Verilog Code for FSM

- CL for next state and output

  Use case statement to check the state, then set next state and output using the input (if then)

```
always @(state or a)
  case (state)
        S0:   if (a) next_state <= S3
              else next_state <= S1
        S1:   if    …
         .
         .
        default: next_state <= S0
```


Verilog Code for FSM

- Use a separate always block to store the sate:

```
always @(posedge clk or posedge reset)
  if (reset) state <= S0;
  else  state <= next_state;
```

Some Guidelines

- Partition your design into leaf cells and non-leaf cells
- For CL use `assign` statements if practical
  - if need to use `always`, in `case` statements assign the signal in all branches
- Include `default` cases in your `case` statement
- Use comments
- Use meaningful signal names, parameters for constants
- Make sure you stimulus list is complete
- Avoid
  - undefined outputs
  - incomplete specification of cases
  - shorted outputs
  - missing begin/end blocks

Common Places for Bugs!

- Clocks, registers and latches

- FPGA and ASIC designers sometimes use conservative methodologies:

  - Only use positive edge-triggered registers
  - Use a common asynchronous reset for all regs
  - Try to use common clocks for all regs, avoid gated clk, use enable instead
  - Resolve all X's and "bus conflict" messages after simulation
  - Understand all synthesis warnings, do not simply ignore them

Blocking and Nonblocking Assignments

- Inside an always block you can have
  - Blocking assignment =
    - a group of them are evaluated sequentially
  - Nonblocking assignment <=
    - are evaluated in parallel

- Use nonblocking, it's a better model for hardware!
  - real gates operate independently