Building State Machines

Two Types of Circuits in Digital System

Combinational logic circuits: The outputs depend solely on the inputs.

Assignment 3 Calculator

Sequential logic circuits: The outputs (next state) depend not only on the inputs, but also on the present state of system.

Project 1 FSM

Key features of a system can be modeled as FSM

- A state machine is a digital device that traverses through a **predetermined** sequence of states in an orderly fashion.
- The system can be described by a **finite** set of states.
- The system has a finite set of inputs and/or events that can **trigger transitions** between states.
- For each state the system may be in, the behavior is defined for **each possible input or event**.
- The system has a particular **initial state (reset states)**.

Two Types of State machine

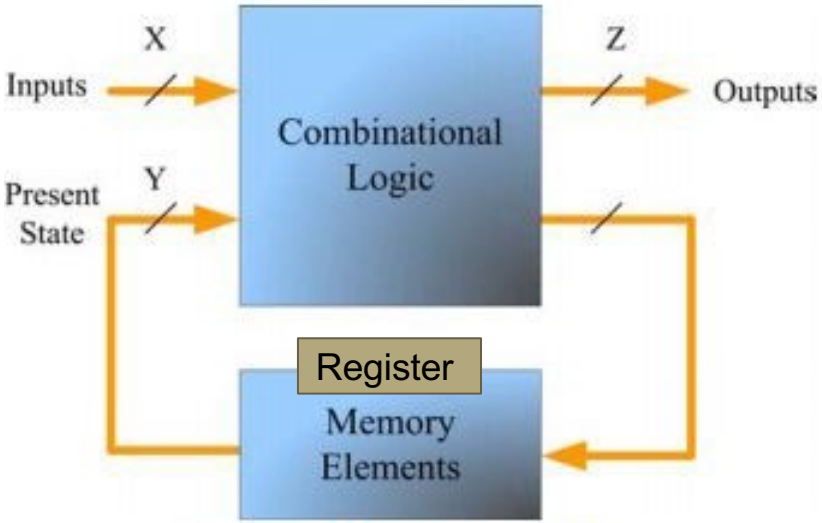


Figure 1: Mealy Type Machine

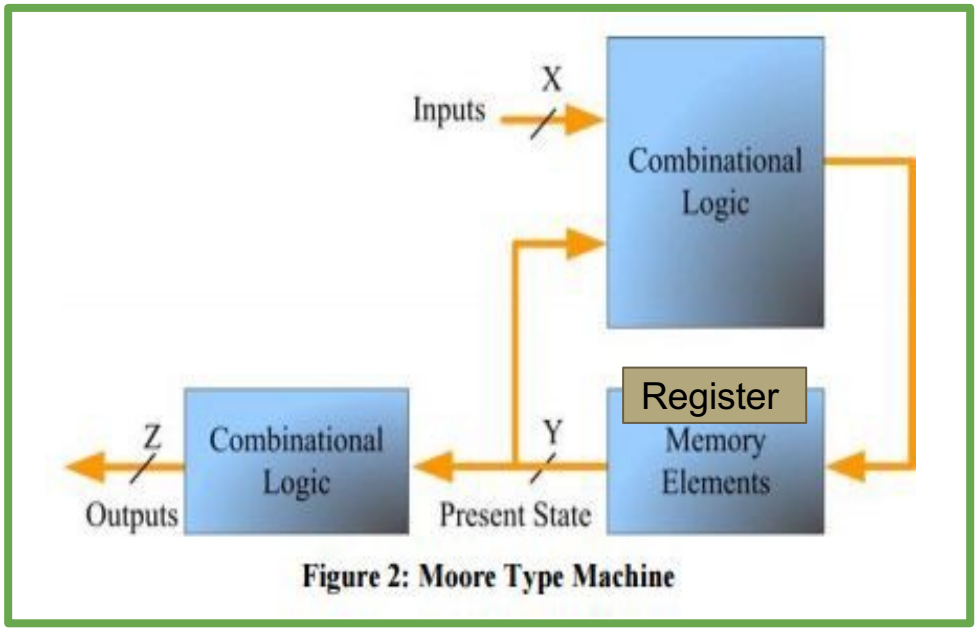


Figure 2: Moore Type Machine

Mealy machine: Output values are determined both by its current state and the current inputs

