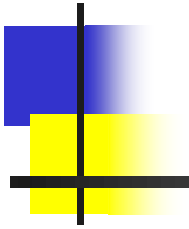


2B : Verilog for Synthesis

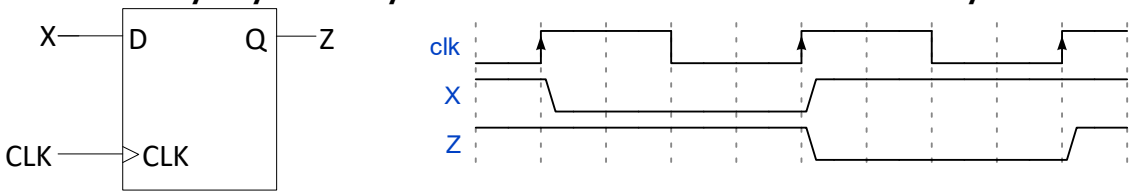
CG3207 Computer Architecture

Rajesh Panicker, ECE, NUS

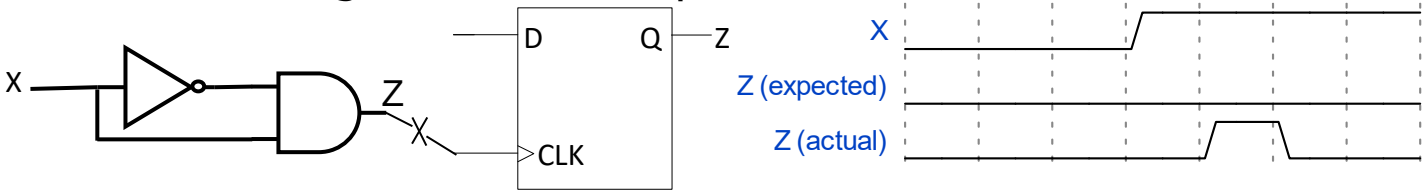
General Rules for Synthesizability

- Do NOT use delays (**#**delay)
  - Combinational (propagation) delays are hardware dependent*; not something the synthesis tool can insert based on HDL code

    *controllable to some extent through timing constraints (.xdc file) such as create_clock, set_max_delay, set_min_delay etc.
  - Clock cycle delays are inserted by the designer explicitly by introducing a physical register (i.e., it is a part of your design), and not done automatically by the synthesis tool based on #delay

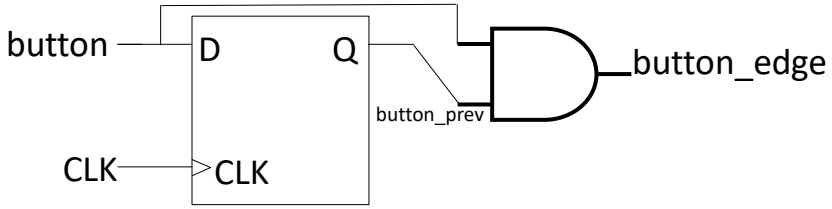


- Try to use only one clock for the entire design
  - Connect the input of every sequential element to this clock
    - Multiple clocks => more clocking resources, complicates timing analysis (related clocks), requires clock domain crossing circuitry (unrelated clocks)
  - The clock should not come from a combinational circuit, as a combinational circuit can have glitches in its output



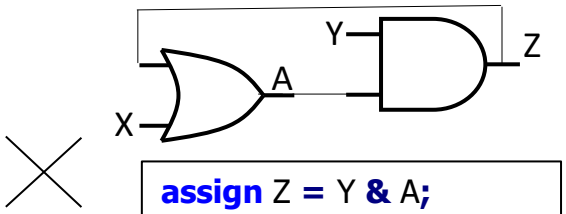
General Rules for Synthesizability…

- Do not use posedge/negedge for anything other than clock or reset of a sequential process. i.e., do not use something like @(posedge button) for detecting a transition
  - Use a synchronous edge detection scheme instead – i.e., by comparing the current value with the previous value stored in a register

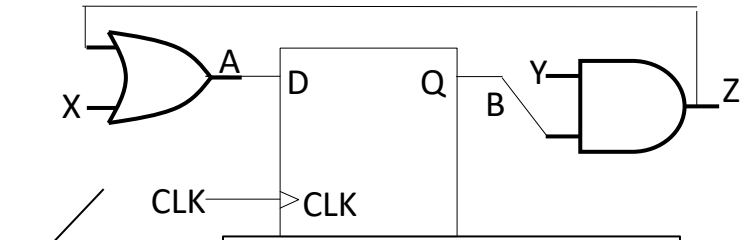



```
assign button_edge = button & button_prev;

always @ (posedge CLK)
begin
        button_prev <= button;
end
```

