Finite State Machine & Memories

<u>Design the following circuits using Verilog **and create a testbench** for each design to check its functionality using QuestaSim. Use a **do file** for question 2.</u>

Use **Vivado** to go through the **design flow** running elaboration, synthesis, implementation for all the designs making sure that there are no design check errors during the design flow. <u>Check the deliverables by the end of the document.</u>

1) Suppose that you are working as a Digital design Engineer in Tesla Company. It is required to design a control unit using Moore FSM for Self-driving cars on highways that controls the acceleration of the car as well as the door locking mechanism with the following specifications. FSM encoding is as follows:

- STOP -> 2'b00, ACCELERATE -> 2'b01, DECELERATE -> 2'b10.

<u>Requirement</u>: run different FSM encoding and create a table comparing the utilization and worst slack for each encoding used. Try using gray, seq, and one_hot encoding. Comment on the results. Take snippets from each schematic from each encoding.

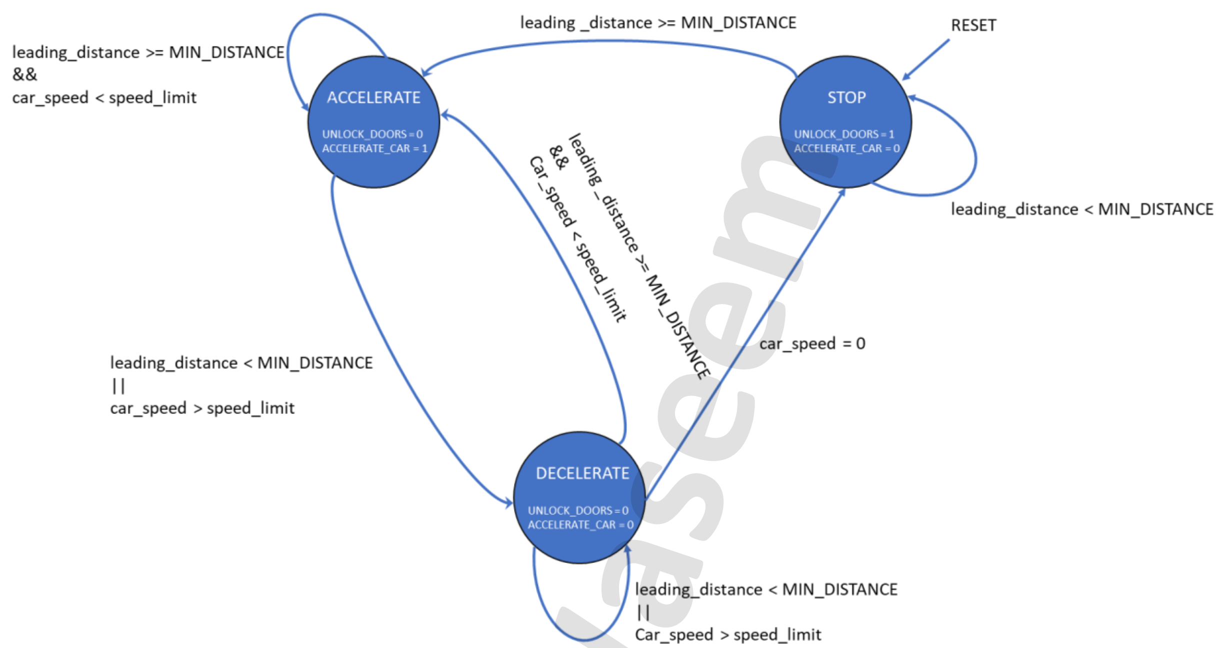
<u>>> Consider creating realistic testbench for this design <<</u>

