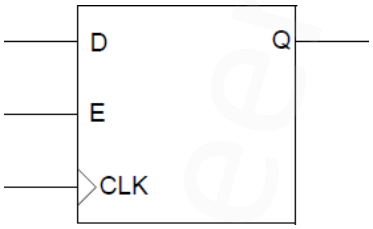
Sequential Logic Design

- Design the following circuits using Verilog **and create a testbench** for each design to check its functionality. **Create a do file for question 5 only.**

- Testbenches are advised to be a mix between randomization and directed testing taken into consideration realistic operation for the inputs. Apply self-checking condition in the testbench for at least one of the design below.

1) Implement D-Type Flip-Flop with active high Enable.

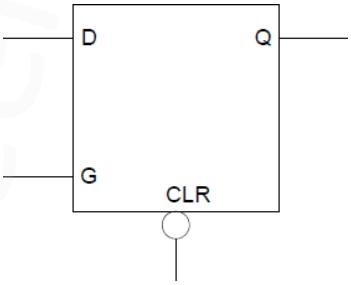


Input	Output
D, E, CLK	Q

Truth Table

E	CLK	D	Q_{n+1}
0	X	X	Q_n
1	not Rising	X	Q_n
1	↑	D	D

2) Implement Data Latch with active low Clear



Input	Output
CLR, D, G	Q

Truth Table

CLR	G	D	Q
0	X	X	0
1	0	X	Q
1	1	D	D

3) Implement the following latch as specified below

Parameters

LAT_WIDTH: Determine the width of input data and output q

Ports

Name	Type	Description
aset	Input	Asynchronous set input. Sets q[] output to 1.
data[]		Data Input to the D-type latch with width LAT_WIDTH
gate		Latch enable input
aclr		Asynchronous clear input. Sets q[] output to 0.
q[]	Output	Data output from the latch with with LAT_WIDTH

If both aset and aclr are both asserted, aclr is dominant.

4)

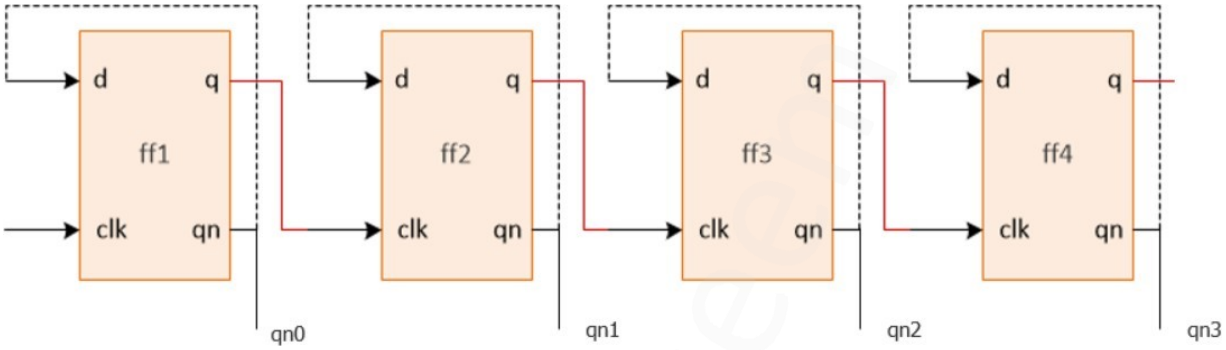
A. Implement T-type (toggle) Flipflop with active low asynchronous reset. T-Flipflop has input t, when t input is high the outputs toggle else the output values do not change.
   - Inputs: t, rstn, clk
   - Outputs: q, qbar
B. Implement Asynchronous D Flip-Flop with Active low reset
   - Inputs: d, rstn, clk
   - Outputs: q, qbar
C. Implement a parameterized asynchronous FlipFlop with Active low reset with the following specifications.
   - Inputs: d, rstn , clk
   - Outputs: q, qbar
   - Parameter: FF_TYPE that can take two valid values, DFF or TFF. Default value = “DFF” Design should act as DFF if FF_TYPE = “DFF” and act as TFF if FF_TYPE = “TFF”. When FF_TYPE equals “DFF”, d input acts as the data input “d”, and when FF_TYPE equals “TFF”, d input acts the toggle input “t”.