- Do not have combinational feedback paths
  - Every circular assignment should be broken by a register (an assignment in a synchronous always block)



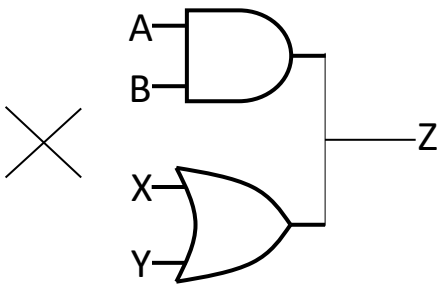

```
assign Z = Y & A;

always @ (*)
begin
        A <= X | Z;
end
```




```
assign Z = Y & B;

always @ (posedge CLK)
begin
        B <= X | Z;
end
```

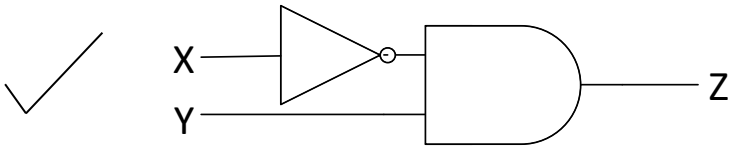
General Rules for Synthesizability…

- regs and wires should not have multiple drivers
  - A wire should appear on the left-hand side of only one assign statement. A reg should appear on the LHS of only one always block




```
always @ (*)
begin
        Z <= A & B;
end

always @ (*)
begin
        Z <= X | Y;
end
```

  - Ok to have a reg at the LHS of multiple statements within the same always block as long as the same type of assignment is used
    - Only blocking or only non-blocking, do not mix the two for a particular reg
    - The last assignment will determine the value of the reg after the always block execution



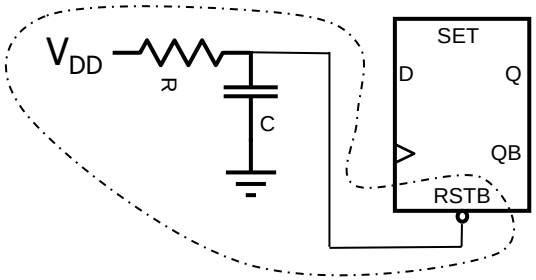

```
always @ (*)
begin
        Z = 1'b0;
if(~X)  //else is implicit
        Z = Y;
end
```

General Rules for Synthesizability…

- Initializations of regs (through an initial block or by an assignment along with reg declaration, say **reg** Z **=** 1'b0**;**) are useful for initializing registers and memories for FPGAs, but are ignored by ASIC synthesis tools
  - wires cannot be meaningfully initialized as they don't store anything
    - Initialization to 0 or 1 will connect the wire to a constant 0 or 1 respectively. Further assignment using assign will lead to it having multiple drivers
  - regs that get synthesized as combinational* circuits cannot be meaningfully initialized

    *Not all regs infer physical registers (we will see later)!
  - Do not assume registers and memory have 0s as initial value

Circuitry (simplified) to initialize to 0.
Similar functionality is built into FPGAs and asserted when programming a bitstream (aka configuration); need to be included explicitly for ASICs



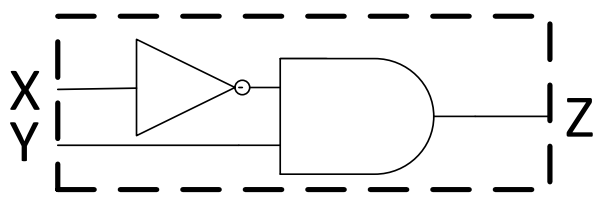
- Every always should follow one of the three templates given in the following slides strictly. **Every single** always

I : Purely combinational

Combinational : all outputs[#] are SOP or POS functions of the current inputs [#]

- Every input must be present in the sensitivity list, or simply use **(*)**. Alternatively, use continuous assignments (assign)
  - Synthesis tools generally fix this and issue a warning, but simulation won't work
- Use only blocking assignments
  - However, blocking assignments for internal calculations (variables used only internally in the always block) and non-blocking assignments for the outputs of the always block (signals used as inputs to other blocks) could be useful with some simulators such as Vivado
- Every reg must be assigned a meaningful value (not something like Z <= Z;) for every possible combination of inputs (i.e., all branches of if / case statements)
- A reg appearing on both LHS and RHS within an always block should appear on the LHS before RHS. A reg assigned using non-blocking assignment should not appear on the RHS of the same block
  - Not really an issue if * is used in sensitivity list

```
always @ (X)
begin
if(~X)
        Z = Y;
end
```



