Session #7

Parameters

- Verilog allows constants to be defined in a module by the keyword ***parameter***.
- Parameters cannot be used as variables.
- Parameter values for each module instance can be overridden individually at compile time. This allows module instances to be customized.
- Parameter values can be changed at module instantiation or by using ***defparam*** statement.
- Verilog also allows defining local parameters, but they cannot be changed by ***defparam*** statement.
- Local parameter are defined using the keyword ***localparam***. These cannot be overridden by ***defparam***.

Parameters

Examples:

```
module hello_world;

 parameter id_num = 0;

 initial
  $display( "Displaying Hello World ID number %d", id_num );

endmodule


module top;

defparam w1.id_num = 1;
defparam w2.id_num = 2;

helloworld w1();
helloworld w2();

endmodule
```

```
parameter port_id = 5;
parameter cache_line = 256;

// defines a register type variable
// of length cache_line

reg [cache_line - 1:0];

// definition of local parameters

localparam state1 = 4'b0000;
localparam state2 = 4'b0001;
localparam state3 = 4'b0010;
localparam state4 = 4'b0011;
```

State Machines

What is a state?

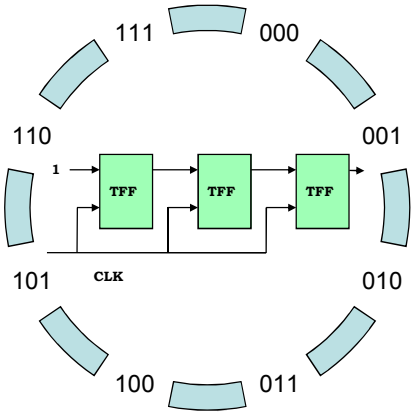
A unique value represented by a set of flip flops is called as a state.

Example:

A counter. Take a 3 bit counter. Each flip flop in the counter has got either 1 or 0 and the combination gives 8 different values. Each value can be called as a “state”.

Now, when the state is “000”, the state machine can do a certain operation, when it is “001”, it can do some other operation, so on and so forth.

A state machine is a digital synchronous circuit which performs a particular task in every given state.

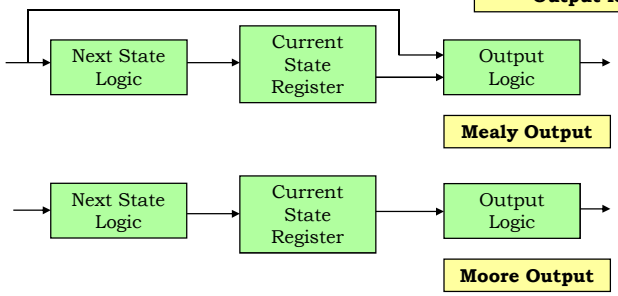


State Machines

By modeling the FSMs in a hardware description language for the use of Synthesis tools, designers can model sequences with out being overly concerned with the circuit implementation. Circuit implementation is left to the synthesis tool.

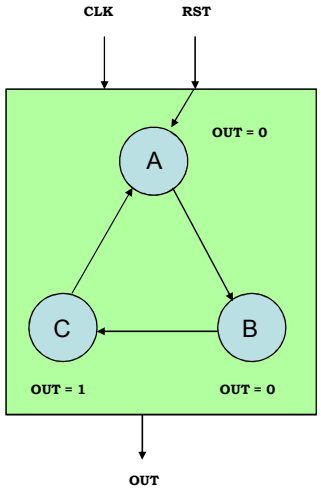
Five parts of a FSM description in Verilog

- **State registers**
- **State parameters**
- **Next state logic: Combinational**
- **Current state assignment: Sequential**
- **Output logic: Combinational**



Mixed outputs are possible for a state machine i.e. some can be Mealy and others can be Moore based on the requirement.

Illustration



Number of states = 3

OUT = 1 when in state C

FSM resets to state A on RST = 1

```
parameter A = 2'b00;
parameter B = 2'b01;
parameter C = 2'b10;
```

```
reg [1:0] current_state;
reg [1:0] next_state;
```

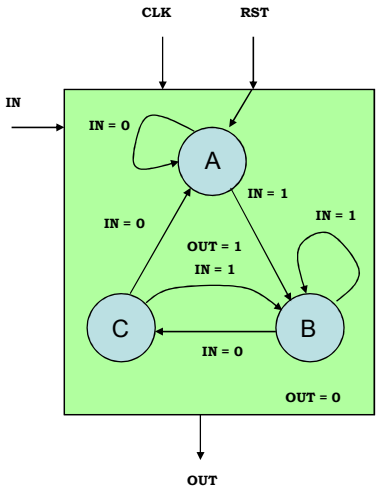
```
always@(*)
 case (current_state)
     A: next_state = B;
     B: next_state = C;
     C: next_state = A;
     default: next_State = A;
 endcase
```

```
always@( posedge CLK )
 if( RST )
  current_state <= A;
 else
  current_state <= next_state;
```

```
always@(*)
 case( current_state )
   A: OUT = 0;
   B: OUT = 0;
   C: OUT = 1;
   default: OUT = 0;
 endcase
```

If outputs are glitchy, register them.

