ECE 270 : Embedded Logic Design ⇒ CO + DC

Resource for Quiz : hdlbits.01xz.net/wiki/Main_Page
Lab videos : youtube

Programming : 1st half → Verilog
2nd half → Embedded C

Theory : FPGA and SOC

Vivado 2019.1 (including SDK)
arch user repository ⇐ AUR ✓

* GRADES:

mid sem	30 %
end sem	30 %
surprise quiz	25 %
lab hw	15 %

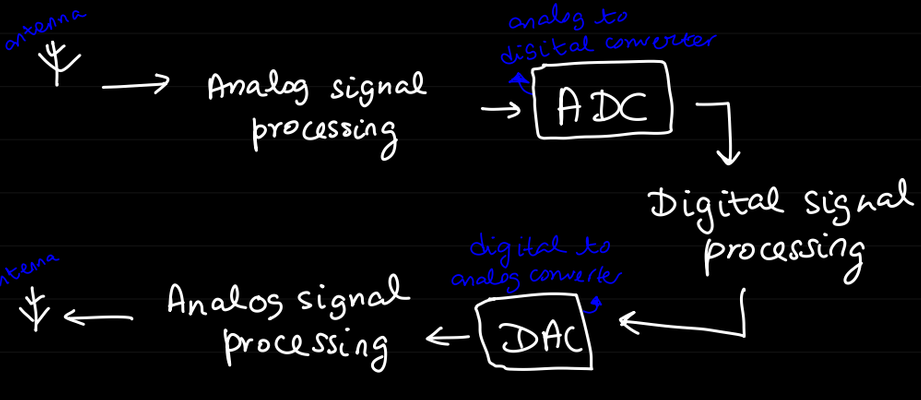
LECTURE : 1

* Which is faster : Analog vs digital ?

⇒ Depends on the use case

* No product is purely digital / analog ?

⇒ Analog is present in nature however digital can be processed easily and has more use cases.



HDL → Hardware Description language
↳ eg ⇒ Verilog

LECTURE : 2

* **combinational circuit** : The output depends upon the present input (same clock cycle)

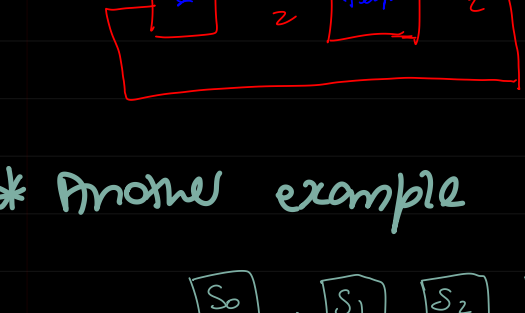
* **sequential circuit** : output depends upon the current input and the current state of the circuit

(you need current state of circuit, need to store state, hence we use flip flops. Note: we need not store current input)

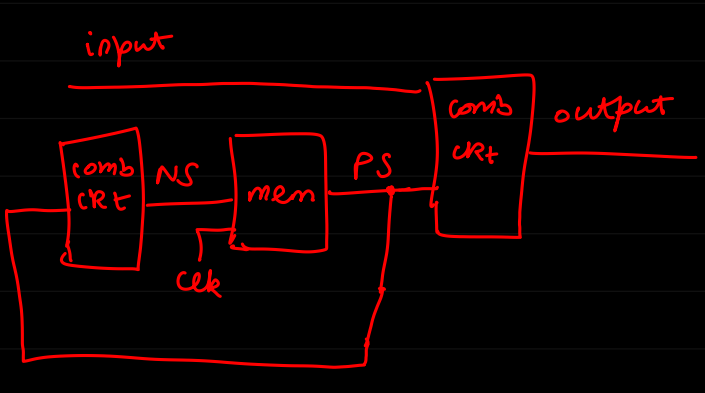
What we get → output + next state (because we go from one state to another)

⇒ Note: combinational circuits use clock as its an input as well.

* **D flip flop** (edge triggered) : input is stored at falling or rising edge of the clock (next state becomes current state)

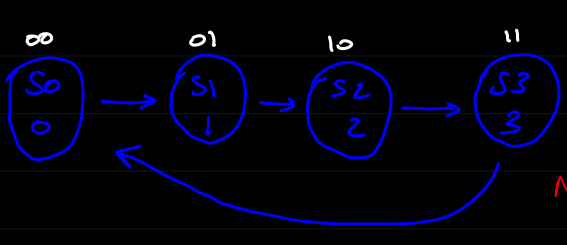


* **Sequential circuit using combinational ckt**



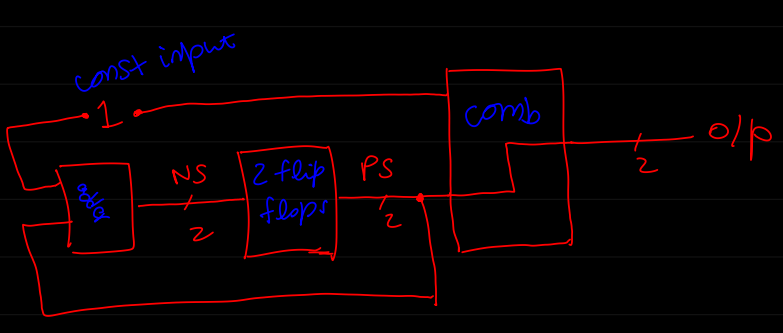
* FSM (finite state machine)