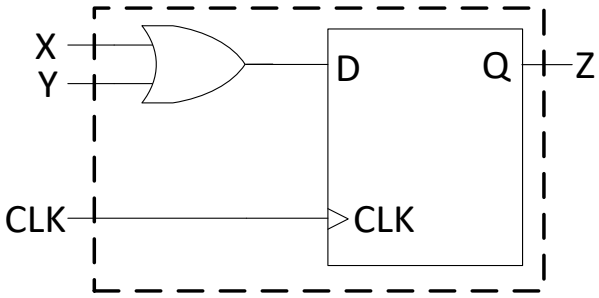

```
always @ (*)
begin
if(X)
        Z = 1'b0;
else
        Z = Y;
end
```

[#]Inputs and outputs in this slide and the following 2 slides refer to inputs and outputs of the block we are implementing, not necessarily that of the module. Inputs are system inputs or outputs from another block. Outputs are system outputs or inputs to another block.

II : Synchronous

Synchronous : output changes only on the rising or falling edge of a single clock

- Only posedge/negedge CLK should be present in the sensitivity list
- Only regs that change on the same edge of the same clock should be assigned in the always block (different time => different always blocks)
- Use non-blocking assignments for the outputs of the always block (signals), as well as for any internal physical registers
  - Keep in mind that the updated values are not available for use at the same clock edge
  - Internal physical registers can also be implemented using blocking assignments where the variable appears on RHS before LHS (not recommended)
- Use blocking assignments for internal combinational parts (variables). In this case, the variable should appear on the LHS before RHS
  - However, it is not a bad idea to move combinational parts into a separate always / assign

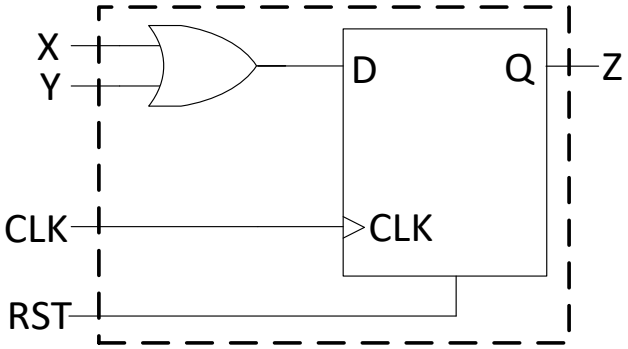



```
always @ (posedge CLK)
begin
        //your code here
        tmp = X | Y;  // tmp is a combinational variable
        Z <= tmp;     //Z is an output (signal)
         // Z <= X | Y; is fine as well
end
```

III : Synchronous with asynch set / reset

All the rules applicable for the previous template, plus

- Only posedge/negedge CLK and posedge/negedge SET/RST should be present in the sensitivity list
- Inside the if statement, output should be assigned a vector of 0's and 1's, and nothing more should be done
- All other code (i.e., the synchronous portion) should be inside the else (begin and end of else). There should not be additional *outer* if / else ifs
- It is fine to have additional ifs *inside* the synchronous part though. These ifs need not have else (why?)
- Try to **AVOID** using this - asynch. set/reset is usually avoidable, especially in FPGA designs




```
always @ (posedge CLK or posedge RST)
begin
if(RST)
        Z <= 1'b0;
        //do not have anything else here
else begin
        //the rest of the code goes here
        Z <= X | Y;
end
```

VERY IMPORTANT !!

