ECE-111: Advanced Digital Design Project: Homework 6

Designs	Vending Machine (Moore and Mealy), Synchronous FIFO, UARTs
Deadline	Nov 13, 2024 at 11:59pm
Max. late days	2 (20% grade reduction per day)

Overview

There will be two parts for this homework. In **homework-6a** you will design a synthesizable SystemVerilog Model of a Vending Machine. Here you will be designing the Vending Machine in both a Mealy and a Moore model. In **homework-6b**, you will develop a synthesizable SystemVerilog code for Synchronous FIFO. In homework-6c, you will develop a synthesizable SystemVerilog code for a UART system. For a UART system, you need to design UART Reciever FSM (UART_RX), UART Top Module (UART Tx-Rx Communication System), and UART Control System.

We have provided a folder called **Lab6.zip** which contains the following:

Homework-6a:

1. vending_machine_moore.sv template code for design
2. vending_machine_moore_testbench.sv full code
3. vending_machine_mealy.sv template code for design
4. vending_machine_mealy_testbench.sv full code

Homework-6b:

1. sync_fifo.sv design template code
2. sync_fifo_testbench.sv full code
3. dual_port_ram.sv design template code

Homework-6c:

Homework-6c includes

1. uart_top folder which includes
    - uart_rx sub folder which consists of uart_rx design template code and full testbench. Students are required to complete this code.
    - uart_tx sub folder which consists of uart_tx full design code and full testbench. Students should review this code and also simulate it for understanding the transmitter.

2. uart_control system folder which includes
    - uart_tc_control, uart_rx_control and uart_control_system design template code. Students are required to complete all of these codes.
    - 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 anduart_rx_control are instantiated inside uart_control_system.

Assignment Tasks

This assignment requires you to complete the following tasks:

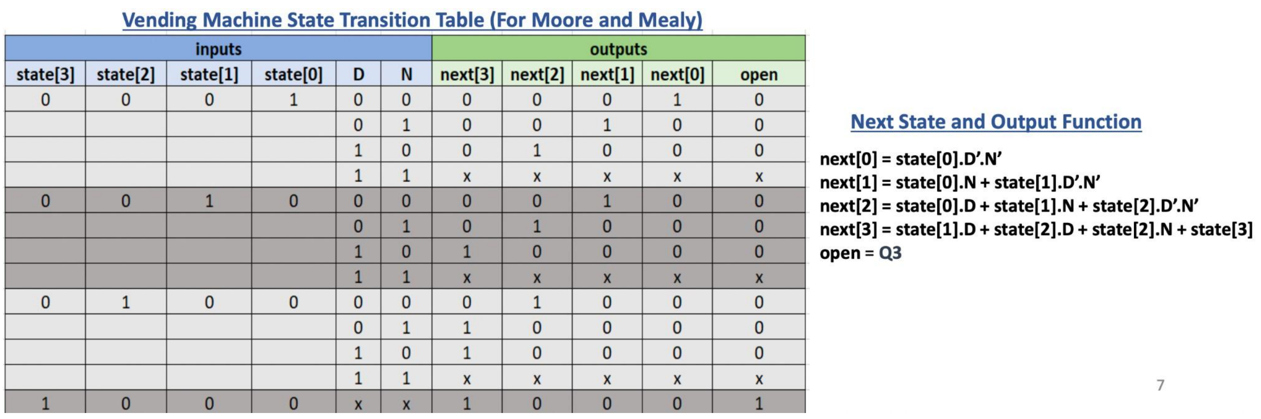
Recommended Tasks:

- Go over discussion video and discussion slides that go over this homework.

<u>For homework-6a:</u>

- **For Vending Machine develop SystemVerilog code for Moore and Mealy FSM.**
    - Use one-hot encoding for state variables.
    - Review Vending state transition table and Moore FSM state transition diagram(Discussion 6) for FSM code development.
    - Design state transition diagram for Mealy implementation.

- Synthesize and review RTL netlist schematic, state machine viewer and resource usage.
- Simulate both Moore and Mealy implementations using the testbenches provided and review waveforms.
- Design's top SystemVerilog module name should be vending_machine_moore and vending_machine_mealy.



Vending Machine State Transition Table (For Moore and Mealy)

inputs						outputs				
state[3]	state[2]	state[1]	state[0]	D	N	next[3]	next[2]	next[1]	next[0]	open
0	0	0	1	0	0	0	0	0	1	0
				0	1	0	0	1	0	0
				1	0	0	1	0	0	0
				1	1	x	x	x	x	x
0	0	1	0	0	0	0	0	1	0	0
				0	1	0	1	0	0	0
				1	0	1	0	0	0	0
				1	1	x	x	x	x	x
0	1	0	0	0	0	0	1	0	0	0
				0	1	1	0	0	0	0
				1	0	1	0	0	0	0
				1	1	x	x	x	x	x
1	0	0	0	x	x	1	0	0	0	1