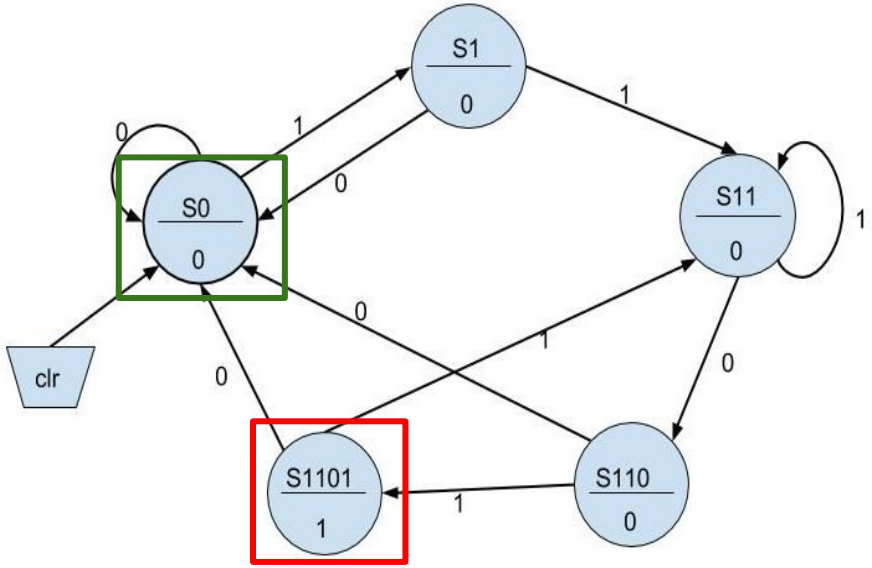
Moore machine: Output values are determined solely by its current state