D. Test the above parameterized Design using 2 testbenches, testbench 1 that overrides the design with FF_TYPE = “DFF” and the testbench 2 overrides parameter with FF_TYPE = “TFF”

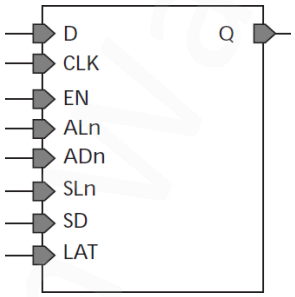
- Testbench 1 should instantiate the design of part B. as a golden model to check for the output of the parameterized design with FF_TYPE = “DFF”
- Testbench 2 should instantiate the design of part A. as a golden model to check for the output of the parameterized design with FF_TYPE = “TFF”

5) Implement the 4-bit Ripple counter shown below using structural modelling (Instantiate the Dff from question 4 part B where the output is taken from the qn as shown below)

- Inputs: clk, rstn;
- Outputs: [3:0] out;



6) Implement the following SLE (sequential logic element)



Input		Output
Name	**Function**	
D	Data	
CLK	Clock	
EN	Enable	
ALn	Asynchronous Load (Active Low)	Q
ADn*	Asynchronous Data (Active Low)	
SLn	Synchronous Load (Active Low)	
SD*	Synchronous Data	
LAT*	Latch Enable	

***Note**: ADn, SD and LAT are static signals defined at design time and need to be tied to 0 or 1.

Truth Table

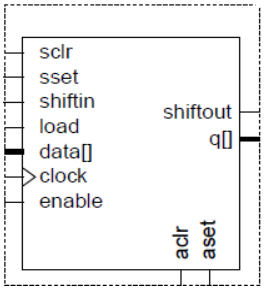
ALn	ADn	LAT	CLK	EN	SLn	SD	D	Q_{n+1}
0	ADn	X	X	X	X	X	X	!ADn
1	X	0	Not rising	X	X	X	X	Qn
1	X	0	↑	0	X	X	X	Qn
1	X	0	↑	1	0	SD	X	SD
1	X	0	↑	1	1	X	D	D
1	X	1	0	X	X	X	X	Qn
1	X	1	1	0	X	X	X	Qn
1	X	1	1	1	0	SD	X	SD
1	X	1	1	1	1	X	D	D

Assignment 4
Counters & Shift Registers

Design the following circuits using Verilog **and create a testbench** for each design to check its functionality. **Create a do file for question 2 only.**

- Testbenches are advised to be a mix between randomization and directed tes
consideration realistic operation for the inputs.

1) Implement the following Parameterized Shift register



- Parameters

Name	Value	Description
LOAD_AVALUE	Integer > 0	Value loaded with aset is high
SHIFT_DIRECTION	"LEFT" or "RIGHT"	Direction of the shift register. Default = "LEFT"
LOAD_SVALUE	Integer > 0	Value loaded with sset is high with the rising clock edge
SHIFT_WIDTH	Integer > 0	Width of data[] and q[] ports

Name	Type	Description
sclr	Input	Synchronous clear input. If both sclr and sset are asserted, sclr is dominant.
sset		Synchronous set input that sets q[] output with the value specified by LOAD_SVALUE. If both sclr and sset are asserted, sclr is dominant.
shiftin		Serial shift data input
load		Synchronous parallel load. High: Load operation with data[], Low: Shift operation
data[]		Data input to the shift register. This port is SHIFT_WIDTH wide
clock		Clock Input
enable		Clock enable input
aclr		Asynchronous clear input. If both aclr and aset are asserted, aclr is dominant.
aset		Asynchronous set input that sets q[] output with the value specified by LOAD_AVALUE. If both aclr and aset are asserted, aclr is dominant.
shiftout	Output	Serial Shift data output
q[]		Data output from the shift register. This port is SHIFT_WIDTH wide

2)

1. Implement 4-bit Ripple counter with asynchronous active low set using behavioral modelling that increment from 0 to 15
   - Input:
     - clk
     - set (sets all bits to 1)
     - out (4-bits)