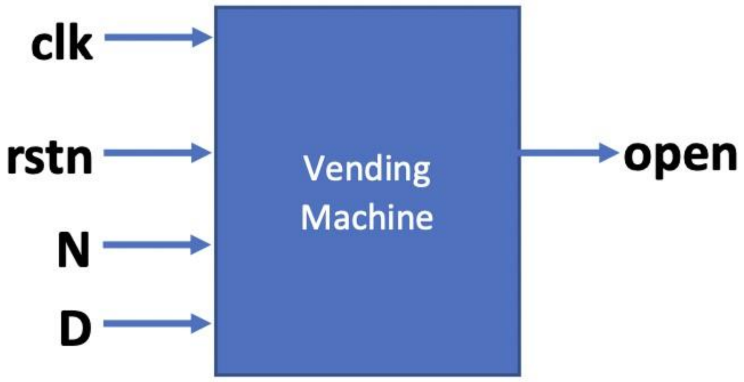
Next State and Output Function

next[0] = state[0].D'.N'
next[1] = state[0].N + state[1].D'.N'
next[2] = state[0].D + state[1].N + state[2].D'.N'
next[3] = state[1].D + state[2].D + state[2].N + state[3]
open = Q3

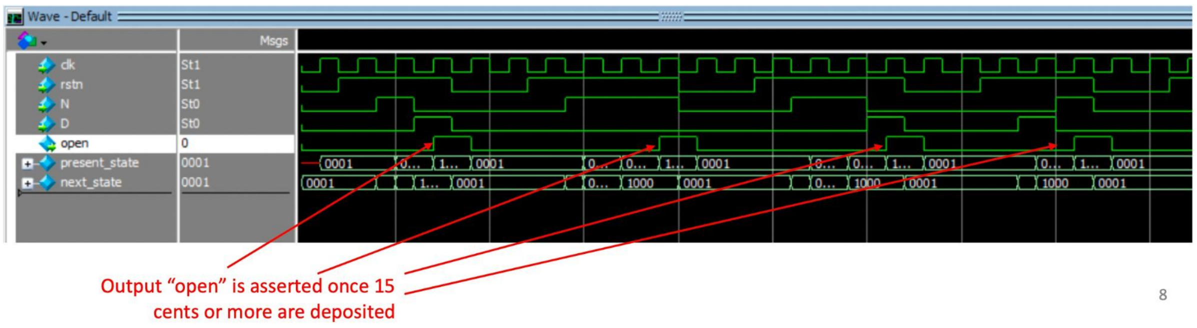
7

- **Assume below mentioned primary Ports for Vending Machine**
  - Input clk : posedge clock
  - Input rstn : reset should be synchronous negedge reset.
  - Input N, D : 1-bit Nickel and Dime inputs indicating Nickel and Dime are deposited if values are '1' .
  - Output open : 1-bit open signal indicating vending machine is dispatching candy.

- **Block Diagram**:

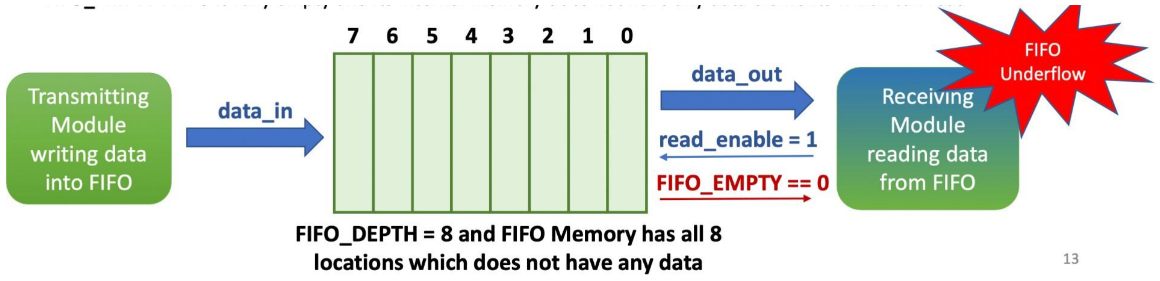


- **Reference Moore FSM Simulation Waveform:**

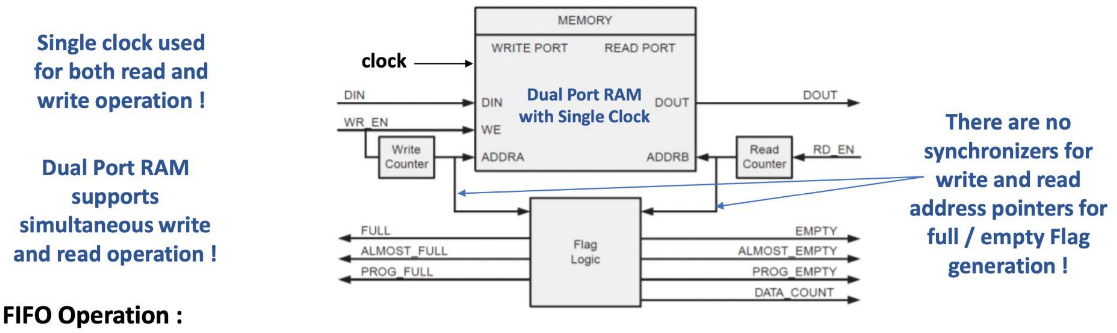


<u>For Homework-6b:</u>

- **FIFO Underflow:**
    - When FIFO internal memory is fully empty and any attempt to read any location of FIFO memory is made, it is called FIFO underflow condition.
    - Underflow expression is : (FIFO_EMPTY == 0) && (read_enable == 1). See the below mentioned Figure.
    - It is the responsibility of the receiving module to ensure it does not read FIFO memory when it is empty.
    - Typically FIFO designs provide a FIFO_EMPTY signal which gives an indication to the receiving module if FIFO has any data element to read or not.
    - Additionally FIFO designs provides such more flags to indicate memory occupancy information
        - **FIFO_ALMOST_EMPTY** : FIFO internal memory has 1 data. element remaining to be read before it is fully empty. Acts like an early empty indication.
        - **FIFO_HALF_EMPTY** : FIFO internal memory has 50% occupancy
        - **FIFO_EMPTY** : FIFO is fully empty and its internal memory does not have any data elements which can be read.