Ports

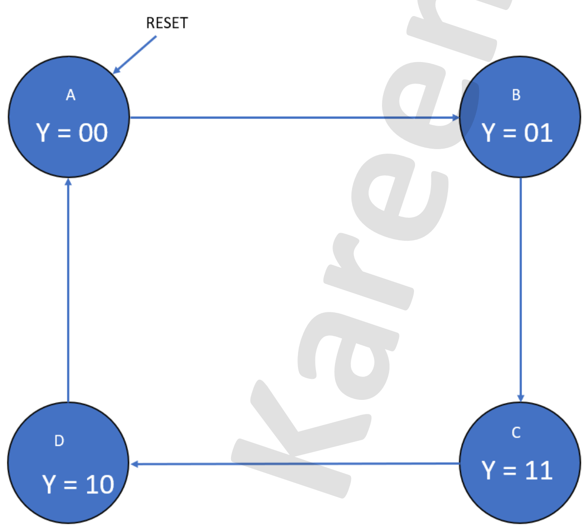
Name	Type	Size	Description
speed_limit	Input	8 bits	Allowable speed limit of the highway
car_speed		8 bits	Current car speed
leading_distance		7 bits	Distance between the car and the vehicle/object in front of it
clk		1 bit	Clock
rst		1 bit	Active high asynchronous reset
unlock_doors	Output	1 bit	Signal that unlock the car doors when HIGH
accelerate_car			Signal that control the flow of the fuel to the engine to accelerate the car when HIGH

Parameters

- MIN_DISTANCE: Minimum distance between two vehicles, default = 7'd40 //40 meters



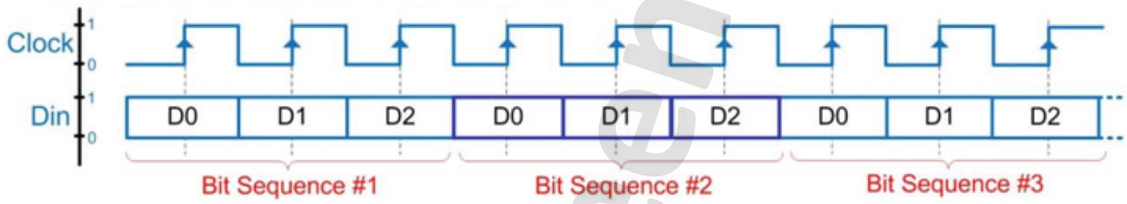
2) Implement a 2-bit gray counter using Moore FSM. Test the following by instantiating the gray counter done in assignment 4 (extra) and taking it as a reference design. When the reset is deasserted, the states will continue to transition from each state to the neighboring state with each clock cycle.



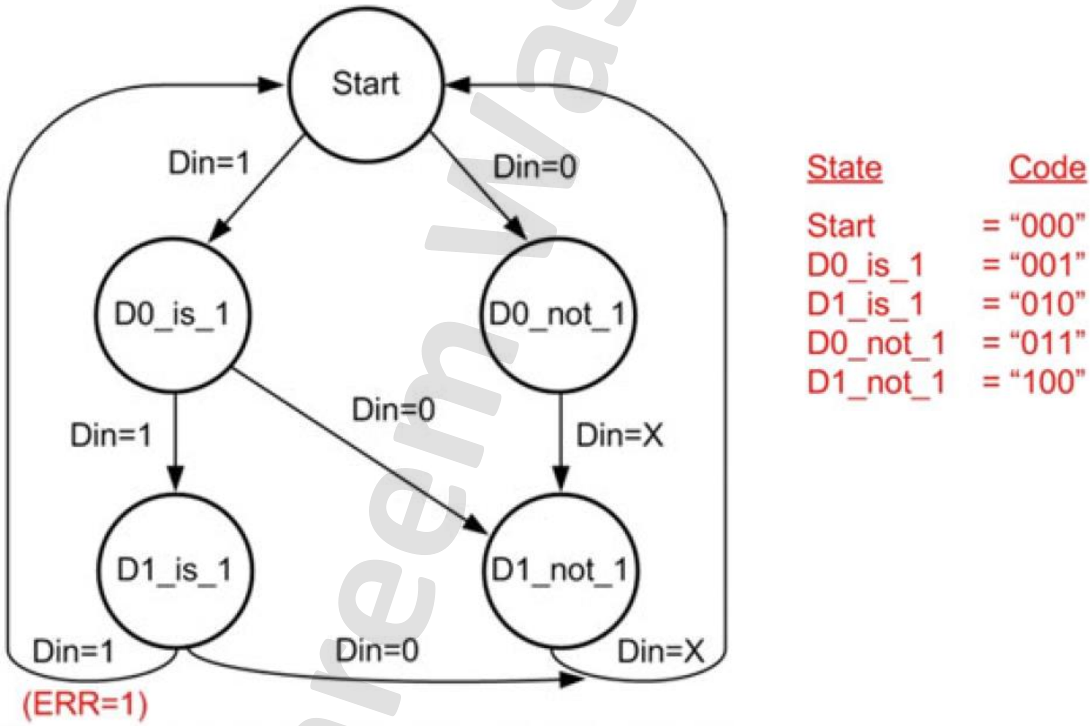
FSM Encoding:
- A = 2’b00
- B = 2’b01
- C = 2’b10
- D = 2’b11

