] `mem_write_data` : memory write data generated to write data to RAM in testbench
 - output logic `mem_write_enable` : memory write enable generated to enable writing to RAM in testbench
 - input logic[7:0] `mem_read_data` : memory read data returned from ROM in testbench
 - output logic[3:0] `mem_read_addr` : memory read address generated to read ROM in testbench
 - output logic `mem_read_enable` : memory read enable generated to enable reading of ROM in testbench
 - output logic `transmission_done` : indicates all data bytes have been transmitted by uart tx control system
 - output logic `message_received` : indicates all data bytes have been received by uart rx control system

Homework-7c : UART TX-RX Communication System

❑ Requirements : (continued)

▪ Primary port list for **uart_rx_control module** :

- input logic clk, rstn : posedge clock, synchronous active low reset
- output logic[7:0] mem_write_data : output data byte to be written to RAM memory
- output logic [3:0] mem_write_addr : address to memory to write data byte received by uart_tx fsm
- output logic mem_write_enable : if set to '1', write data byte to memory in testbench
- input logic uart_rx_done : comes from uart_rx FSM as indication that parallel data byte is received and available to be written in testbench RAM memory
- input logic [7:0] uart_rx_data : parallel data byte received from uart_rx FSM
- output logic message_received : indicates that all data bytes are received by uart_rx FSM and written to RAM memory in testbench

○ Primary port list for **uart_tx_control module** :

- input logic clk, rstn : posedge clock, synchronous active low reset
- input logic[7:0] mem_read_data : input data bytes from memory
- output logic [3:0] mem_read_addr : address to memory to read input data bytes
- output logic mem_read_enable : if set to '0', read data bytes from memory
- output logic transmission_done : set to '1' by FSM when all data bytes are transmitted to receiver
- input logic uart_tx_done : comes from uart_tx FSM as indication that data byte requested by tx control FSM has been transmitted to uart receiver
- output logic [7:0] uart_tx_data : data byte sent to uart_tx FSM to transmit serially data to uart_rx
- output logic uart_tx_start : tx control FSM instructs uart_tx FSM to start data byte transmission to uart_rx

Homework-7a, 7b, 7c : Report Submission and Lab7 Folder

❑ Homework Report should include :

- SystemVerilog design code snapshot for :
 - UART Receiver (uart_rx), UART Top (uart_top), UART Control System (uart_control_system)
- Synthesis resource usage generated from Quartus for all three 7a, 7b, 7c designs
- Simulation snapshot and explain simulation result for all three 7a, 7b, 7c designs
- Modelsim transcript window with test passed message for for all three 7a, 7b, 7c designs
- FSM state transition diagram and state transition table snapshot generated from Quartus for 7a, 7c only
- Explanation of FPGA resource usage in report is not required.

❑ Lab7 folder includes :

- uart_top folder which includes
 - uart_rx sub folder with uart_rx design template code and full testbench. Students needs to complete the design code.
 - uart_tx sub folder with uart_tx full design code and full testbench. Students should review UART transmitter design code and also simulate for understanding purpose
 - uart_top design template code and full testbench code
- uart_control_system folder which includes :
 - uart_tx_control, uart_rx_control and uart_control_system design template code. Student needs to complete both these design code.
 - Full testbench code for uart_control_system. There is no separate testbench for uart_tx_control and uart_rx_control. Instead these modules will be verified as part of uart_control_system simulation as both uart_tx_control and uart_rx_control are instantiated inside uart_control_system
- For learning purpose, student can change the stimulus in initial block in testbench files for 7a, 7b, 7c