7 6 5 4 3 2 1 0
Transmitting Module writing data into FIFO
data_in
data_out
read_enable = 1
FIFO_EMPTY == 0
Receiving Module reading data from FIFO
FIFO Underflow
FIFO_DEPTH = 8 and FIFO Memory has all 8 locations which does not have any data
13

Synchronous FIFO


Single clock used for both read and write operation !
Dual Port RAM supports simultaneous write and read operation !
MEMORY
WRITE PORT
READ PORT
clock
DIN
WR_EN
Write Counter
DIN
WE
ADDRA
Dual Port RAM with Single Clock
DOUT
ADDRB
DOUT
Read Counter
RD_EN
There are no synchronizers for write and read address pointers for full / empty Flag generation !
FULL
ALMOST_FULL
PROG_FULL
Flag Logic
EMPTY
ALMOST_EMPTY
PROG_EMPTY
DATA_COUNT

FIFO Operation :

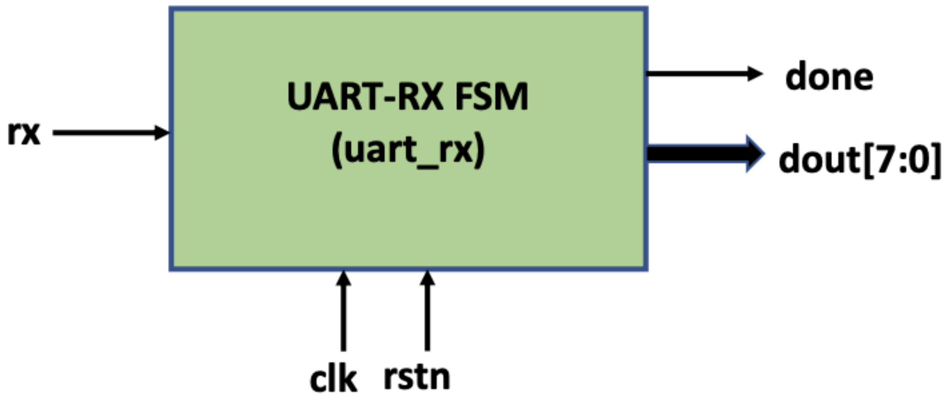
- Within FIFO there is a has dual port memory block for storage. **It allows simultaneous write and read operation !**
- FIFO has an input data in port (DIN) and an output read port (DOUT).
- Each data port has its own associated pointers which points to a location in the memory
- After a FIFO reset both the write and read pointers will be at the first memory location within the FIFO.
- Each write operation will cause the write pointer to increment to the next location in memory
- Each read operation will cause the read pointer to increment to the next location
- FIFO has full and empty signals indicating whether it has no empty location for any new data to be stored (full condition) or it has no data available for reading (empty condition)
- Full marker prevents overriding of existing data. This is known as fifo overrun or overflow condition
- Empty marker prevents reading junk data, when it is empty. This is known as fifo underrun or underflow condition

- **Review and Study SystemVerilog RTL model for M-bit width and N-depth Synchronous FIFO:**
  - Review System Verilog code for Synchronous FIFO provided in Lab folder.
  - Review SystemVerilog code for dual port ram in Lab folder.
  - Synthesize sync_fifo model and run simulation using testbench provided.
  - Review synthesis results (resource usage and RTL netlist/schematic).
  - Review input and output signals in simulation waveform.
    - Review dual_port_ram input and output signals in waveform to understand how write and read operations are performed
    - Review sync_fifo input and output signals in waveform and understand write and read operations are performed

<u>For Homework-6c</u>

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 bps) = 8.68us, NUM_CLKS_PER_BIT = 8.68us/ 20ns = 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.