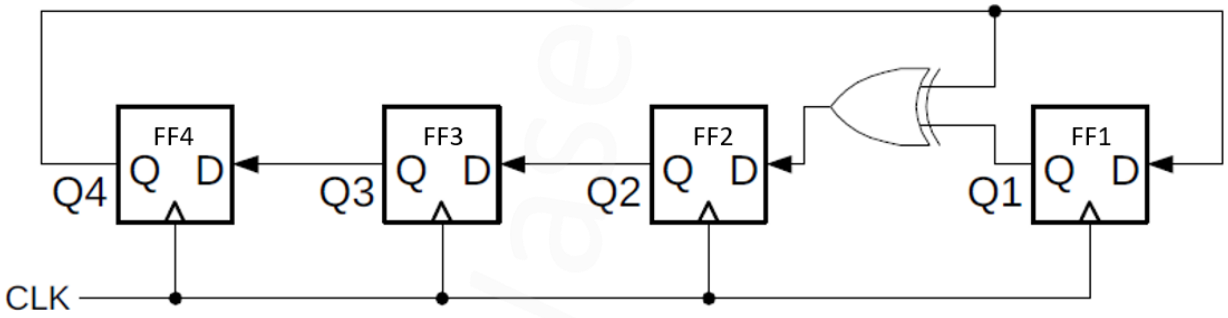
2. Use the structural counter done in the previous assignment as a golden reference.
3. Test the above behavioral design using a self-checking testbench
    - Testbench should instantiate the previous two designs

3) Extend on the previous counter done in question 3 to have extra 2 single bit outputs (div_2 and div_4). Hint: Observe the output bits of the "out" bus to generate the following

- div_2: output signal that acts as output clock with a frequency half the input clock signal.
- div_4: output signal that acts as output clock with a frequency quarter the input clock signal.

4) Implement the following Linear feedback shift register (LFSR)

- LFSR Inputs: clk, rst, set
- LFSR output: out (4 bits) where out[3] is connected to Q4, out[2] is connected to Q3, etc.



- LFSR can be used as a random number generator.
- The sequence is a random sequence where numbers appear in a random sequence and repeats as shown on the figure on the right

0001
0010
0100
1000
0011
0110
1100
1011
0101
1010
0111
1110
1111
1101
1001
0001

FF2, FF3, FF4 have the following specifications:

- D input
- Clk input
- Input async rst (active high) – resets output to 0
- Output Q

FF1 have the following specifications:

- D input
- Clk input
- Input async set (active high) – set output to 1
- Output Q

Note: the rst and set signals should be activated at the same time to guarantee correct operation

5) Implement N-bit parameterized Full/Half adder

- Parameters
  - WIDTH: Determine the width of input a,b, sum
  - PIPELINE_ENABLE: if this parameter is high then the output of the sum and carry will be available in the positive clock edge (sequential) otherwise the circuit is pure combinational, default is high
  - USE_FULL_ADDER: if this parameter is high then cin signal will be used during the cout and sum calculation from the input signals, otherwise if this parameter is low ignore the cin input, default is high
- Ports

Name	Type	Description
a	Input	Data input a of width determined by WIDTH parameter
b		Data input b of width determined by WIDTH parameter
clk		Clk input
cin		Carry in bit
rst		Active high synchronous reset
sum	Output	sum of a and b input of width determined by WIDTH parameter
cout		Carry out bit

Deliverables:

1) The assignment should be submitted as a PDF file with this format <your_name>_Assignment4 for example Kareem_Waseem_Assignment4
2) Snippets from the waveforms captured from QuestaSim for each design with inputs assigned values and output values visible.

