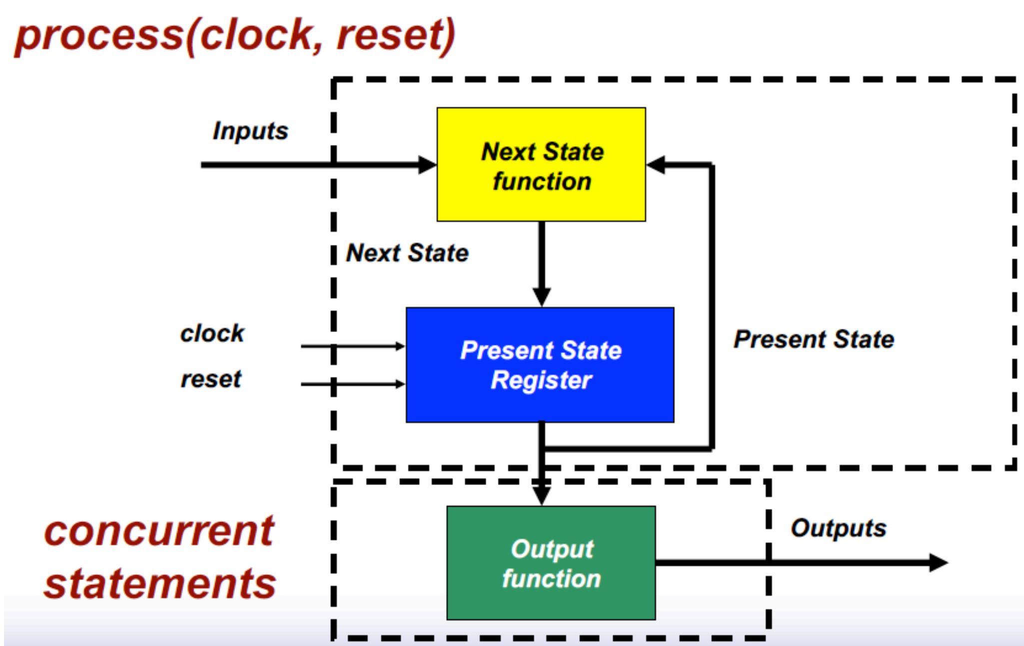
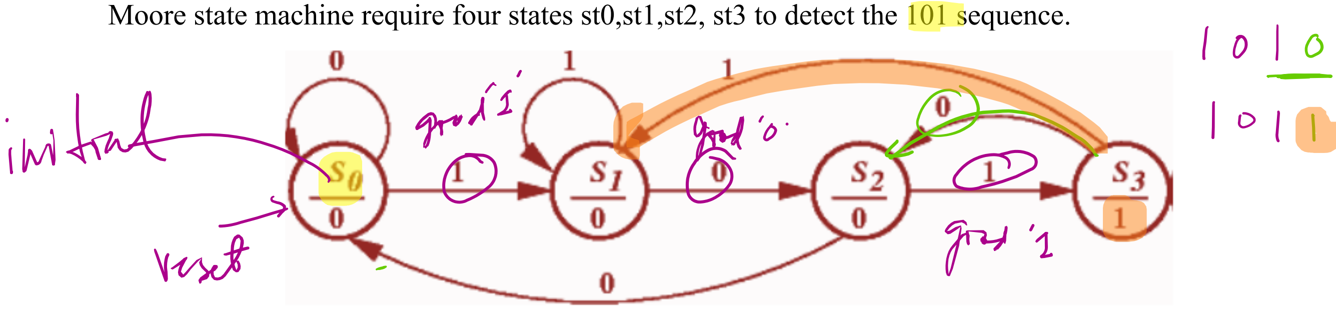
A. Moore machine 101

Let's construct the sequence detector for the sequence 101 using Moore state machine. The Output of the State machine depends only on present state. The output of state machine are only updated at the clock edge. *The FSM uses a synchronous reset and clock has a higher priority than reset.*