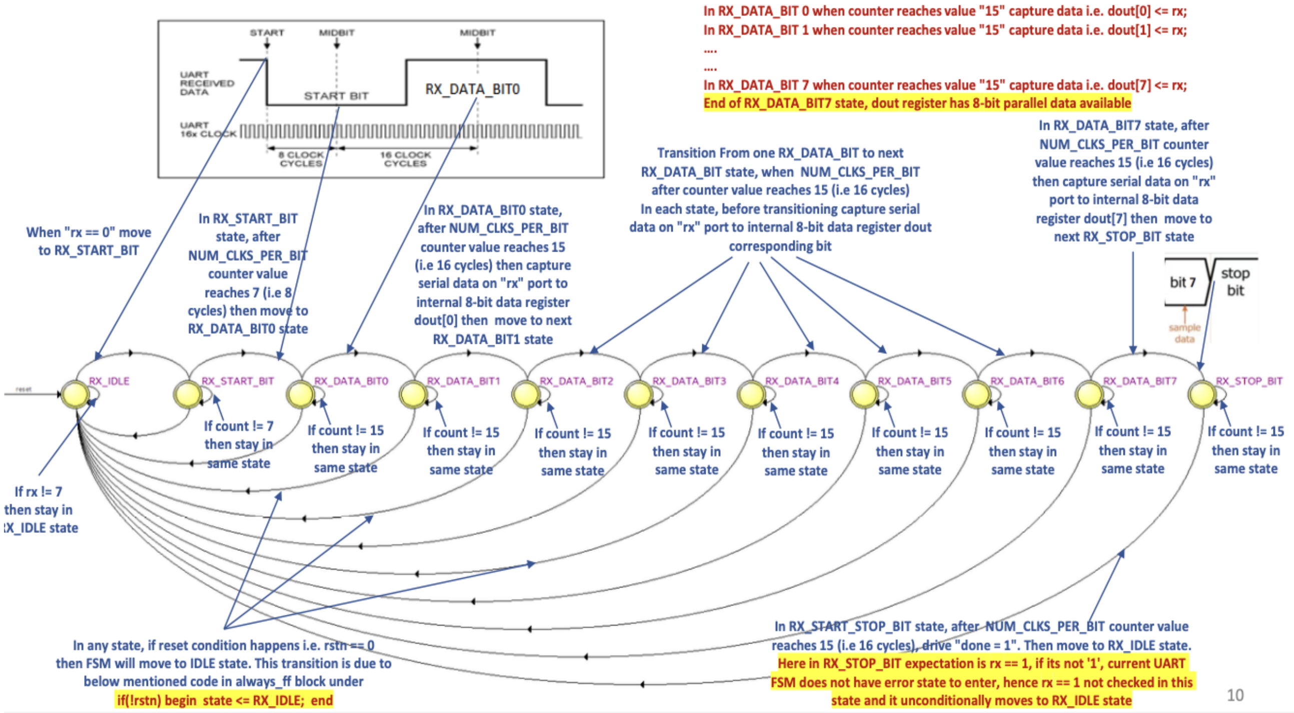
How may states are required to design UART-RX FSM ?

- 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 sate = 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

❑ 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

When "rx == 0" move to RX_START_BIT

In RX_START_BIT state, after NUM_CLKS_PER_BIT counter value reaches 7 (i.e 8 cycles) then move to RX_DATA_BIT0 state

In RX_DATA_BIT0 state, after NUM_CLKS_PER_BIT counter value reaches 15 (i.e 16 cycles) then capture serial data on "rx" port to internal 8-bit data register dout[0] then move to next RX_DATA_BIT1 state

Transition From one RX_DATA_BIT to next RX_DATA_BIT state, when NUM_CLKS_PER_BIT after counter value reaches 15 (i.e 16 cycles)
In each state, before transitioning capture serial data on "rx" port to internal 8-bit data register dout corresponding bit

In RX_DATA_BIT7 state, after NUM_CLKS_PER_BIT counter value reaches 15 (i.e 16 cycles) then capture serial data on "rx" port to internal 8-bit data register dout[7] then move to next RX_STOP_BIT state

