# Digital design

## 1st Task – Knight rider project.

• Reference documentation

### 1. Description

• We're going to implement a four module, synthesizable RTL design in verilog, that produces a Knight-Rider (one-hot bouncing LED) pattern.

### 1.1 Modules:

i) Combinational: Decoder (3-8 one hot)
↳ Goes as output[7:0] from input [2:0]

ii) Sequential: Up/down counter (4 bits)
↳ Use async reset ∧ enable ∧ direction input

iii) Sequential: FSM (2-state: R/L)
↳ Finite state machine design to control direction based on edge flags.

iv) Top integration: coordinate all signals, generate edge flags for the FSM, and generate gated step pulse (avoid extra step when is bouncing)

### 1.2 Block diagram:

Lets see a diagram for each module and their signals:

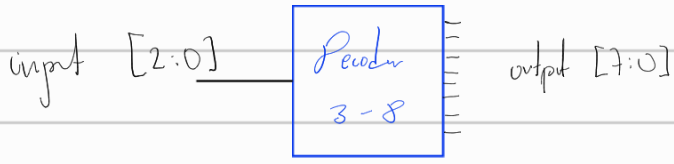
; clock: clk ; reset: asynchronous ; enable: continue pattern ; output bus [7:0]

i) 3 to 8 decoder

a) Function: Converts 3 bit input bus [2:0] into an 8 bit one-hot (or demux) output [7:0]

b) I/O: One 3-bit input ; one 8-bit output

c) Expected behavior: For each binary value of [2:0], exactly one bit of output is asserted using one-hot codification.

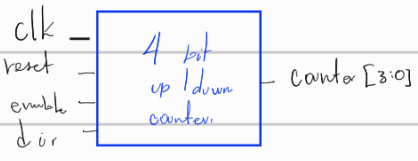



(ii) 4-bit up/down counter

a) Function: A 4-bit counter with asynchronous active-low reset, synchronous enable and a direction control input.

b) I/O: inputs: clk, reset, enable, dir.
output: counter 4 bits [3:0]

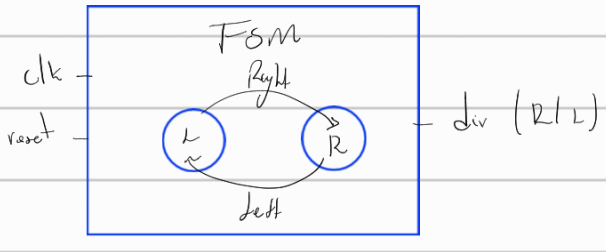
c) Expected behavior: when enabled, the finite state machine generate a 1 or 0 on their state. (if its 0, the counter goes up) and resets to zero on an asynchronous reset.




(iii.) Two state FSM:

a) Function: A minimal finite state machine with TWO states (Right/Left) that drives the direction signal for the counter.

b) I/O: inputs: clk, reset and position flags left or right
output: dir (note that its the input on the counter.



# iv) Integrated knight_Rider top module

a) Function: This top module wires together the counter, FSM and decoder to produce the bouncing one-hot led pattern.