Physical Registers Are Inferred When …

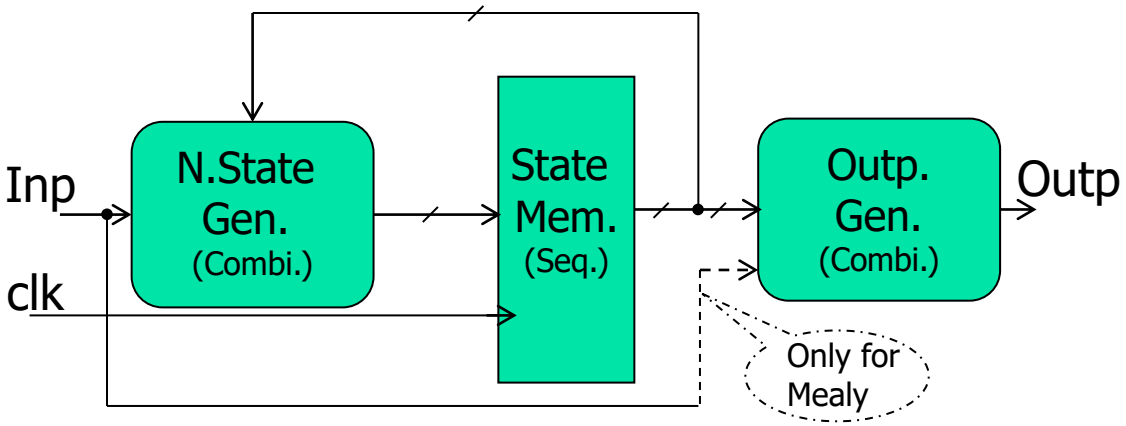
- Non-blocking assignments of regs in a synchronous always block
  - Their value changes on the ACT, and is not available for use until the next ACT
- Blocking assignment of regs in the synchronous part of an always block where they are read (on RHS) before being written (on LHS)*
  - They need to retain information between ACTs
  - Inferring registers this way is not recommended
  - Else, synthesized as wires or combinational circuits. Not all regs infer physical registers!
- *Blocking assignment also infers a register if the reg is an output, i.e., is on the RHS of a statement in another block. Nevertheless, DO NOT write code to infer registers this way
- If there are two registers storing identical content, they could be merged by the synthesis tool as a part of optimization (configurable)
- If a register content is not used in a parent module, it is optimized away, along with the combinational circuits exclusively feeding it

Example : Draw the Schematic

```
assign Z = X & Y;
always @ (posedge CLK)
begin
            B = A;
            C = B ^ Z;
            D <= C;
            E = C;
            A = E;
end
```

Note : This design does not separate combinational and sequential parts and infers registers through blocking assignments, which are not recommended practices. This example is just for illustration of concepts

State Machines in Verilog (self-reading)




```
module SM(
    input clk,
    input Inp,
    output reg Outp
//no reg if assigned using assign
    );

//only 1 bit required for state in the
example that follows.
    reg [0:0] state, n_state;
    parameter S1 = 1'b0, S2 = 1'b1;

//Implementation of state memory,
//state and output generators

endmodule
```

Note : A reset input which resets State Memory on power on is essential for ASIC designs. It is unnecessary in FPGAs as there is a dedicated Global Set/Reset (GSR) circuitry which initializes registers to their initialization values

Mealy and Moore Machines

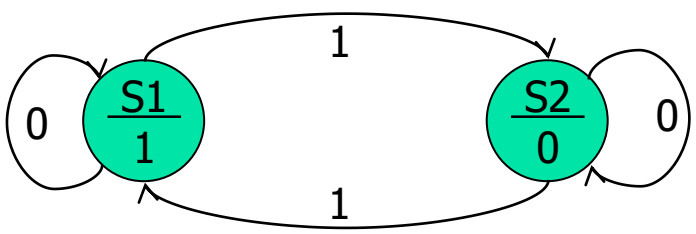
- Mealy – output depends on input(s) and state
- Moore – output depends only on state

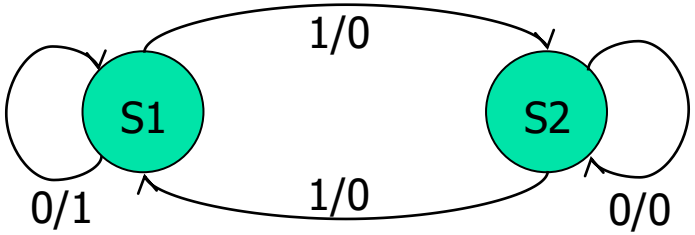
State	Inp	N-State	Outp
S_1	0	S_1	1
S_1	1	S_2	1
S_2	0	S_2	0
S_2	1	S_1	0