If rx != 7 then stay in RX_IDLE state

If count != 7 then stay in same state

If count != 15 then stay in same state

If count != 15 then stay in same state

If count != 15 then stay in same state

If count != 15 then stay in same state

If count != 15 then stay in same state

If count != 15 then stay in same state

If count != 15 then stay in same state

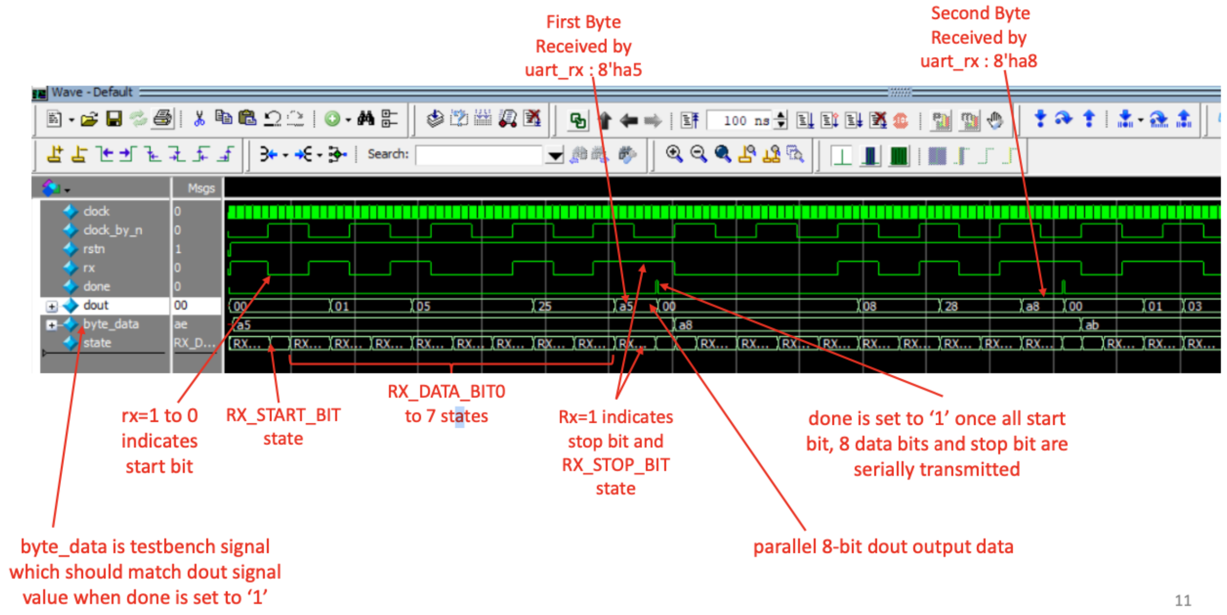
If count != 15 then stay in same state

If count != 15 then stay in same state

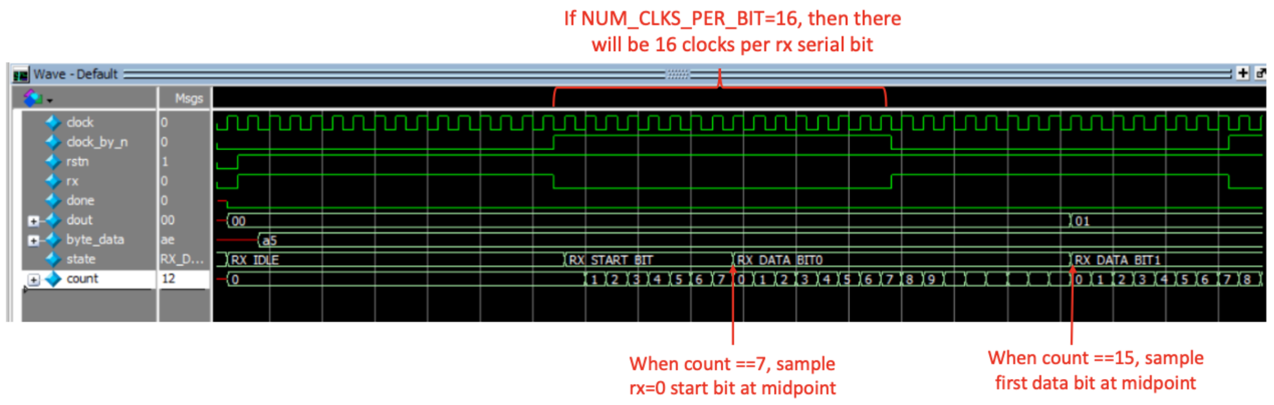
In any state, if reset condition happens i.e. rstn == 0 then FSM will move to IDLE state. This transition is due to below mentioned code in always_ff block under
if(!rstn) begin state <= RX_IDLE; end

In RX_START_STOP_BIT state, after NUM_CLKS_PER_BIT counter value reaches 15 (i.e 16 cycles), drive "done = 1". Then move to RX_IDLE state.
Here in RX_STOP_BIT expectation is rx == 1, if its not '1', current UART FSM does not have error state to enter, hence rx == 1 not checked in this state and it unconditionally moves to RX_IDLE state

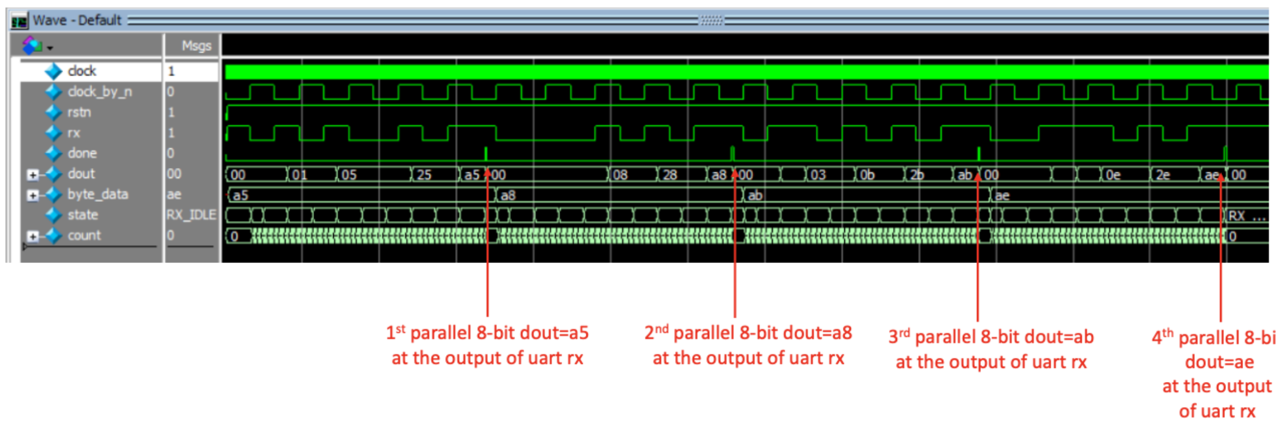
UART Receiver (UART-RX) Simulation Waveform View 1


First Byte Received by uart_rx : 8'ha5
Second Byte Received by uart_rx : 8'ha8
Wave - Default
Msgs
clock
clock_by_n
rstn
rx
done
dout
byte_data
state
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

UART Receiver (UART-RX) Simulation Waveform View 2

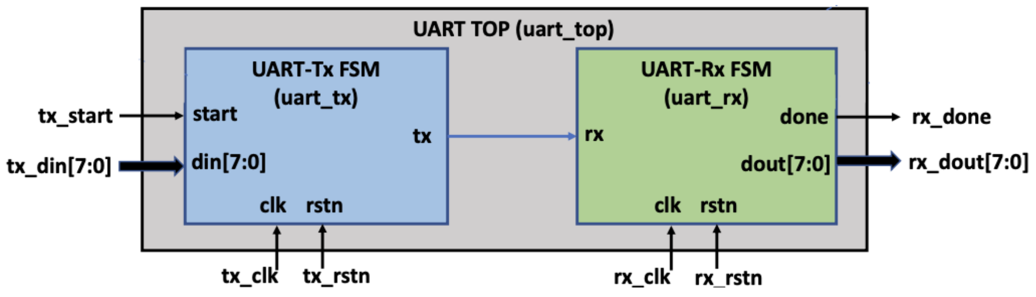

If NUM_CLKS_PER_BIT=16, then there will be 16 clocks per rx serial bit
Wave - Default
Msgs
clock
clock_by_n
rstn
rx
done
dout
byte_data
state
count
RX IDLE
RX START BIT
RX DATA BIT0
RX DATA BIT1
When count ==7, sample rx=0 start bit at midpoint
When count ==15, sample first data bit at midpoint