Project 1. Sequence Detector using FSM

Example: Detect Input Sequence

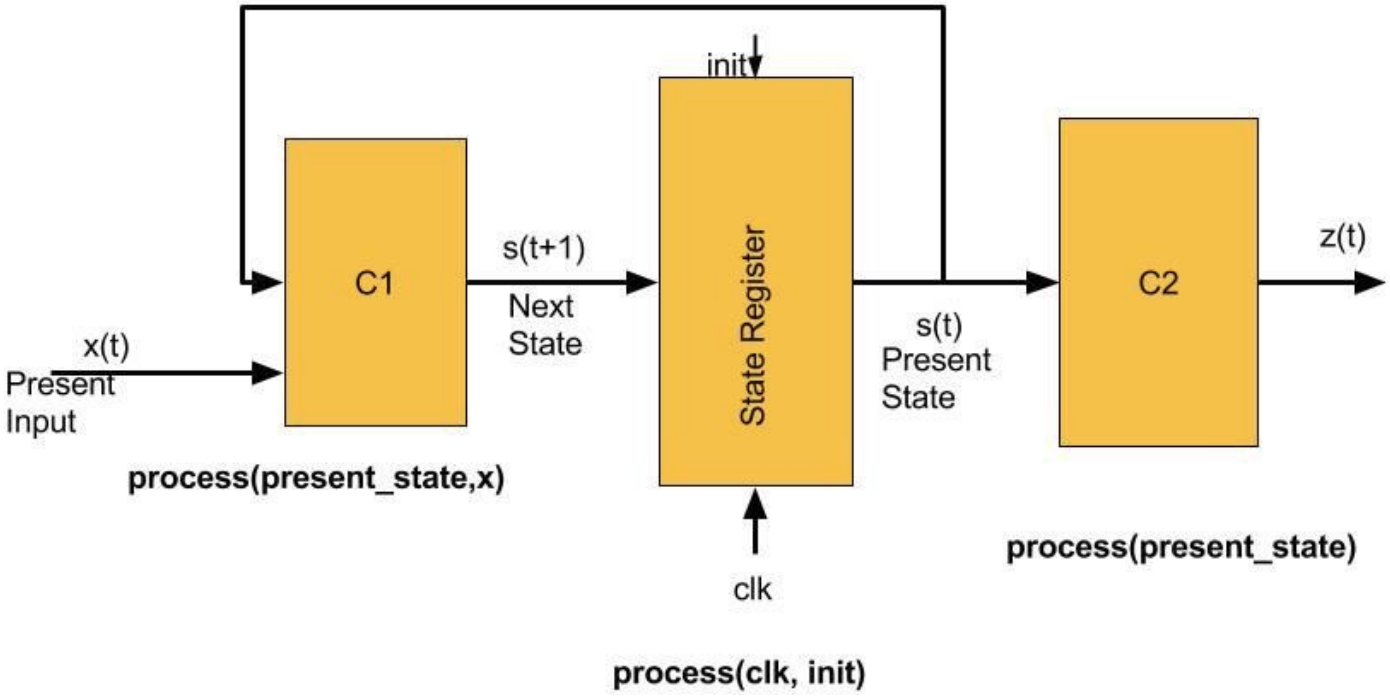
1101

(Overlap is allowed)



Input:	1	0	1	1	0	1	1	0	1	0	0	1	1	0	1	0
Output:	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0

VHDL Moore Machine Block Diagram



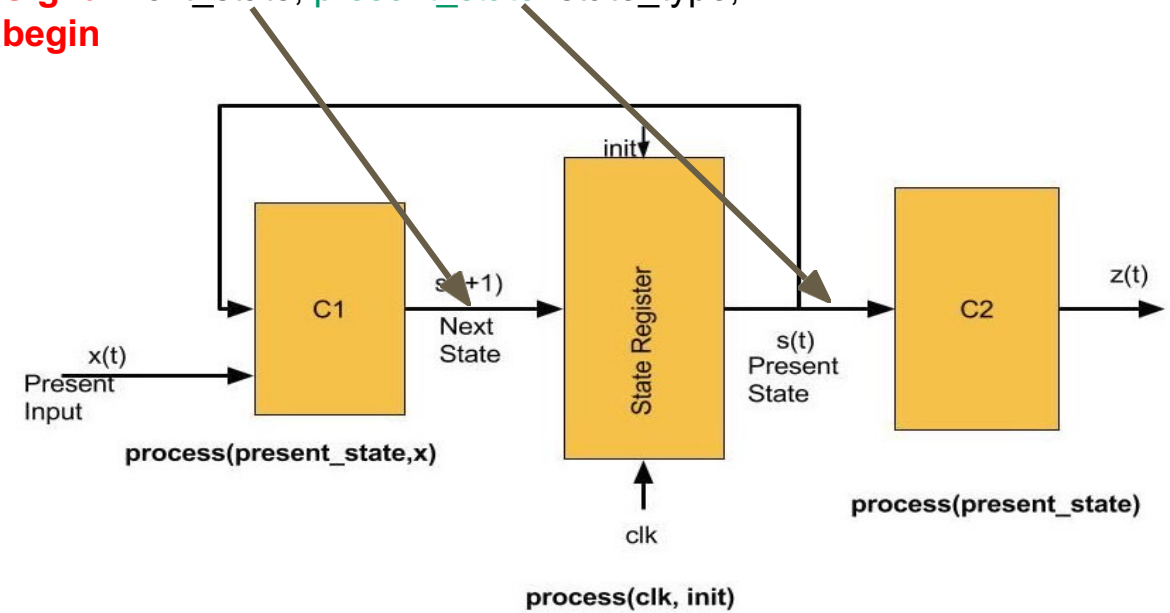
Moore Machine:
The output z(t) only depends on present state