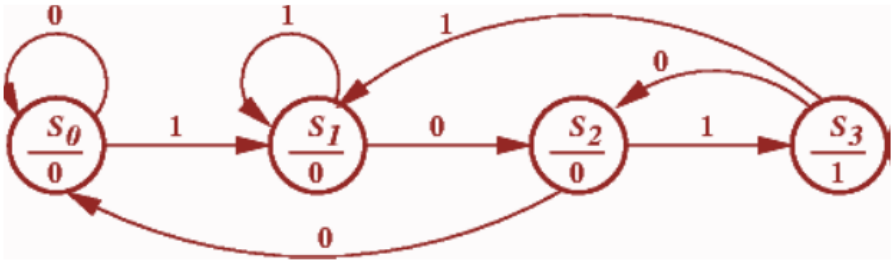
Moore state machine require four states st0,st1,st2, st3 to detect the 101 sequence.



VHDL code for Sequence detector (101) using Moore state machine

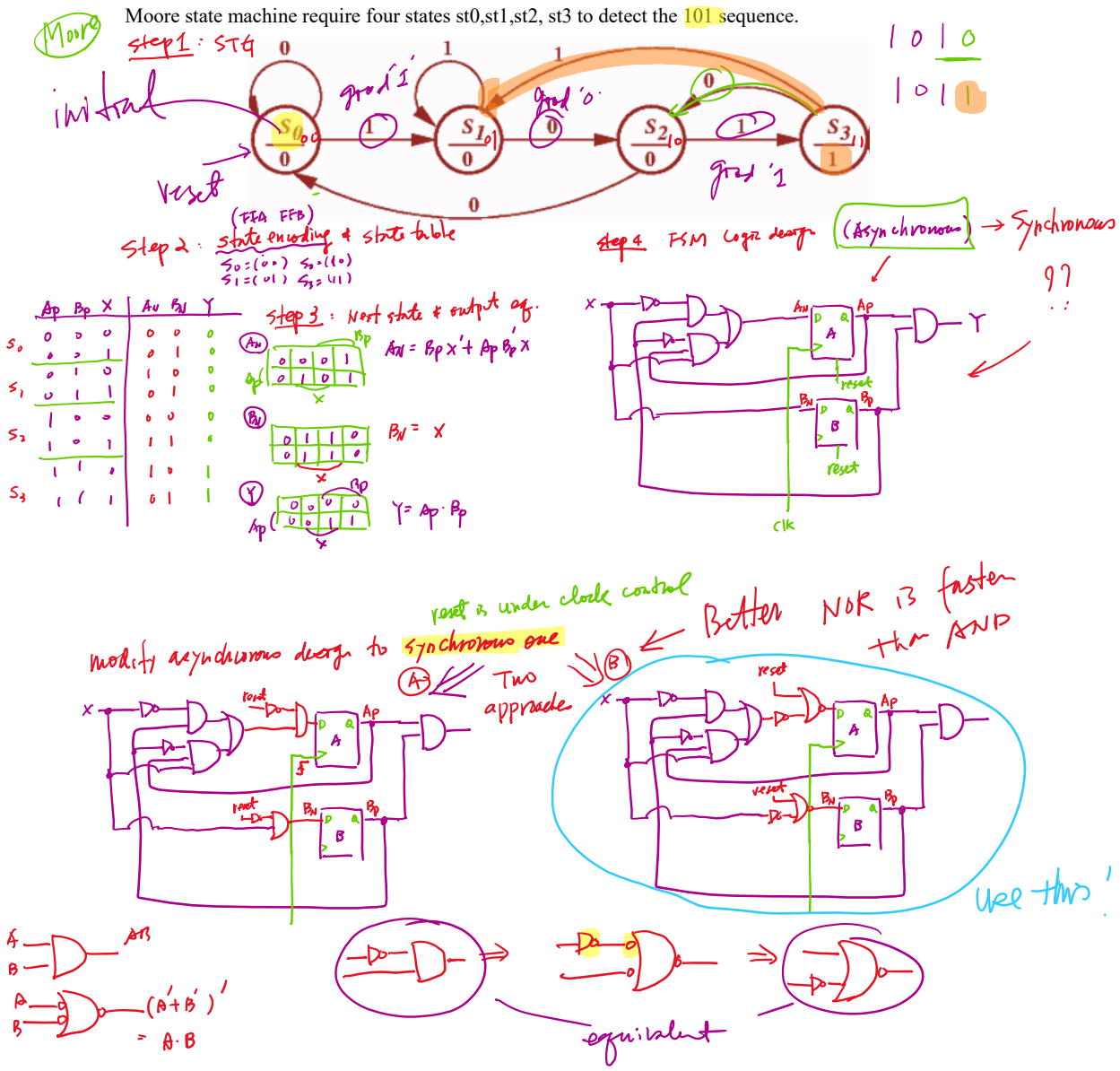
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity moore is
Port (   clk : in STD_LOGIC;
         din : in STD_LOGIC;
         rst : in STD_LOGIC;
```