- Inputs: clk, rst (active high async)
- Outputs: y, 2-bit output
- Create a constraint file and go through the design flow where the clk is connected to W5 pin, reset is connected to a button and the y output is connected to two LEDs and then generate a bitstream file.

3) We are required to design a circuit to monitor an incoming serial bit stream in a communication system between a transmitter and a receiver. The data is transmitted in groups of three bits, and the sequence "111" indicates an error in the transmission. When the receiver detects this sequence, it asserts an output signal called "ERR" to notify the transmitter that an error has occurred. Upon receiving this signal, the transmitter must resend the data. For all other sequences, ERR should remain low (0).



The following is the Mealy FSM transition diagram for the 3-bit serial bit sequence detector. Use binary state encoding (000, 001, 010, etc.).



- Inputs: Din, clk, rst (active high async)
- Outputs: ERR, 1-bit output
- Create a testbench randoming data D_in and observing the ERR on the waveform.
- Create a constraint file and go through the design flow where the clk is connected to W5 pin, reset is connected to a button, Din to a switch and the ERR output is connected to a LED and then generate a bitstream file.

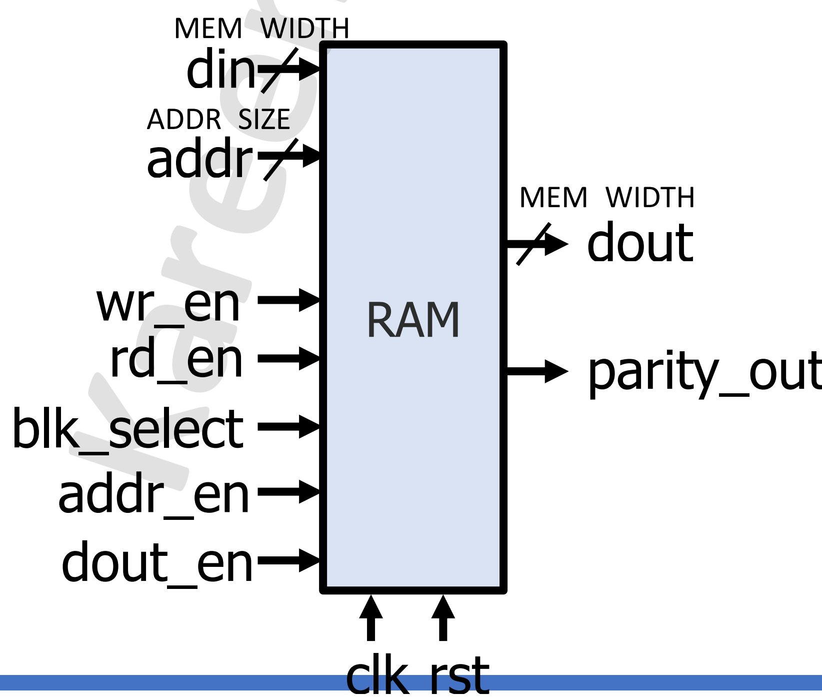
- Run the synthesis flow using seq, gray and one_hot encoding, report the number of LUTs, registers used for each encoding as well as the worst slack for the hold and setup times.
    - Comment on which encoding can be used in case you want to run the design on the highest frequency possible or in case you want the minimum area possible.