Moore Machine Top Level

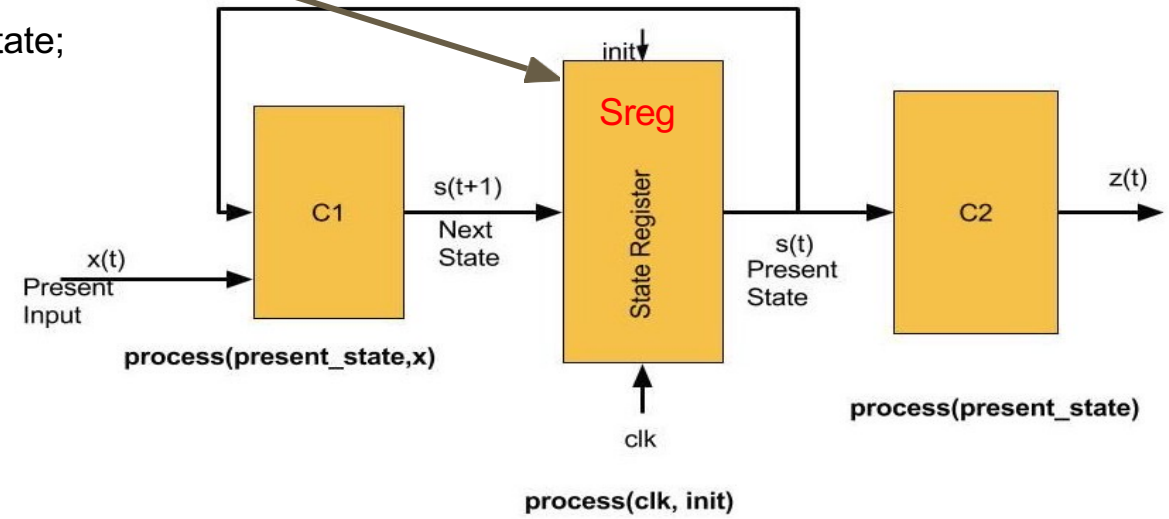
Interface Specification


```
--sequence detector
Library IEEE;
Use IEEE.STD_LOGIC_1164.all;
Entity fsm is
        Port ( clk: in STD_LOGIC;
               init: in STD_LOGIC;
               Din: in STD_LOGIC;
               Dout: out
STD_LOGIC);
end fsm;
```

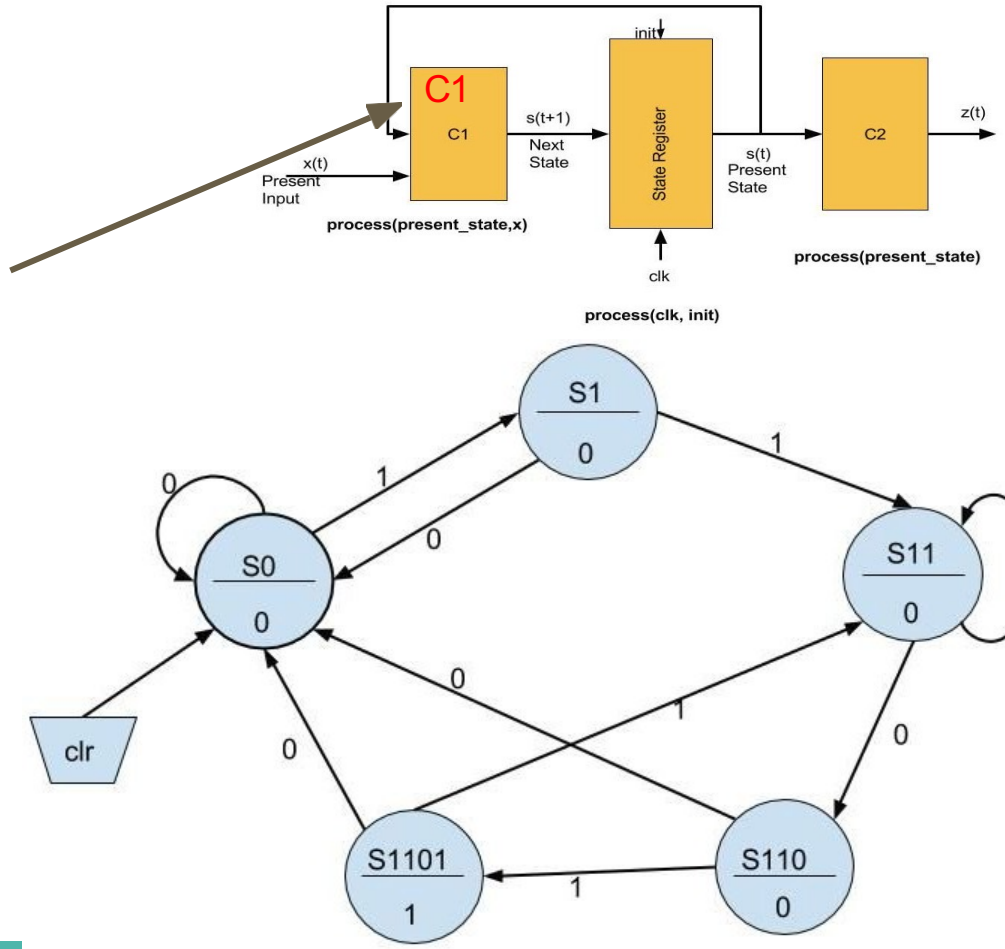
Architecture fsm **of** fsm **is**
type state_type **is** (s0, s1, s2, s3, s4);
Signal next_state, present_state: state_type;
begin