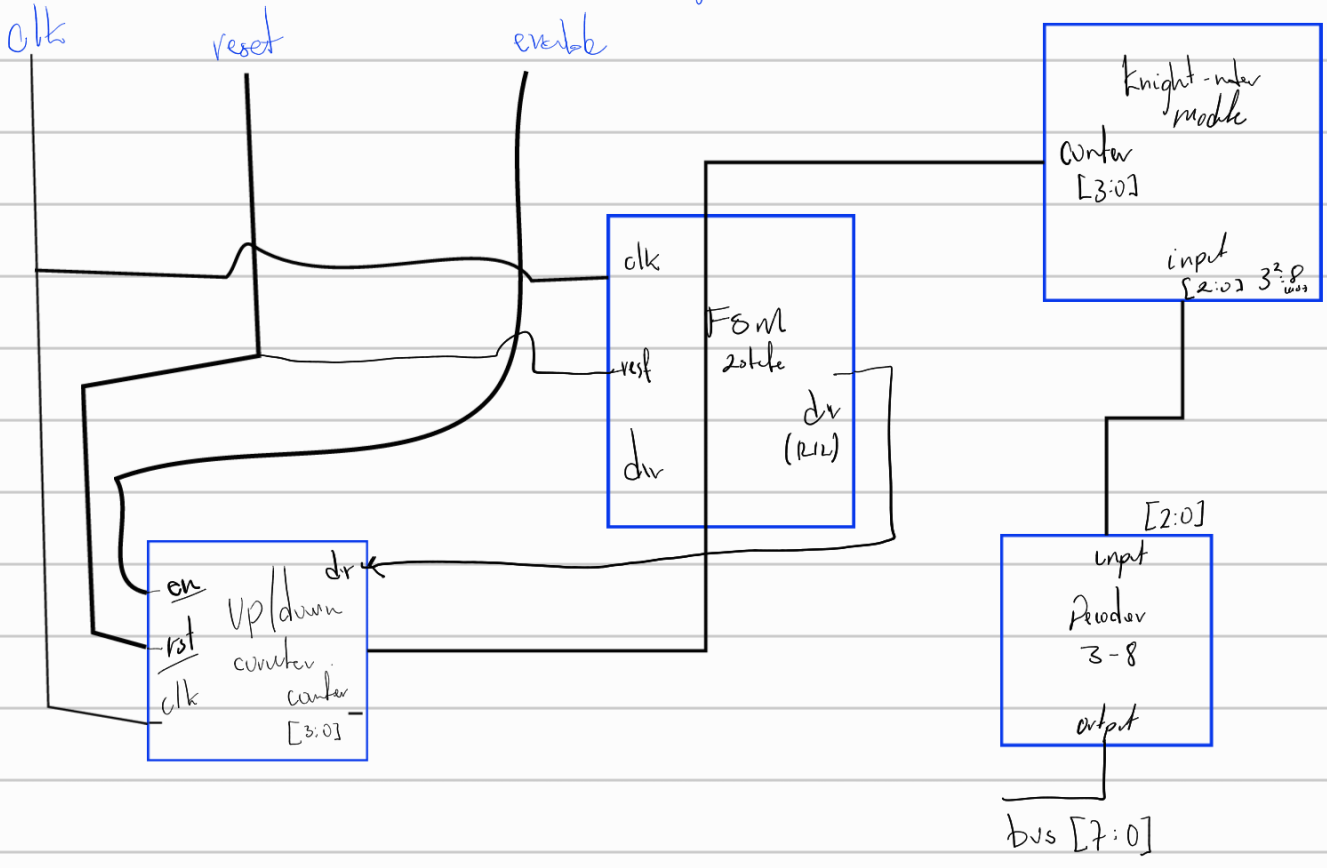
Its interface are: clk, reset, enable and output [7:0]

See that, internally, the up/down counter outputs a 4 bit count, its lower three bits of input select one of the eight posible positions. Besides, the FSM monitor the input, to see if its on the limit (to count down) or in the bottom (to cont up)

Also, the decoder converts the input into the one-hot led output.

b) i/o : input: counter / output : input [2:0]

Lets see a detailed integration:



# - 2. Test benchs, simulations and verification process:

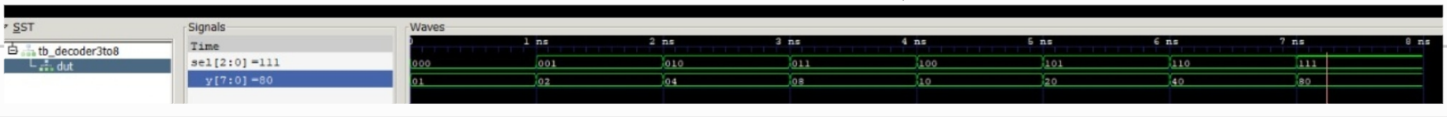
Para la verificación en este caso, podemos observar la tabla de fronteras por cada módulo, e.g, comenzando con el decoder 3-8 por one hot

i) As we know, given a 3 bit input (8 cases) we can assing one only output for the data bus of 8 bits.

Were only going to use the 8 leds cases from the 256 posible cases of the data bus:

| i/p [3:0] | output [7:0] | ⟺ hexadecimal | Active led (interpretation) |
|---|---|---|---|
| 000 | 0000 0001 | 01 | led 1 |
| 001 | 0000 0010 | 02 | led 2 |
| 010 | 0000 0100 | 04 | led 3 |
| 011 | 0000 1000 | 08 | led 4 |
| 100 | 0001 0000 | 10 | led 5 |
| 101 | 0010 0000 | 20 | led 6 |
| 110 | 0100 0000 | 40 | led 7 |
| 111 | 1000 0000 | 80 | led 8 |

from where, seeing the gtkwave simulation, the decoder module success with the objective and performance.