UART Receiver (UART-RX) Simulation Waveform View 3

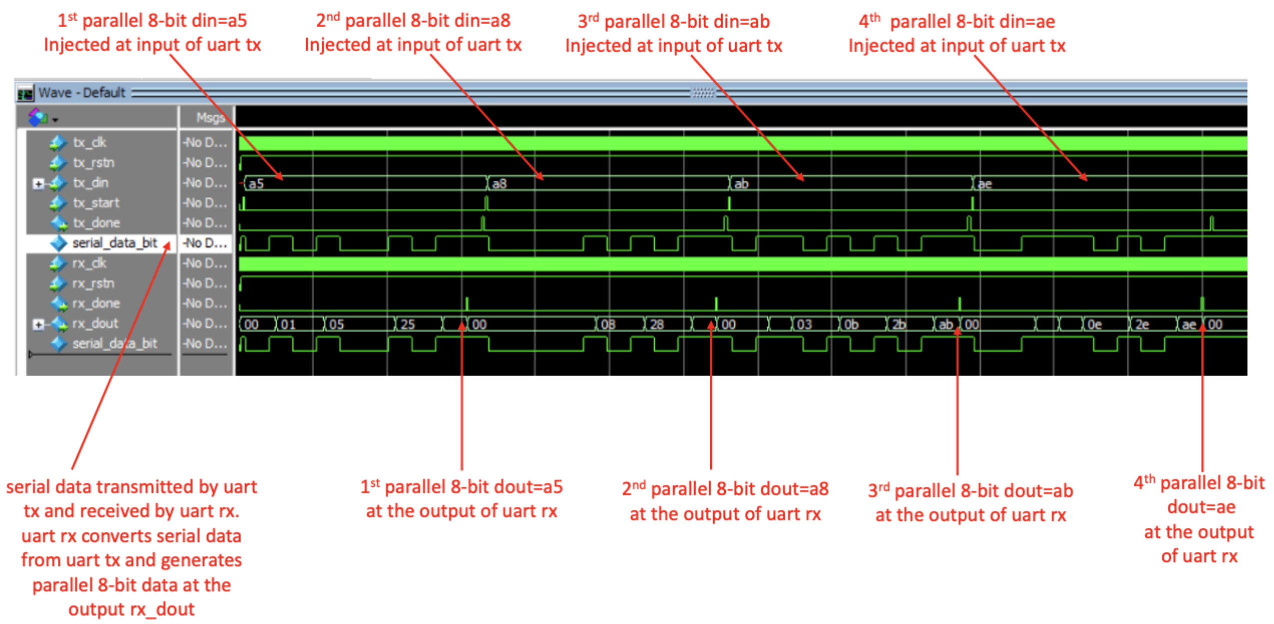

Wave - Default
Msgs
clock
clock_by_n
rstn
rx
done
dout
byte_data
state
count
1st parallel 8-bit dout=a5 at the output of uart rx
2nd parallel 8-bit dout=a8 at the output of uart rx
3rd parallel 8-bit dout=ab at the output of uart rx
4th parallel 8-bit dout=ae at the output of uart rx

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 folderSynthesis
- uart_top module along with uart_tx and uart_rx module
- 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