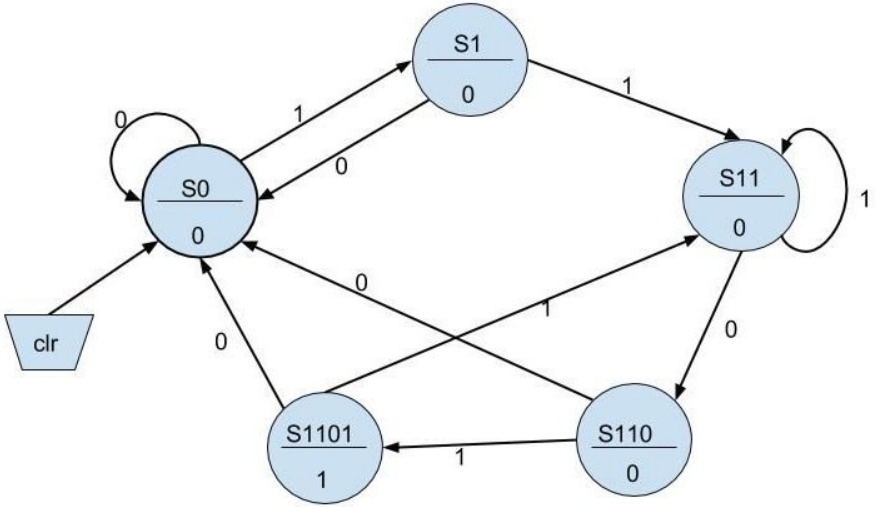


```
Sreg: process (clk, init)
Begin
        If init = '1' then
                present _state <= s0;
        Elsif clk'event and clk='1' then
                Present_state <= next_state;
        end if;
End process;
```



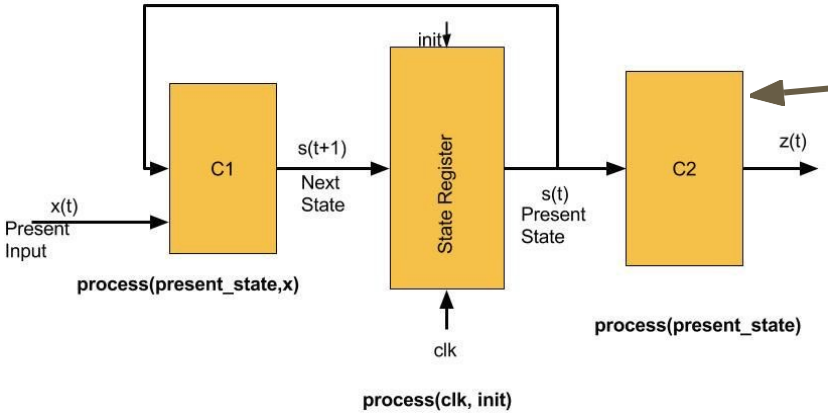
```
C1: process (present_state, din)
Begin:
        Case present_state is
                When s0 =>
                        If din = '1' then
                        Next_state <= s1;
                        Else
                          Next_state <= s0;
                        End if;
                When s1=>
                        If din = '1' then
                        Next_state <= s2;
                        Else
                          Next_state <= s0;
                        End if;
```





continue...

```
When s2 =>
        If din = '0' then
          Next_state <= s3;
        Else
          Next_state <= s2;
        End if;
When s3 =>
        If din = '1' then
          Next_state <= s4;
        Else
          Next_state <= s0;
When s4 =>
        If din='0' then
          Next_state <= s0;
        Else
        next_state<= s2;
        End if;
When others =>
        Null;
End case;
End process;
```




```
C2: process(present_state)
Begin
        If present_state = s4 then
                Dout <= '1';
         Else
                Dout <= '0';
         End if;
End process;
End seqdata;
```