4) Implement the following single port synchronous write/read and create a testbench for it

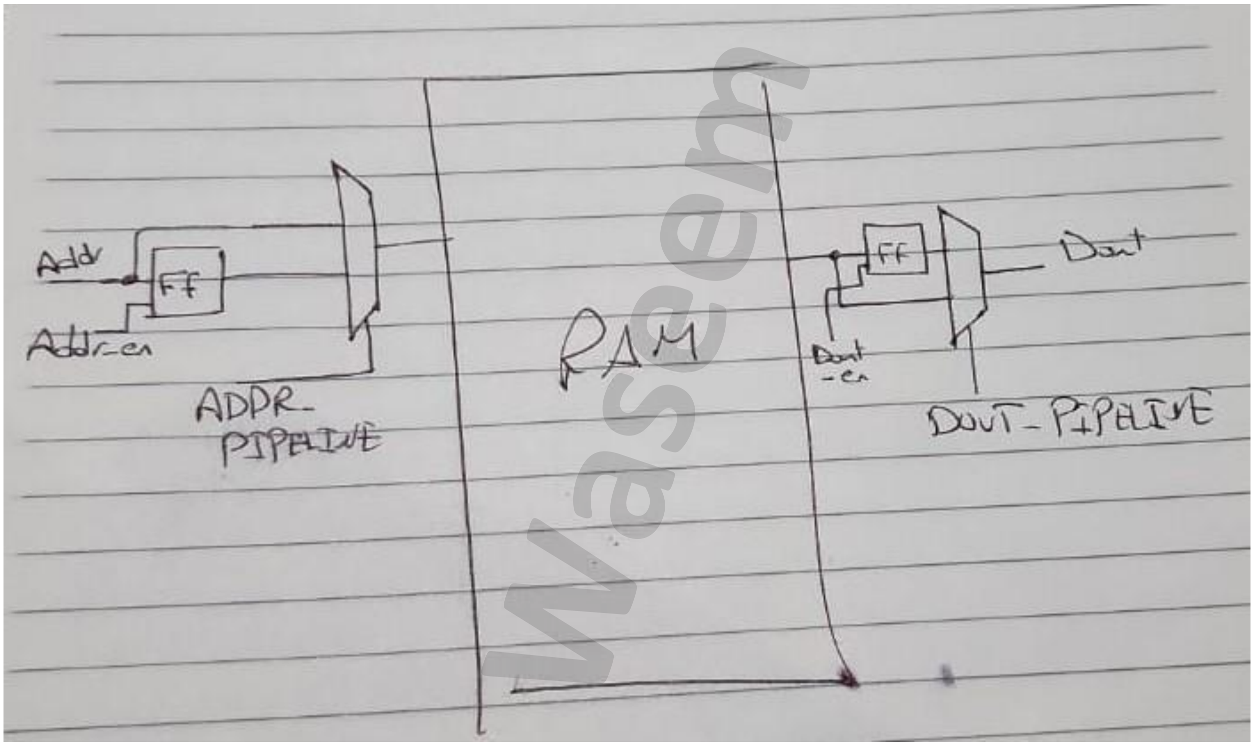
- Parameters

Name	Description	Default values
MEM_WIDTH	Data in/out and memory word width	16
MEM_DEPTH	Memory depth	1024
ADDR_SIZE	Address size based upon the memory depth	10
ADDR_PIPELINE	If “TRUE” then the address should be pipelined before writing/reading the RAM, if ”FALSE” then the address input will be assigned directly to the RAM’s address port	FALSE
DOUT_PIPELINE	If “TRUE” then the data out should be pipelined, if ”FALSE” then the output will be out of the RAM directly	TRUE
PARITY_ENABLE	If the parameter value is 1 then the parity should be calculated and assigned to parity_out port, if the parameter is 0 then the parity_out port should be tied to 0	1