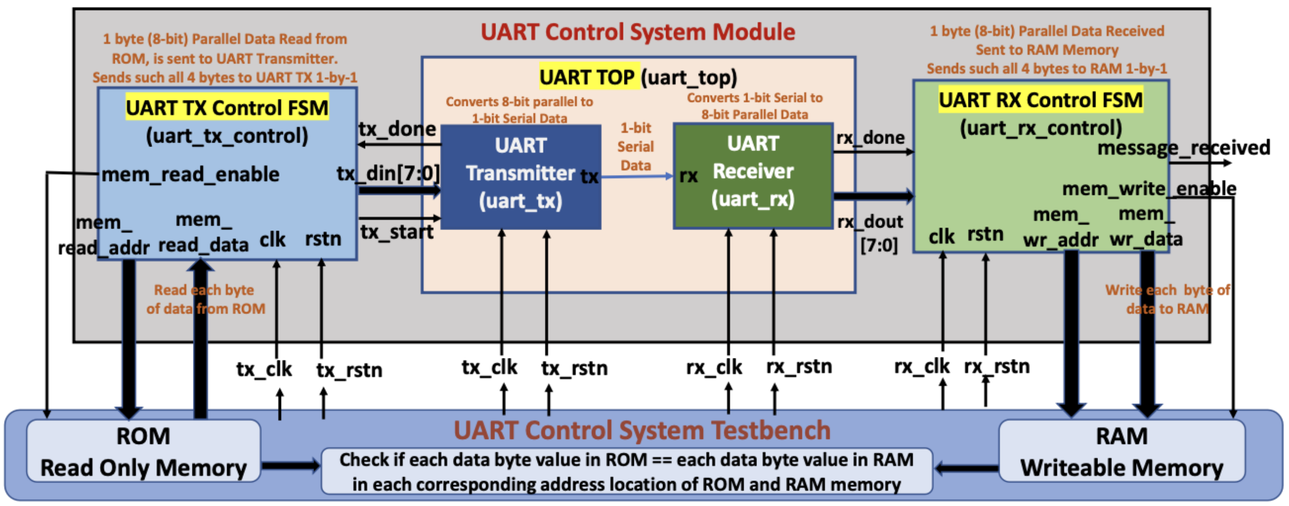


❑ UART Top (i.e. UART TX-RX) Simulation Waveform

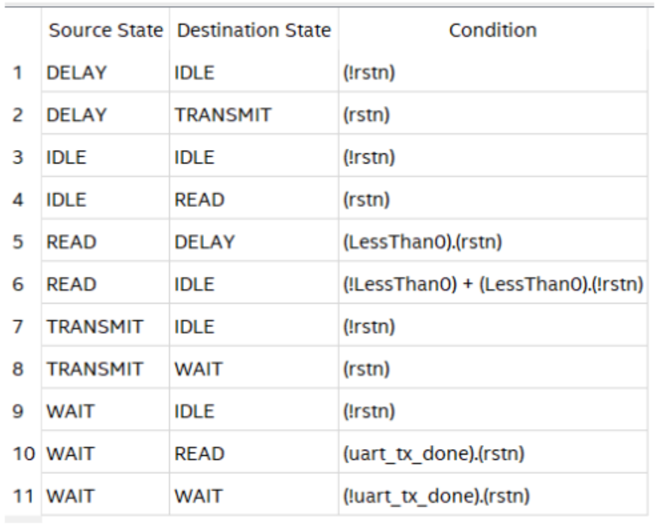

1st parallel 8-bit din=a5 Injected at input of uart tx
2nd parallel 8-bit din=a8 Injected at input of uart tx
3rd parallel 8-bit din=ab Injected at input of uart tx
4th parallel 8-bit din=ae Injected at input of uart tx
Wave - Default
Msgs
tx_clk
tx_rstn
tx_din
tx_start
tx_done
serial_data_bit
rx_clk
rx_rstn
rx_done
rx_dout
serial_data_bit
-No D...
a5
a8
ab
ae
00 01 05 25 00 08 28 00 03 0b 2b ab 00 0e 2e ae 00
serial data transmitted by uart tx and received by uart rx. uart rx converts serial data from uart tx and generates parallel 8-bit data at the output rx_dout
1st parallel 8-bit dout=a5 at the output of uart rx
2nd parallel 8-bit dout=a8 at the output of uart rx
3rd parallel 8-bit dout=ab at the output of uart rx
4th parallel 8-bit dout=ae at the output of uart rx

Design UART Control System Module:

- 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

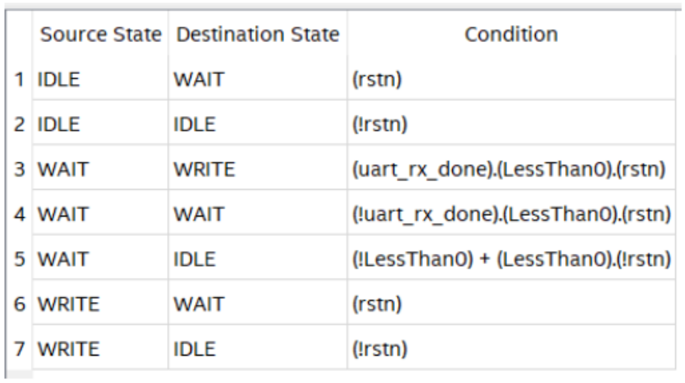

UART Control System Module
1 byte (8-bit) Parallel Data Read from ROM, is sent to UART Transmitter. Sends such all 4 bytes to UART TX 1-by-1
UART TX Control FSM
(uart_tx_control)
mem_read_enable
mem_read_addr
mem_read_data
clk
rstn
tx_done
tx_din[7:0]
tx_start
UART TOP (uart_top)
Converts 8-bit parallel to 1-bit Serial Data
UART Transmitter (uart_tx)
tx
1-bit Serial Data
Converts 1-bit Serial to 8-bit Parallel Data
rx
UART Receiver (uart_rx)
rx_done
rx_dout [7:0]
1 byte (8-bit) Parallel Data Received Sent to RAM Memory Sends such all 4 bytes to RAM 1-by-1
UART RX Control FSM
(uart_rx_control)
message_received
mem_write_enable
mem_wr_addr
mem_wr_data
clk
rstn
Read each byte of data from ROM
Write each byte of data to RAM
tx_clk
tx_rstn
rx_clk
rx_rstn
UART Control System Testbench
ROM Read Only Memory
Check if each data byte value in ROM == each data byte value in RAM in each corresponding address location of ROM and RAM memory
RAM Writeable Memory

UART TX Control FSM State Transition Diagram:



	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)

UART RX Conrol FSM State Transition Diagram



	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)

Requirements for UART Control System:

