State Machines
- We can use the truth table to represent the current state and next state based on the input of a sequential circuit.
- However, truth tables are less straightforward to represent the transitions between state. So, we introduce a graphic way to represent the transitions — state machines.

- A state machine is a general method of representing a time/history dependent process.
- It uses circles to represent states and edges to represent the transition from a state to another.
- A simple example of using state machine is to represent open and close a door.
  - If a door is currently open and you pull the door, it’s still open.
  - If a door is currently open and you push the door, it’s closed.
  - If a door is currently closed and you push the door, it’s still closed.
  - If a door is currently closed and you pull the door, it’s open.

  • In this example, the door has two states: open and closed. The actions, pull and push, can change the state of the door depending on the door’s current state.
  • We can draw a state machine for this example.

![State Machine Diagram]

- The difference between this example and the sequential circuits is that sequential circuits usually have outputs.
- Therefore, we need to introduce outputs into the state machines.

Moore Machine
- Moore machines let the output associate with the state.
- The input in Moore machines decides the next state.
- The output in Moore machines only depends on the state you transit into, not on how you got into the state (the input value).
- Here is a Moore machine describing the transitions among three states.

![Moore machine diagram]

- In the circle mm/nn represents state/output; x on the edges indicates the input.

- **Example**
  - Design a controller for an elevator.
  - The elevator can be at one of the four floors: Ground, First, Second, and Third.
  - There is a button that controls the elevator, and it has two possible values: up and down.
  - The elevator goes up one floor every time you set the button to “up”, and goes down one floor every time you set the button to “down”.
  - There are two lights in a row in the elevator that indicate the current floor. Both lights off (00) indicates the ground floor; The left light off and right light on (01) indicates the first floor; The right light off and left right on (10) indicates the second floor; Both lights on (11) indicates the third floor.
  - Each time, the controller checks the current floor and current input before changing floors and lights.
  - Draw a Moore machine diagram for the specification.
Implement the Moore Machine Using VHDL

- After we have the Moore machine ready, we can use VHDL to implement the circuit.
- In a VHDL program, a state machine circuit needs a state variable to hold the current state. From the perspective of circuit, the current state is held in flip-flops. So, the number of states of a state machine determines the number of flip-flops/memory bits required.

```vhdl
-- Implement the Elevator Controller
library ieee;
use ieee.std_logic_1164.all;

entity elevator is
  port (
    clk: in std_logic;
    reset: in std_logic;
    input: in std_logic;
    output: out std_logic_vector (1 downto 0)
  );
end entity;

architecture behavior of elevator is
  -- declare an enumeration type for the state machine's states
  type state_type is (s0, s1, s2, s3);
  -- declare a signal/variable used to store current state
  signal state: state_type;
  begin
    -- the process that describes the transitions of a state machine must be clocked
    -- and must contain an if statement that checks the rising edge of clk
    process (clk, reset)
      begin
        if reset = '1' then
          state <= s0; -- initialize state machine at the startup
        elsif (rising_edge(clk)) then
          -- the state machine behavior transitions are defined with case statement
          case state is
            when s0 =>
              if input = '1' then
                state <= s1;
              else
                state <= s0;
              end if;
            when s1 => if input = '1' then state <= s2; else state <= s0; end if;
            when s2 => if input = '1' then state <= s3; else state <= s1; end if;
            when s3 => if input = '1' then state <= s3; else state <= s2; end if;
            end case;
          end if;
      end process;
    -- the process describing the outputs and the associated states
    process (state)
      begin
        case state is
          when s0 => output <= "00";
          when s1 => output <= "01";
          when s2 => output <= "10";
          when s3 => output <= "11";
        end case;
      end process;
  end behavior;
```
- A testbench for this elevator Moore machine.

```vhdl
-- A testbench for elevator Moore machine
library ieee;
use ieee.std_logic_1164.all;

entity testElevator is
end entity;

architecture behavior of testElevator is
  component elevator
    port ( 
      clk: in std_logic;
      reset: in std_logic;
      input: in std_logic;
      output: out std_logic_vector (1 downto 0)
    );
  end component;

  -- local signals
  signal clk: std_logic;
  signal reset: std_logic;
  signal input: std_logic;
  signal output: std_logic_vector (1 downto 0);

  constant num_cycles: integer := 7;

  begin
    elevator1: elevator port map (clk, reset, input, output);
    reset <= '1', '0' after 1 ns;

    process begin
      for i in 1 to num_cycles loop
        clk <= '0';
        wait for 1 ns;
        clk <= '1';
        wait for 1 ns;
        end loop;
        wait;
      end process;

      input <= '1', '0' after 1 ns, '1' after 3 ns, '0' after 9 ns;
  end behavior;
```

- This test code generate a sequence of input 10011111000000

```vhdl```
- Compile and run the code. We can get a gtkwave like this.