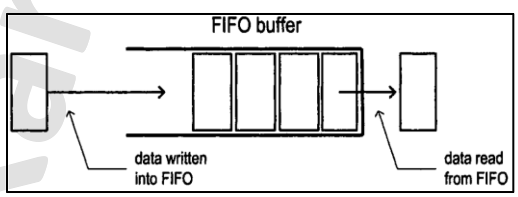
- Added ports functionality
    - addr_en: enable signal for the flipflop that pipelines the address
    - dout_en: enable signal for the flipflop that pipelines the data out
    - parity_out: calculates the even parity on the dout bus
    - rst: active high synchronous reset



The internal pipelining done inside of the design is demonstrated by the following snippet



5) Implementing a FIFO (First-In-First-Out) memory. FIFO is memory structure that stores and retrieves data elements in the order they were added. The FIFO memory will be designed to have two main operations: writing (enqueuing) data and reading (dequeuing) data. We'll use two internal pointers (counters) to keep track of the write and read positions within the memory. The write pointer advances when new data is written, and the read pointer advances when data is read.



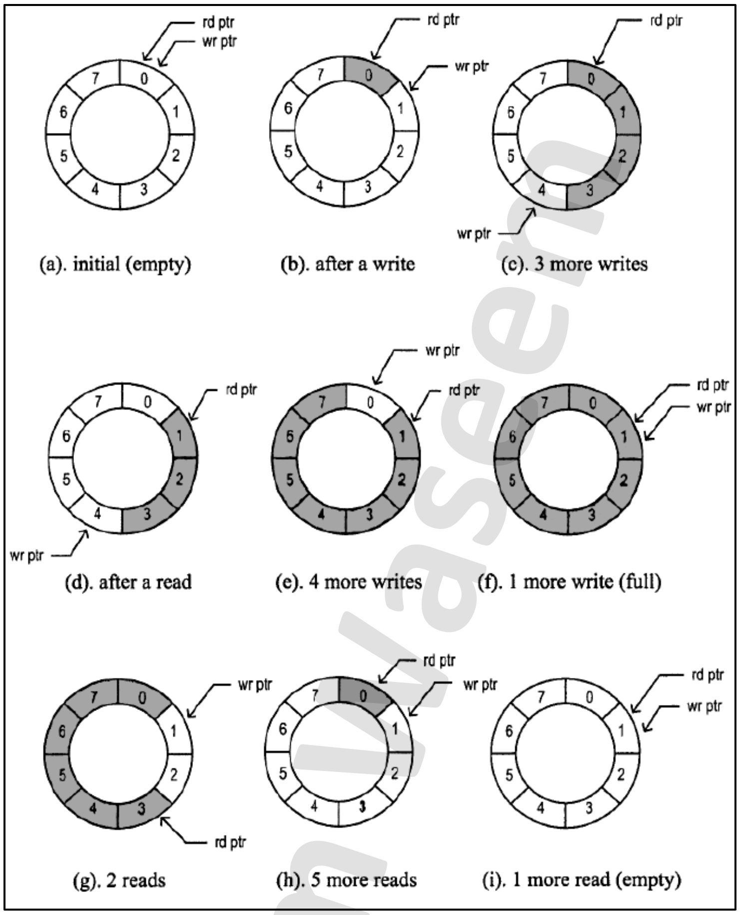


Figure 1 FIFO With depth of 8 words

Parameters

- FIFO_WIDTH: DATA in/out and memory word width (default: 16)
- FIFO_DEPTH: Memory depth (default: 512)

Ports

Port	Width	Direction	Function
din_a	FIFO_WIDTH	Input	Write Data: The input data bus used when writing the FIFO.
wen_a	1		Write Enable: If the FIFO is not full, asserting this signal causes data (on din_a) to be written into the FIFO
ren_b	1		Read Enable: If the FIFO is not empty, asserting this signal causes data (on dout_b) to be read from the FIFO
clk_a	1		Clock signal for port a, used in the writing operation

clk_b	1		Clock signal for port b, used in the reading operation
rst	1		Active high synchronous reset. It resets the dout_b, internal write counter & internal read counters
dout_b	FIFO_WIDTH	Output	Read Data: The output data bus used when reading from the FIFO.
full	1		Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is not destructive to the contents of the FIFO.
empty	1		Empty Flag: When asserted, this signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is not destructive to the FIFO.