```
        dout : out STD_LOGIC);
end moore;
```



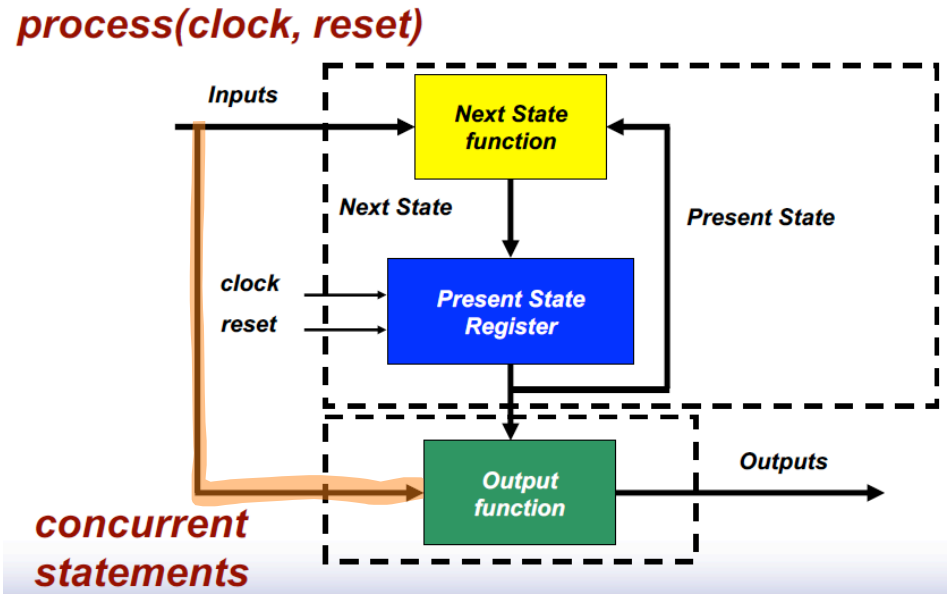

```
architecture Behavioral of moore is
type state is (st0, st1, st2, st3);
signal present_state, next_state : state;
begin

next_state_decoder : process(present_state, din)
begin
        case (present_state) is
                when st0 =>
                        if (din = '1') then
                                next_state <= st1;
                        else
                                next_state <= st0;
                        end if;
                when st1 =>
                        if (din = '1') then
                                next_state <= st1;
                        else
                                next_state <= st2;
                        end if;
                when st2 =>
                        if (din = '1') then
                                next_state <= st3;
                        else
                                next_state <= st0;
                        end if;
                when st3 =>
                        if (din = '1') then
                                next_state <= st1;
                        else
                                next_state <= st2;
                        end if;
                when others =>
                        next_state <= st0;
        end case;
```

```
end process;

synchronous_process: process(clk, rst)
begin
-- Synchronous reset; clock has a higher priority than reset.
        if (clk = '1' AND clk'event) then
                if (rst = '1') then
                        present_state <= st0;
                else
                        present_state <= next_state;
                end if;
        end if;
end process;