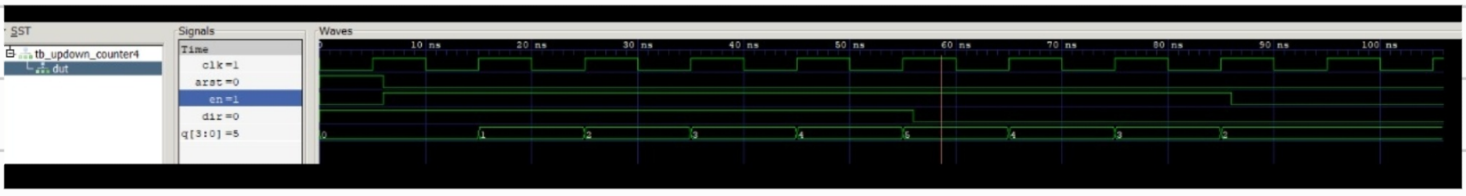



as we see, the simulation confirms correct one-hot coding behavior, curse for each input combination (8 cases), exactly one output bit is set high, following $y = 1 << sel$

The output sequence from GTKwave confirms our truth table.

(ii) Proceed with the 4 bits up/down counter:

For this case we work with many inputs and outputs, so going to test it from the tb running:




as inputs, we have: . clk is our clock (100 MHz)
. arst as our asynchronous reset.
. dir as the control direction (1=up; 0=down)
. en as the enable in order to start counting

and only one output q[3:0] that give us the actual value on the counter.

e.g: As we see on the time lapse between 80ns - 100ns, the enable is disabled (or has their actual value on low) so the value is stable(2), ie, counter doesn't continue working, as we see when the enable is high, and reach the limit to start descending with the dir = 0

(iii) We can continue now with the FSM

This finite state machine acts as direction controller for the system (presently, set the value for the "dir" variable)

The FSM presents 2 states:

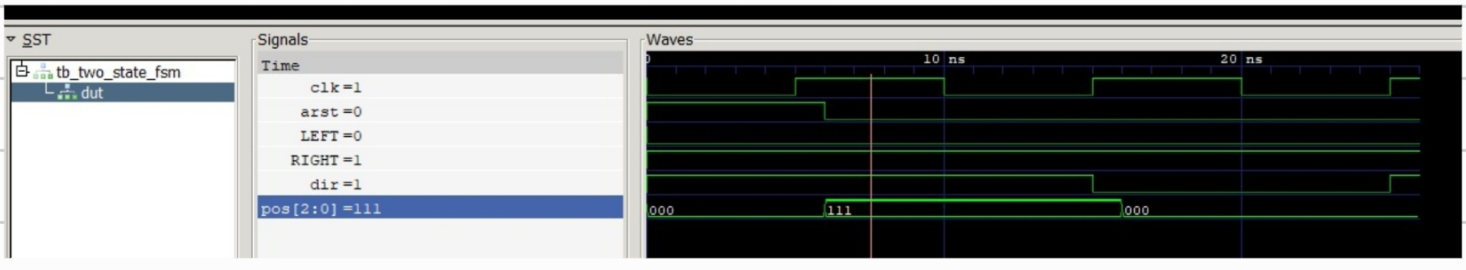
ii) Right (1): Move to right. → position increased

iii) Left (0): Move to left → position decreased

The FSM its part of the control loop of the system.

- Read the possibles cases of the counter position [2:0]

and send a controll direction to achieve.



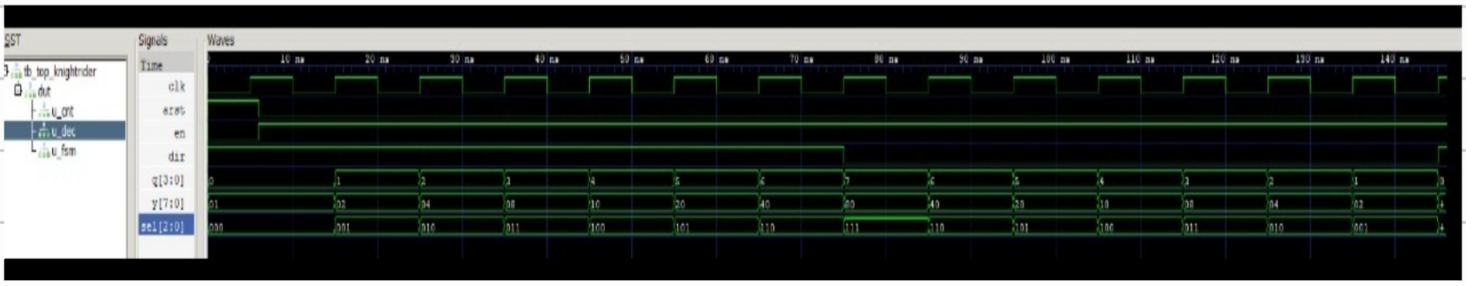

As we see on the GTKWave simulation, from the 0 - 5 ns, the async reset arst is high, so the FSM starts with the the default direction case, and move to the right, from the 6-12 ns, the FSM detects a top right limit, where pos is 111 (last case) and then the dir value change, moving the function for the counter.