Moore

State	Inp	N-State	Outp
S_1	0	S_1	1
S_1	1	S_2	0
S_2	0	S_2	0
S_2	1	S_1	0

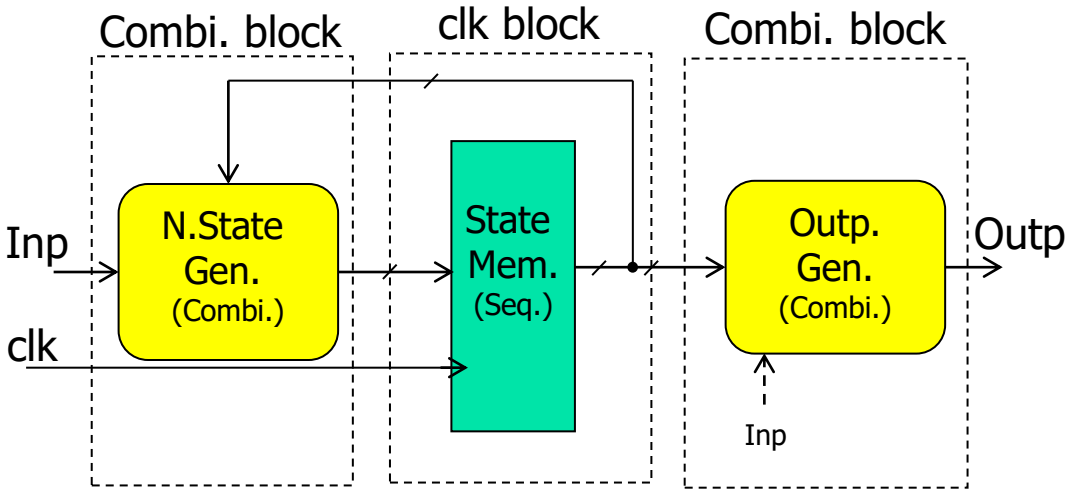
Mealy





Note : The 2 machines in this slide are **not** functionally equivalent

Moore Machine in Verilog




```
-- combinational block (output)
always @ (*)
begin
   case(state)
   // Outp is not a function of Inp
   S1 :
      Outp = 1'b1;
   S2 :
      Outp = 1'b0;
   endcase
end
```

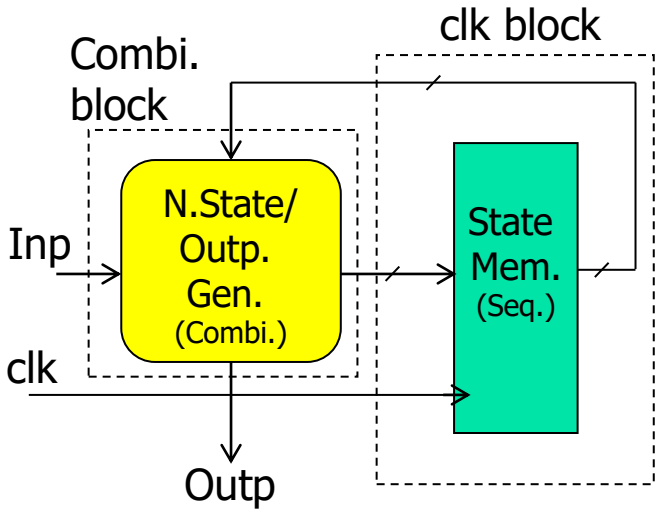
```
-- clk block
always @ (posedge clk)
   state <= n_state;
```

```
-- combinational block (next state)
always @ (*)
begin
   case(state)
   S1 :
      if(Inp == 1'b0)
      begin
         n_state = S1;
      end
      else
      begin
         n_state = S2;
      end
   S2 :
      if(Inp == 1'b0)
      begin
         n_state = S2;
      end
      else
      begin
         n_state = S1;
      end
   endcase
end
```

The combinational blocks can also be implemented using assign

Moore Machine in Verilog…

- Combined next state / output generation
  - Possible savings due to combined combinational logic optimization (depends on synthesis tool settings)




```
-- clk block
always @ (posedge clk)
    state <= n_state;
```

```
-- combinational block
always @ (*)
begin
   case(state)
   // Outp is not a function of Inp
   S1 :
      Outp = 1'b1;
      if(Inp == 1'b0)
      begin
         n_state = S1;
      end
      else
      begin
         n_state = S2;
      end
   S2 :
      Outp = 1'b0;
      if(Inp == 1'b0)
      begin
         n_state = S2;
      end
      else
      begin
         n_state = S1;
      end
   endcase
end
```

Mealy Machine in Verilog

```
-- clk block
always @ (posedge clk)
   state <= n_state;
```

- Can you implement a single always block
  - Mealy machine?
  - Moore machine?

  Hint : A clocked always block should contain only those outputs which change on a particular clock edge

```
-- combinational block
always @ (*)
begin
   case(state)
   // Outp *is* dependent on Inp
   S1 : if(Inp == 1'b0)
      begin
         n_state = S1;
         Outp = 1'b1;
      end
      else
      begin
         n_state = S2;
         Outp = 1'b0;
      end
   S2 : if(Inp == 1'b0)
      begin
         n_state = S2;
         Outp = 1'b0;
      end
      else
      begin
         n_state = S1;
         Outp = 1'b0;
      end
   endcase
end
```