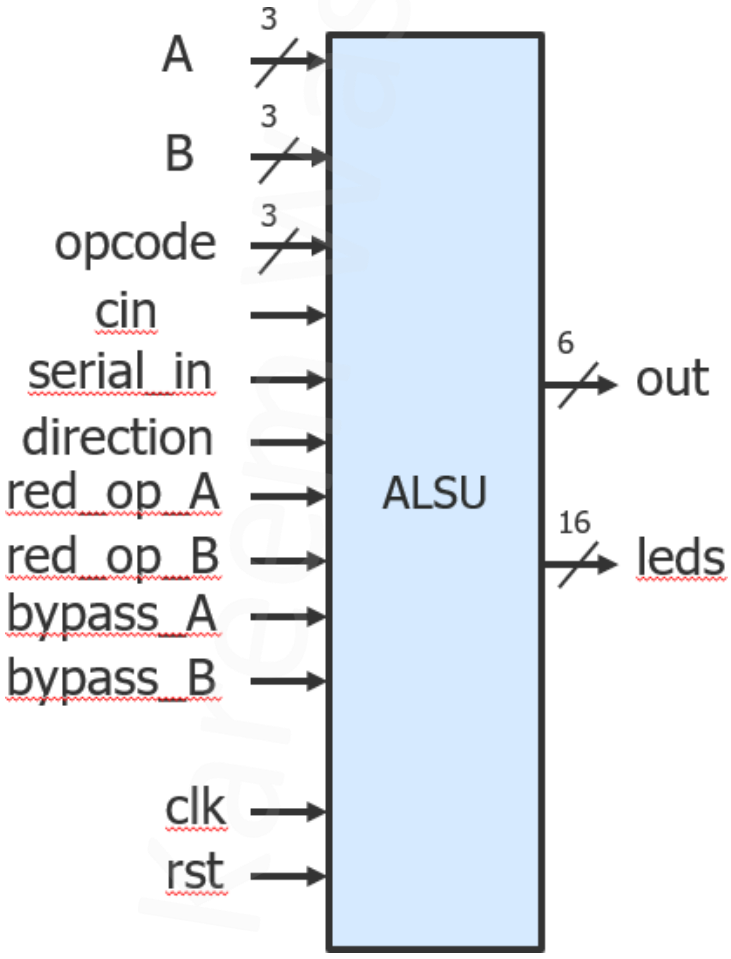
Note that your document should be organized as 5 sections corresponding to each design above, and in each section, I am expecting the Verilog code, and the waveforms snippets

Assignment 5

Design the following circuits using Verilog **and create a testbench** for each design to check its functionality. **Create a do file for both questions.**

1) ALSU is a logic unit that can perform logical, arithmetic, and shift operations on input ports

- Input ports A and B have various operations that can take place depending on the value of the opcode.
- Each input bit except for the clk and rst will be sampled at the rising edge before any processing so a D-FF is expected for each input bit at the design entry.
- The output of the ALSU is registered and is available at the rising edge of the clock.



Inputs

Each input bit except for the clk and rst will have a DFF in front of its port. Any processing will take place from the DFF output.

Input	Width	Description
clk	1	Input clock
rst	1	Active high asynchronous reset
A	3	Input port A
B	3	Input port B
cin	1	Carry in bit, only valid to be used if the parameter FULL_ADDER is "ON"
serial_in	1	Serial in bit, used in shift operations only
red_op_A	1	When set to high, this indicates that reduction operation would be executed on A rather than bitwise operations on A and B when the opcode indicates AND and XOR operations
red_op_B	1	When set to high, this indicates that reduction operation would be executed on B rather than bitwise operations on A and B when the opcode indicates AND and XOR operations
opcode	3	Opcode has a separate table to describe the different operations executed
bypass_A	1	When set to high, this indicates that port A will be registered to the output ignoring the opcode operation
bypass_B	1	When set to high, this indicates that port B will be registered to the output ignoring the opcode operation
direction	1	The direction of the shift or rotation operation is left when this input is set to high; otherwise, it is right.

Outputs and parameters

Output	Width	Description
leds	16	When an invalid operation occurs, all bits blink (bits turn on and then off with each clock cycle). Blinking serves as a warning; otherwise, if a valid operation occurs, it is set to low.
out	6	Output of the ALSU

Parameter	Default value	Description
INPUT_PRIORITY	A	Priority is given to the port set by this parameter whenever there is a conflict. Conflicts can occur in two scenarios, red_op_A and red_op_B are both set to high or bypass_A and bypass_B are both set to high. Legal values for this parameter are A and B
FULL_ADDER	ON	When this parameter has value "ON" then cin input must be considered in the addition operation between A and B. Legal values for this parameter are ON and OFF

Opcodes & Handling invalid cases

Invalid cases

1. Opcode bits are set to 110 or 111
2. red_op_A or red_op_B are set to high and the opcode is not AND or XOR operation

Opcode	Operation
000	AND
001	XOR
010	Addition
011	Multiplication
100	Shift output by 1 bit
101	Rotate output by 1 bit
110	Invalid opcode
111	Invalid opcode