        dout <= '1' WHEN present_state = st3 ELSE '0';
end Behavioral;
```

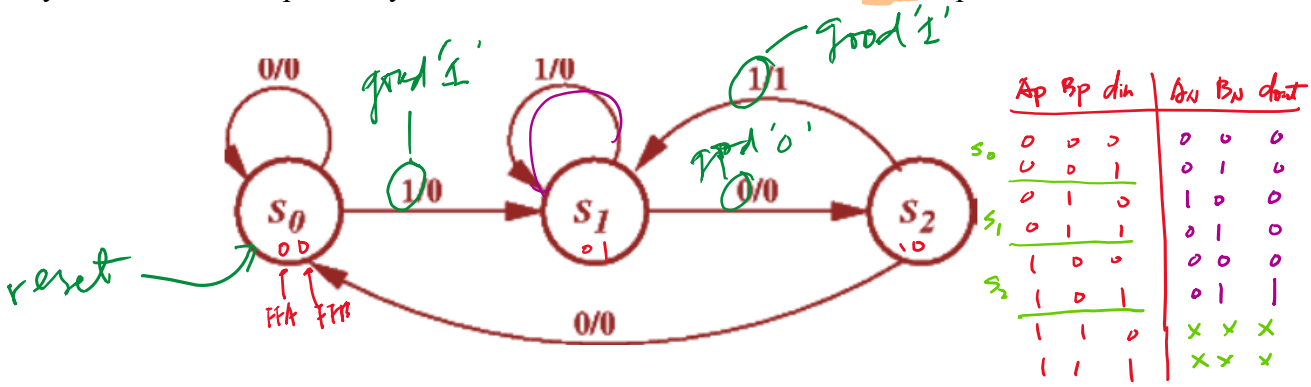

Moore state machine require four states st0,st1,st2, st3 to detect the 101 sequence.
Moore
step1 : STG
initial
reset
grad '1'
grad '0'
1 0 1 0
1 0 1 1
step 2 : state encoding & state table
(FFA FFB)
step 3 : Next state & output eq.
step 4 FSM logic design
(Asynchronous) → Synchronous
reset
clk
modify asynchronous design to synchronous one
reset is under clock control
Better NOR is faster than AND
Two approach
use this!
equivalent

B. Mealy machine 101

Let's construct the sequence detector for the sequence 101 using Mealy state machine. The output of the state machine depends on both present state and current input. When the input changes, the output of the state machine updated without waiting for change in clock input.



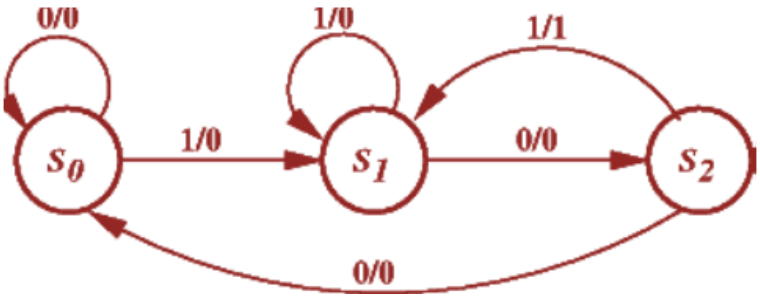
Mealy state machine require only three states st0,st1,st2 to detect the 101 sequence.



VHDL code for Sequence detector (101) using Mealy state machine

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mealy is
Port (   clk : in STD_LOGIC;
         din : in STD_LOGIC;
         rst : in STD_LOGIC;
         dout : out STD_LOGIC);
end mealy;
```

```
architecture Behavioral of mealy is
type state is (st0, st1, st2);
signal present_state, next_state : state;
begin
```




```
begin
        case (present_state) is
                when st0 =>
                        if (din = '1') then
                                next_state <= st1;
                                d_out <= '0';
                        else
                                next_state <= st0;
                                d_out <= '0';
                        end if;
                when st1 =>
                        if (din = '1') then
                                next_state <= st1;
                                d_out <= '0';
                        else
                                next_state <= st2;
                                d_out <= '0';
                        end if;
                when st2 =>
                        if (din = '1') then
                                next_state <= st1;
                                d_out <= '1';
                        else
                                next_state <= st0;
                                d_out <= '0';
                        end if;
                when others =>
                        next_state <= st0;
                        d_out <= '0';
        end case;
end process;

synchronous_process: process(clk)
begin
```

```
        if (clk = ‘1’ AND clk’event) then
                if (rst = '1') then
                        present_state <= st0;
                else
                        present_state <= next_state;
                end if;
        end if;
end process;

end Behavioral;
```