6) You are required to modify the ALSU done in assignment 4 by using the IP catalog to generate following 2 IPs:

- multilper to be used when the opcode is 3
- adder to be used when the opcode is 2.
    - Adder IP takes in the A and B and the carry in as inputs. Use a generate if block to instantiate the IP connecting its ports to the ALSU A, B and cin ports when the FULL_ADDER parameter is ON, else connect only A and B and connect the carry in port of the Adder IP to zero.

Bonus (Extra):

The output “out” shall be displayed on the seven segment display

Adjustments to the outputs:

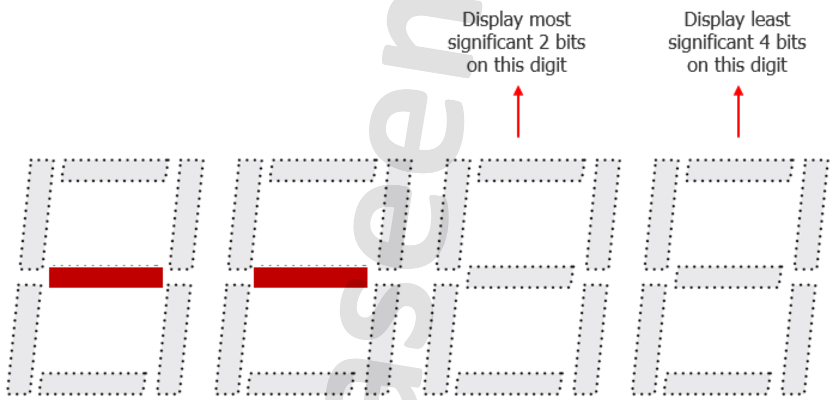
- “out” port will not be output anymore and it will be an internal register
- The output ports are shown below where two outputs are added to be connected to the seven segment display
- You can learn more on how to drive the seven segment display in Basys reference manual and in this [link]

Output	Width	Description
leds	16	All bits are blinking when an invalid operation takes place. Acts as a warning
anode	4	Output to drive the anodes of the seven segment display
cathode	7	Output to drive the cathodes of the seven segment display

Seven segment display:

1. The least significant 4 bits of the internal register “out” which is the ALSU output will be displayed on the right most digit

2. The most significant 2 bits of the internal register “out” will be displayed on the second digit on the right
3. The 2 digits on the left will be displaying a dash in the middle of them as they are not used
4. If invalid case occurs then “E404” will be displayed on the 4 digits



Deliverables:

1) The assignment should be submitted as a PDF file with this format <your_name>_Assignment5 for example Kareem_Waseem_Assignment5.
2) RTL and Testbench code snippets
3) Snippets from the waveforms captured from QuestaSim for the design with inputs assigned values and output values visible.
4) Snippets from the RTL schematic after the elaboration
5) Snippets from the technology schematic after synthesis
6) Snippet from the utilization & timing report after the synthesis and implementation.
7) Snippet of the “Messages” tab showing no critical warnings or errors after running elaboration, synthesis, and implementation.
8) Linting snippet from the linting tool showing no warnings or errors (use the default methodology and goals)

Note that your document should be organized as follows:

1. Verilog Code
    a. RTL Code
    b. Testbench code
2. Simulation Tool
    a. Do file
    b. QuestaSim Snippets
3. Synthesis Tool
    a. Constraint File
    b. Elaboration (“Messages” tab & Schematic snippets)
    c. Synthesis (“Messages” tab, Utilization report, timing report & Schematic snippets). Memories should be inferred in the synthesis schematic in the memories questions.

   d. Generate the gate level netlist and take a snippet of the generated netlist file.
   e. Implementation ("Messages" tab, Utilization report, timing report & device snippets)
   f. Snippet for successful bitstream generation for last question only (bonus part)
4. Linting Tool
   a. Run lint on the RTL and take a snippet showing no warnings or errors

<u>Deliverables for questions 6</u>: Same as the above but you do not have to run Questasim again for simulations, only add snippets of the Verilog code after adding the IP catalog

Note that your document should be organized as 6 sections corresponding to each design.