Output when invalid cases occurs

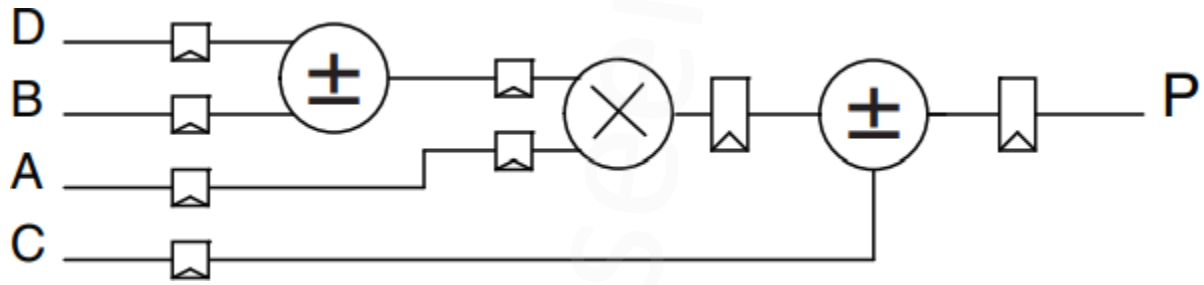
1. leds are blinking
2. out bits are set to low

Note:

- You can use in the testbench the system function "$urandom_range" which returns randomized unsigned integer within a range if you want to strict the opcode to have valid opcodes only

- signal_in = $urandom_range(5,15); //randomized between 5 and 15

Q2) Design the following simplified version of the DSP block DSP48A1 from Xilinx Spartan-6 FPGA



Port	Type	Width
A	Input	18
B	Input	18
C	Input	48
D	Input	18
clk	Input	1
rst_n (sync active low)	Input	1
P	Output	48

Parameters:

- OPERATION: take 2 values either “ADD” or “SUBTRACT”, Default value “ADD”
    - When subtracting use “D - B” and “multiplier_out – C”. multiplier_out is an internal signal

Finite State Machine

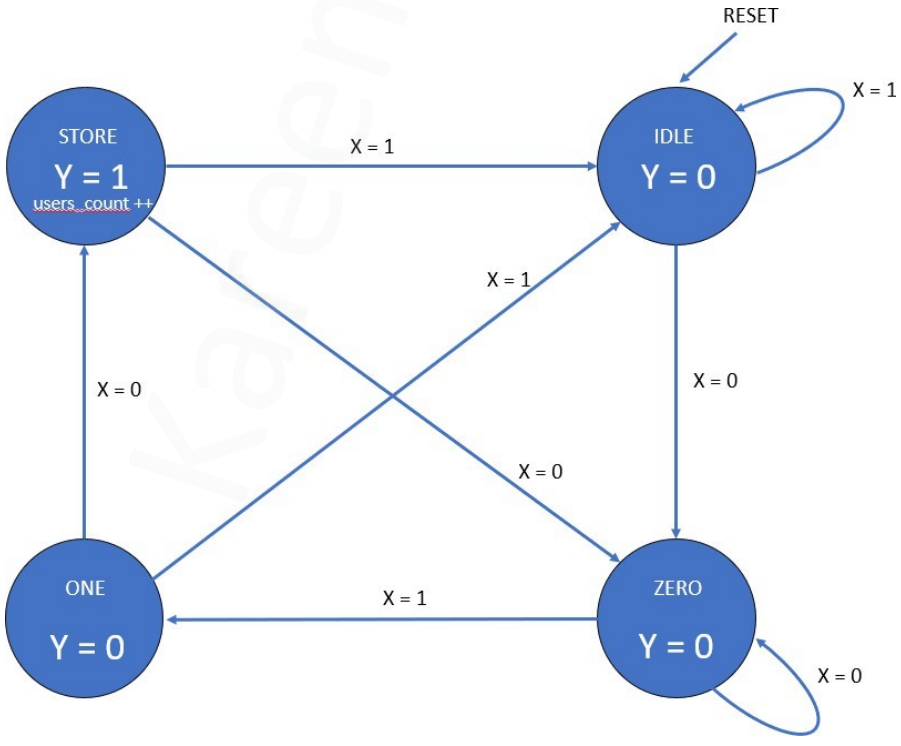
Design the following circuits using Verilog **and create a testbench** for each design to check its ~~functionality using QuestaSim. Use a **do file** for question 3.~~

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. Check the deliverables by the end of the document

Requirements:

1. Design Moore FSM that detects "010" non-overlapped pattern. (Draw state transition diagram)

Name	Type	Size	Description
x	Input	1 bit	Input sequence
clk			Clock
rst			Active high asynchronous reset
y	Output	1 bit	Output that is HIGH when the sequence 010 is detected
count		10 bits	Outputs the number of time the pattern was detected



2) 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.