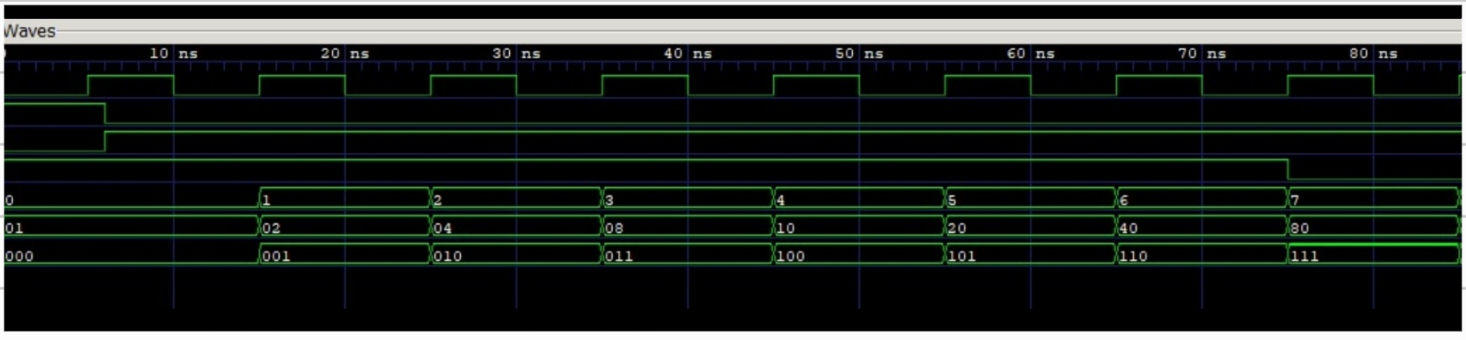
- Finally, as we see on the last time lapse, the FSM detects the other limit and restart, only working on the two cases of interest. ; pos == 000 & pos = 111

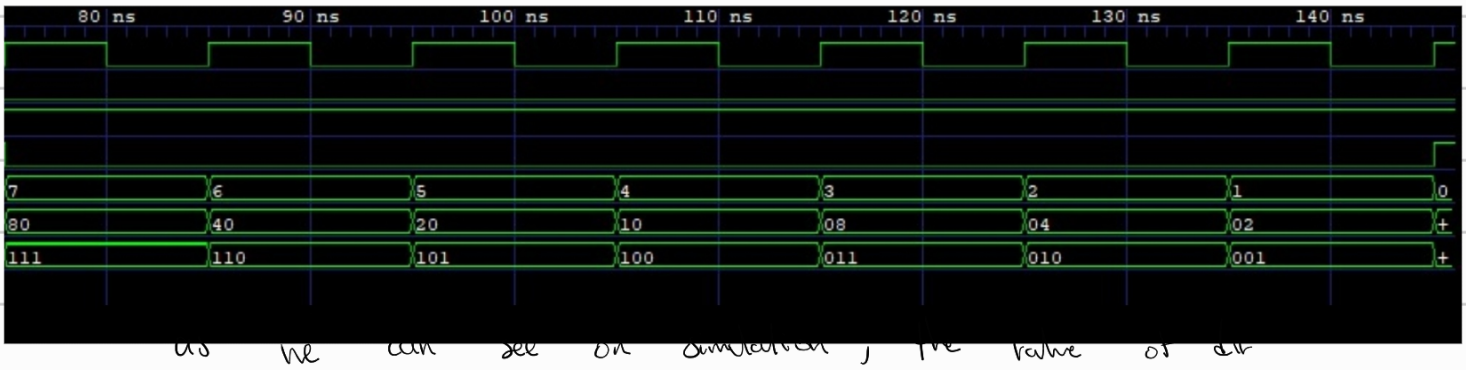
↳ This interest its relationated with a simple FSM moore, where his transition only depends for the actual position (a synchronous entry)

(v) On the last one, we integrate the modules to implementate the knight rider project.

In order to analyse the system performance, we've dist the clock, asynchronous reset, counter enable, the direction for the FSM, the binary counter value q[3:0] and finally the decoder output.




as we can see on simulation, the value of dir only change when the position led reach the limit ie, pos == 111 v 000, also, its notice that q[3:0] goes up/down. in reference what dir (synchronous) and finally y[7:0] indicate the knight rider pattern, demonstrate proper boundary detection, FSM driver direction reversal, bidirectional counting and correct mapping between bin position and led pattern using one-hot encoding.

- Indicating that, the knight rider top module, successfully integrates the FSM, counter and decoder.