Illustration



Consider an input to the previous case. Few things should be observed here:

1. **The transition can happen depending on the input.**
2. **Output can be a function of input as well as current state.**
3. **Output can change during transition.**
4. **Being in each state can have a certain output.**

In the given example:

General, value assigned to output is by default is 0. Only when there is a transition from state C to state B with IN = 1, only then value of output changes to 1.

Always assign default value to the outputs of the state machine.

State changes always happen in the next clock cycle (active edge) and the current state variable gets updated every clock cycle.

Modifications to state decoding block

```
always@(*)
 case (current_state)
     A:
         if (IN )next_state = B;
         else next_state = A;
     B:
         if(!IN) next_state = C;
          else next_state = B
     C:
         if(IN) next_state = B;
          else next_state = A;
     default: next_State = A;
 endcase
```

Rest of the blocks remain the same.

Modifications to output block

```
always@(*)
 case( current_state )
   A: OUT = 0;
   B: OUT = 0;
   C:
        if(IN) OUT = 1;
         else OUT = 0;
   default: OUT = 0;
 endcase
```

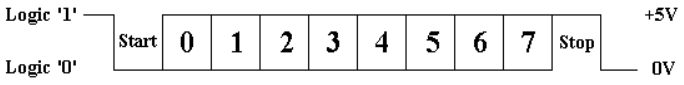
OUT = 1 will be active for exactly one clock cycle, producing a pulse at the output.

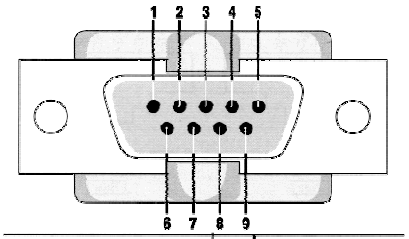
Sanity Logic

- State machines are prone to getting stuck in a particular state. There could be various reasons:
  - Not mentioning the return condition
  - Not mentioning the output transition for a particular state
  - Waiting for a signal which might never change
  - Looping among states
  - Not mentioning the default state
- As a design guideline, take care of the above conditions, that they do not occur
- Also, provide sanity logic which can restart the state machine if it gets stuck.

Reset state or providing the default state is the one of the simplest sanity logic for all state machines.

Case Study: UART





Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Designing a UART transmitter and receiver and the strategy to verify them. It is important to bring this up to the standard of a product. How to achieve this?

Mini Project

Exercise 1

- Design a state machine which can identify two given pattern 101 and 1101
  - Design non-overlapping Moore machine
  - The same state machine should find both patterns.
  - Model it in Verilog and verify its functionality

Assume necessary conditions to complete the specifications given above and come up with a strategy to make this a complete IP or a unique product.

Try designing overlapping mealy machine for this exercise as an assignment.

90 Minutes

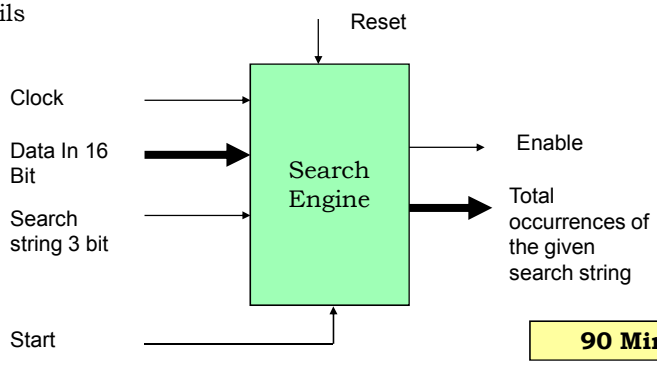
Exercise 2

- Design self checking test benches for the following designs;
  - 4:1 Multiplexer
  - Serial In Serial Out Register
  - Universal UP/DOWN counter
  - Divide by 3 counter 50% duty cycle

240 Minutes

Exercise 3

- Design and model a circuit which can find any three bit pattern in a the input data. Refer to the following block diagram to identify the details

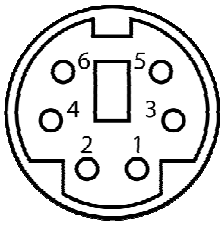


90 Minutes

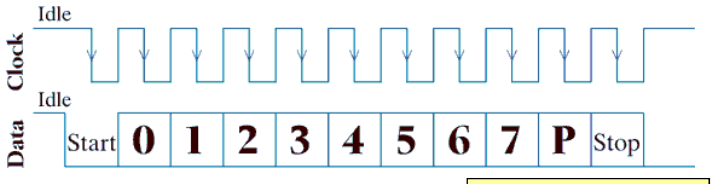
Write a test bench to verify the above functional model

Exercise 4

Design a PS/2 receiver and verify the same. The receiver should extract 8 data bits from the serial stream and provide it at the output of the module. Provide an enable signal for the output data to assert its validity.



Pin 1	+DATA	Data
Pin 2	Not connected	Not connected*
Pin 3	GND	Ground
Pin 4	Vcc	+5 V DC at 275 mA
Pin 5	+CLK	Clock
Pin 6	Not connected	Not connected**



90 Minutes