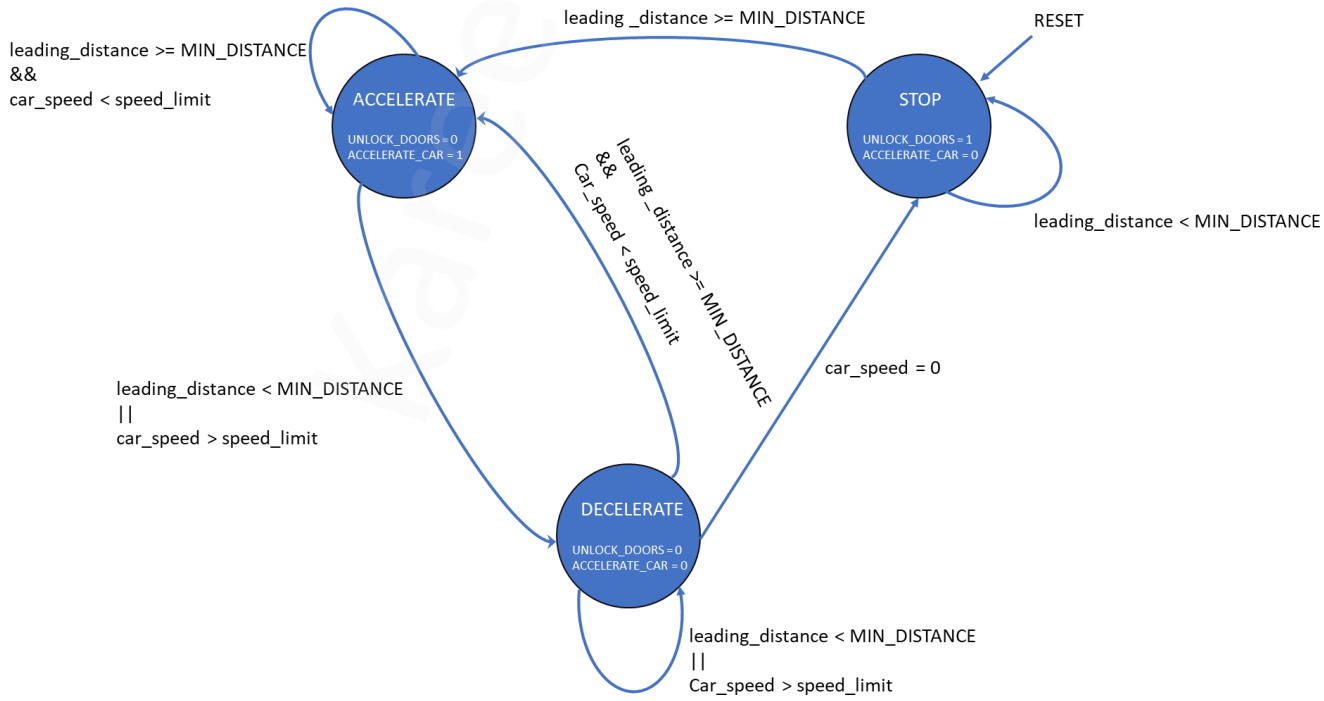
>> Consider creating realistic testbench for this design <<