⇒ up counter

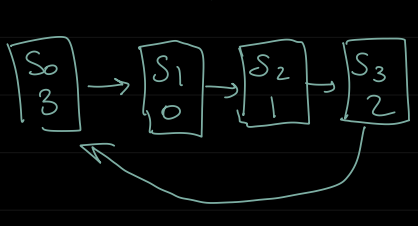


Note: if curr state = S_n, the output is n

2 bits required to store in mem

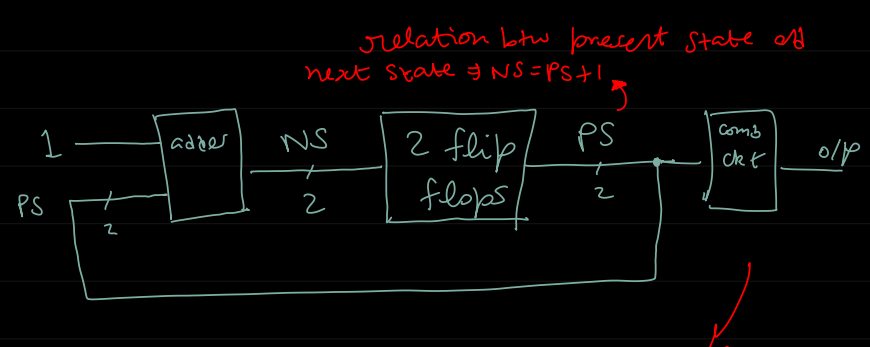


* another example



$S_0 = 00$
$S_1 = 01$
$S_2 = 10$
$S_3 = 11$

relation btw present state & next state ⇒ NS = PS + 1



eg: if PS = S_0 (00) ⇐ { Here we should use 2 not gates for the 2 bits

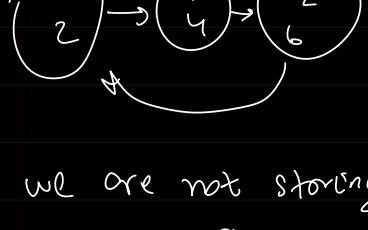
output is 3 (11)

i.e. NOT(0), NOT(0) ⇒ (11)

So, we are storing current state + processing input (using comb ckts)

* Example : 3

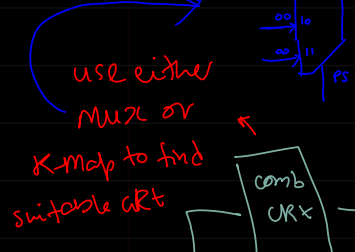
counter : 2 → 4 → 6



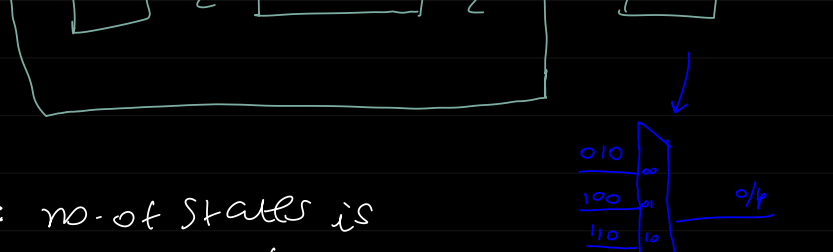
S_0 00
S_1 01
S_2 10

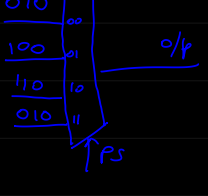
we are not storing 2/4/6 {output}
we are storing $S_0/S_1/S_2$ {state}

since we represent states using 2 bits, we need 2 flip flops



use either mux or K-map to find suitable ckt





NOTE: no. of states is equal to no. of flip flops

* Any combinational circuit can be implemented using memory (↳ lookup table) **LUT**

In DC, we learnt that we can implement any comb. ckt using gates.

However implementing it using memory is more versatile since we can always change memory (i.e. it is dynamic)

Conclusion,

Combinational ckts can be implemented using...

① Pure gates
② Lookup Table

* **FPGA** Field Programmable Gate Array

→ Array of generic logic gates

→ Gates where logic function can be programmed

→ Programmable interconnection btw gates

→ Field : System can be programmed in the field (after fabrication)

CLB : configurable logic blocks
BUFG : global buffer
BUFR : regional "
BUFIO : input/output "

* Comb CKts on FPGA
≡ Generic FPGA Design Flow

Requirements
↓
Design specifications
↓
Design Formulation
↓
Design Entry { includes: block diagram*
Hand-drawn schematics, VHDL, Verilog
↓ CAD = computer aided design
Behavioural Simulation : done using test benches / test bench waveforms (or self checking test bench)
ensures that the design is functionally correct { doesn't guarantee the ckt will work on the hardware
input: VHDL description → output: circuit netlist (gate level circuit representation) ⇐ Logic Synthesis (automatically converts code to visual representation (instead of manual K-map))
↓
Post Synthesis Simulation ⇒ we use the same test bench and test the netlist for its functionality
↓
Implementation ⇐ mapping, placement, routing ⇒ how the components are connected with each other
(maps the logic defined by an NGD file into FPGA elements; compares gate level circuit & FPGA circuit; ↳ location of diff components in the circuit in consideration with its placement requirements (eg. component temp))
↓
ASIC / SoC FPGA