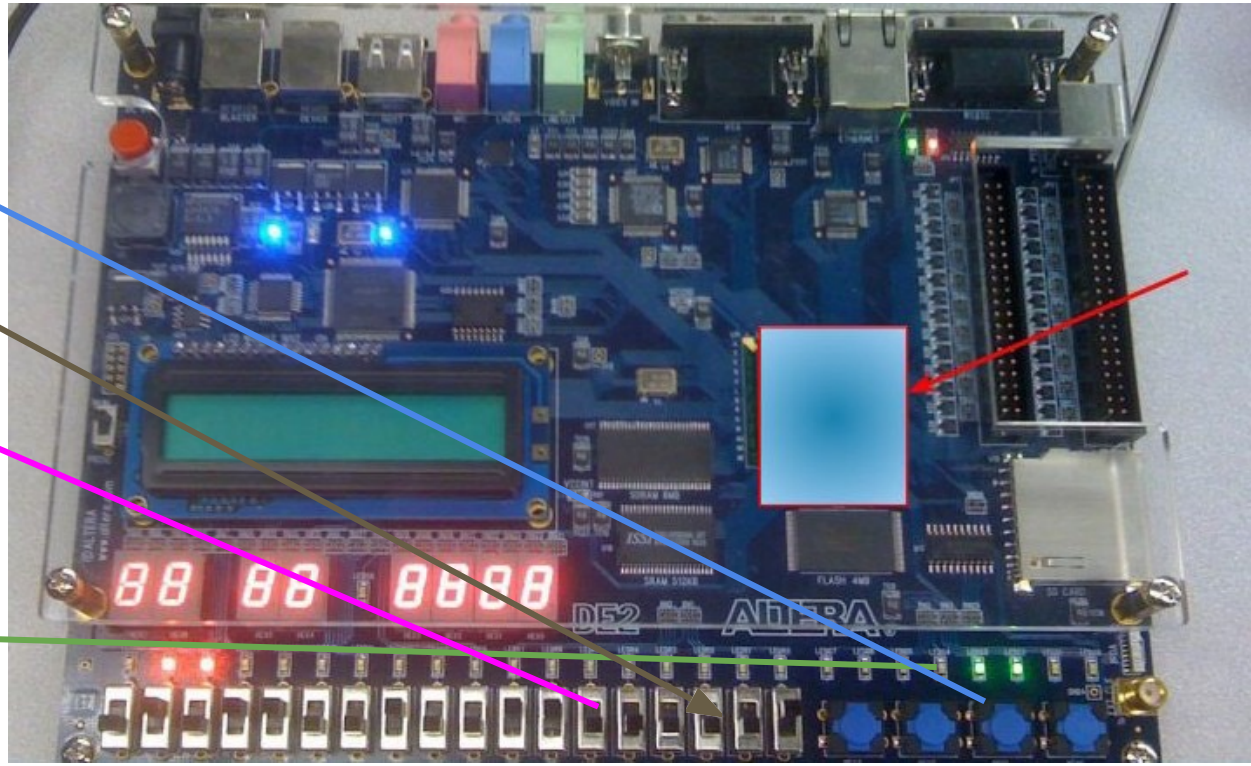
DE1 Board download

Clk : Key (Pulse)

Din : Toggle Switch

Init : Go back to the initial state

Correct_Out: Green Led



Clock Division

The frequency divider is a simple component which objective is to **reduce the input frequency**. The component is implemented through the use of the ***scaling factor*** and a ***counter***. The ***scaling factor*** is the relation between the input frequency and the desired output frequency:

$$\text{Scale} = \frac{50\text{MHz}}{100\text{Hz}} = 500000$$

The ***counter*** is an n-bit counter will start counting the number of clock pulses.

Output Clock Frequency	Scale
50M Hz	1
1M Hz	50
1 Hz	50,000,000

Example of 3 bit Binary Counter

Clock Pulse	1st bit (MSB)	2nd	3rd (LSB)
0	0	0	0 (initial state)
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Remain in State

- As you can see, if you use 2nd bit of output of counter as a clock input to another source, the frequency provided is essentially half of original frequency, while it is 1/4th of the original frequency if we use the 1st bit as output!
- After clock division, if we generate a 6 Hz clock to drive this state diagram then a delay of 1 second is achieved by staying in a state for 6 clock cycles. Similarly, a delay of 5 second is achieved by staying in a state for 30 (6 * 5)clock cycles. The counter variable in the state diagram will be reset to zero when moving to the next state after a timeout.