- 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

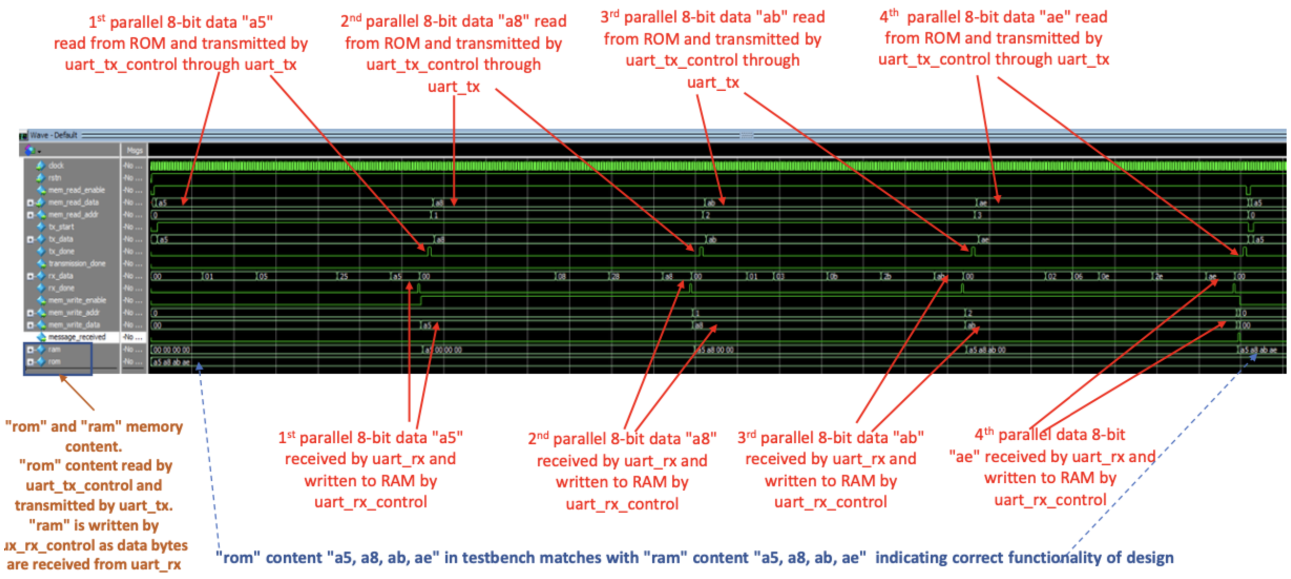
RAM in testbench
    - **output** logic mem_write_enable : memory write enable generated to enable writing to RAM in testbench
    - **input** logic[7:0] mem_read_data : memory read data returned from ROM in testbench
    - **output** logic[3:0] mem_read_addr : memory read address generated to read ROM 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

- 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
- transmissted 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

UART Control System Simulation snapshot for reference purpose


1st parallel 8-bit data "a5" read from ROM and transmitted by uart_tx_control through uart_tx
2nd parallel 8-bit data "a8" read from ROM and transmitted by uart_tx_control through uart_tx
3rd parallel 8-bit data "ab" read from ROM and transmitted by uart_tx_control through uart_tx
4th parallel 8-bit data "ae" read from ROM and transmitted by uart_tx_control through uart_tx
clock
rstn
mem_read_enable
mem_read_data
mem_read_addr
tx_start
tx_data
tx_done
transmission_done
rx_data
rx_done
mem_write_enable
mem_write_addr
mem_write_data
message_received
ram
rom
"rom" and "ram" memory content.
"rom" content read by uart_tx_control and transmitted by uart_tx.
"ram" is written by ux_rx_control as data bytes are received from uart_rx
1st parallel 8-bit data "a5" received by uart_rx and written to RAM by uart_rx_control
2nd parallel 8-bit data "a8" received by uart_rx and written to RAM by uart_rx_control
3rd parallel 8-bit data "ab" received by uart_rx and written to RAM by uart_rx_control
4th parallel data 8-bit "ae" received by uart_rx and written to RAM by uart_rx_control
"rom" content "a5, a8, ab, ae" in testbench matches with "ram" content "a5, a8, ab, ae" indicating correct functionality of design

Submission Requirements

Submit a report on Gradescope in PDF format which includes the following :

<u>For homework-6a:</u>

- SystemVerilog design code for both Moore and Mealy FSM.
- Synthesis resource usage and schematic generated from RTL netlist viewer for both Moore and Mealy FSM.
- Simulation waveform snapshots and explain the simulation results for both Moore and Mealy FSM
- FSM state transition diagram for both Moore and Mealy implementations.
- Explanation of FPGA resource usage in the report is not required
- State transition diagram needs to be submitted in report and it can be either hand drawn with picture taken and pasted in report or it could be drawn in word or powerpoint or auto-generated state machine diagram from Quartus prime is also acceptable. If you're submitting an auto-generated Quartus state machine diagram then also attach a state transition table generated from Quartus prime.
- For Moore FSM, diagram from the discussion slides can also be used.
- Simulation transcript is **not** required in report since there are no prints from testbench

<u>For homework-6b:</u>

- Synthesis resource usage snapshot and RTL netlist viewer schematic generated from Quartus for sync_fifo top level module.
- Simulation snapshot (including the transcript) and explain simulation result of sync_fifo :
  - Describe how data is sent to sync fifo and read from sync_fifo, Explain full and empty flags are generated, explain how sync fifo works.
  - Explain role of dual_port_ram in sync_fifo design and how read and write operation is performed to dual_port_ram.
- Code snippet not required.
- Explanation of FPGA resource usage in the report is not required.

<u>For Homework-6c:</u>

- SystemVerilog code snapshot for UART Receiver, UART Top, and UART Control System.
- Synthesis resource usage for all three parts.
- Simulation snapshot and a brief explanation for simulation snapshot.
- Modelsim transcripts with test passed message for all three parts of UART.
- FSM State Transition diagram and state transition table, either generated from Quartus Prime or hand-drawn.