Ports

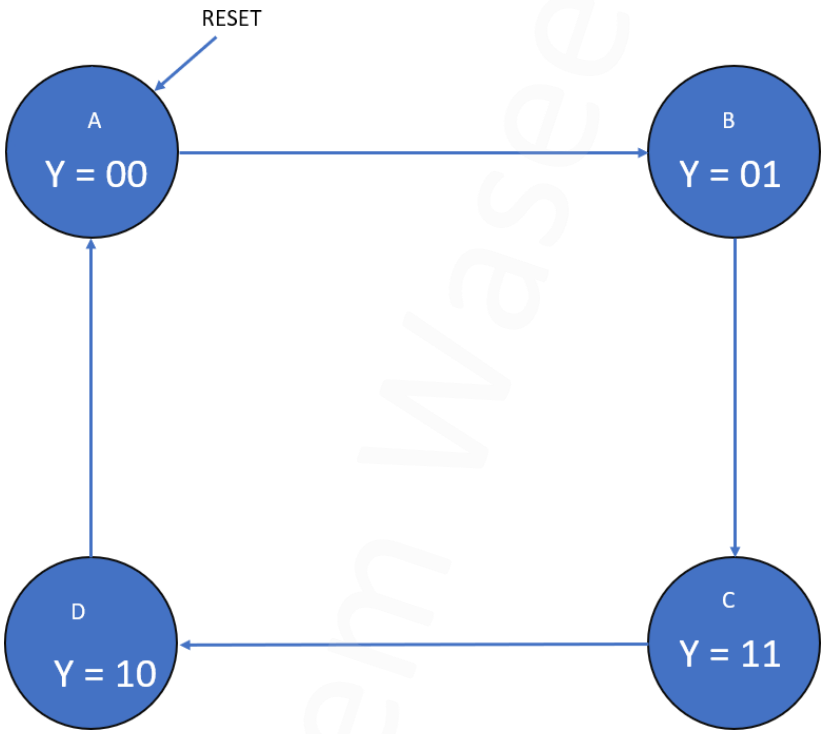
Name	Type	Size	Description
speed_limit	Input	8 bits	Allowable speed limit of the highway
car_speed		8 bits	Current car speed
leading_distance		7 bits	Distance between the car and the vehicle/object in front of it
clk		1 bit	Clock
rst		1 bit	Active high asynchronous reset
unlock_doors	Output	1 bit	Signal 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



3) Implement 2-bit gray counter using Moore FSM. Test the following by instantiating the gray counter done in the 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:

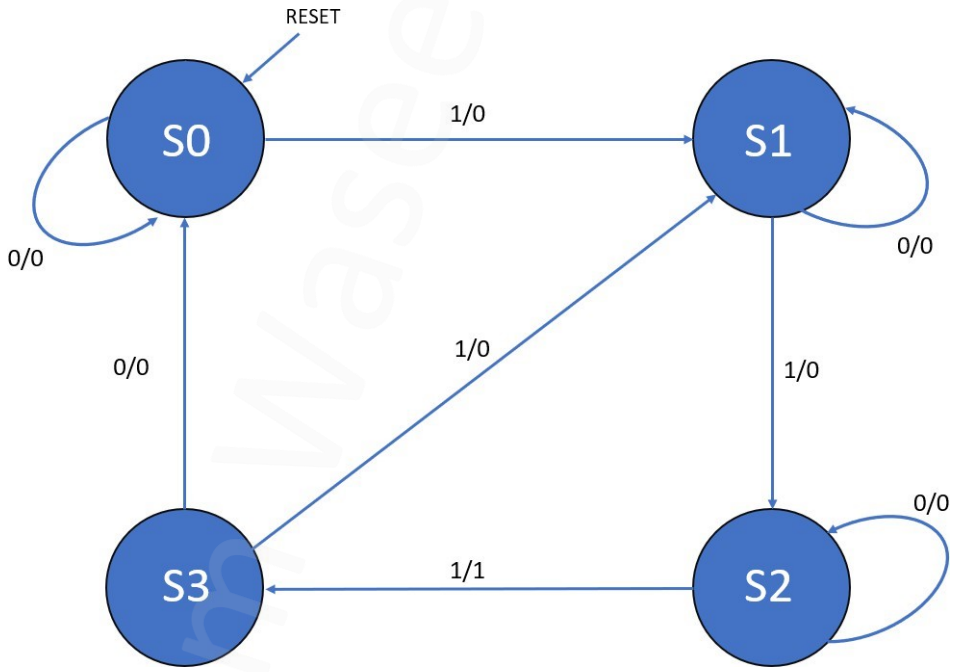
- y, 2-bit output

Create a constraint file and go through the design flow where the reset is connected to a button and the y output is connected to two leds and then generate a bitstream file

4) Create a sequence detector using a Mealy state machine. There is one input (in) and one output (y) in the Mealy state machine. If and only if the total number of 1s received is divisible by 3, the output yout is 1. The rst of the design is active high async.

FSM Encoding:

- S0 = 2'b00
- S1 = 2'b01
- S2 = 2'b10
- S3 = 2'b11



Create a constraint file and go through the design flow where the input in is connected to a switch, output y is connected to a led and the reset is connected to a button and then generate a bitstream file