CPEG 422/622
EMBEDDED SYSTEMS DESIGN

Chengmo Yang

chengmo@udel.edu

Evans 201C

LECTURE 7
SEQUENTIAL LOGIC

REVIEW

- Last Lecture:

Vivado reports

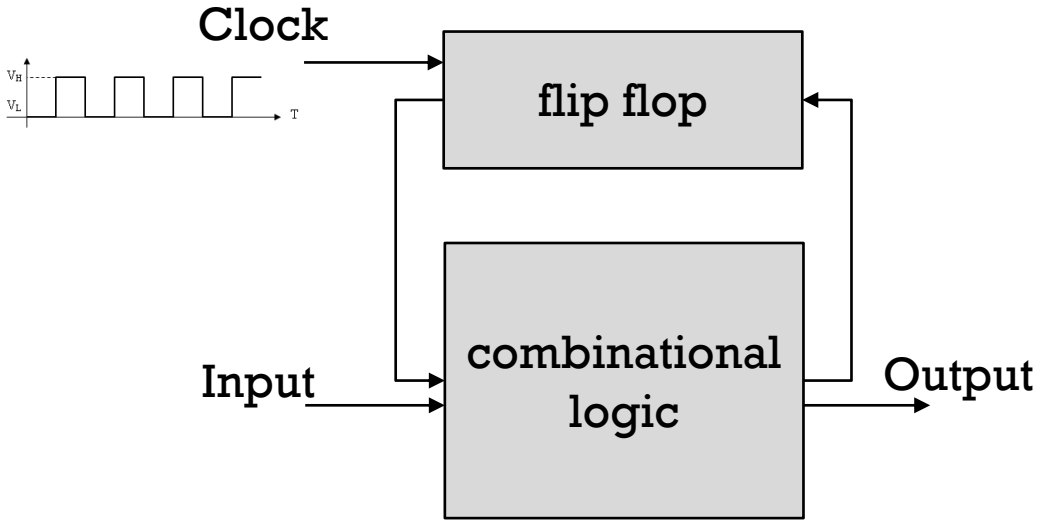
- This lecture:

Sequential logic

- Shift register
- Finite state machine (FSM)

SEQUENTIAL LOGIC

- Sequential logic:
  1. Use memory components such flip-flop to store states.
  2. Output depends on both inputs and the state.
  3. Synchronous sequential logic is always driven by a **clock**.

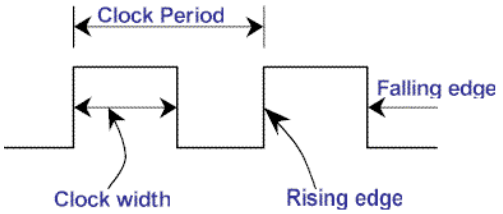


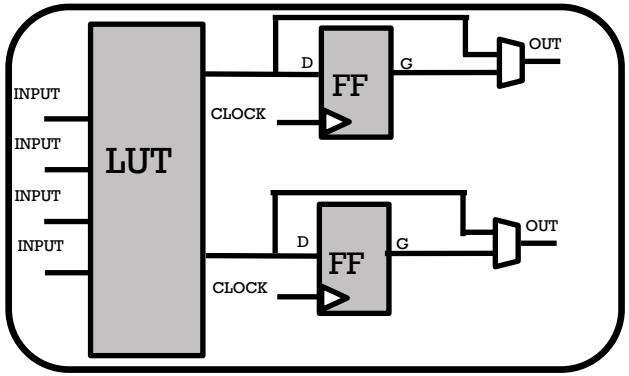
SEQUENTIAL LOGIC IN FPGA

- In FPGA:

Lookup table (LUT) is for combinational logic implementation.

Flip-flops (FF) are for sequential logic implementation. An FF captures the input at the rising edge of the clock.





D flip-flop truth table

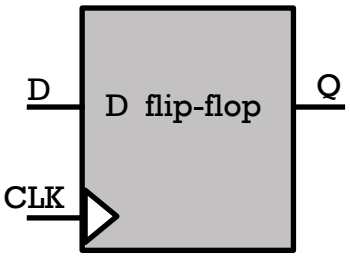
clk	D	Q	Q(next)
X	X	0	**0**
X	X	1	**1**
↑	0	X	**0**
↑	1	X	**1**

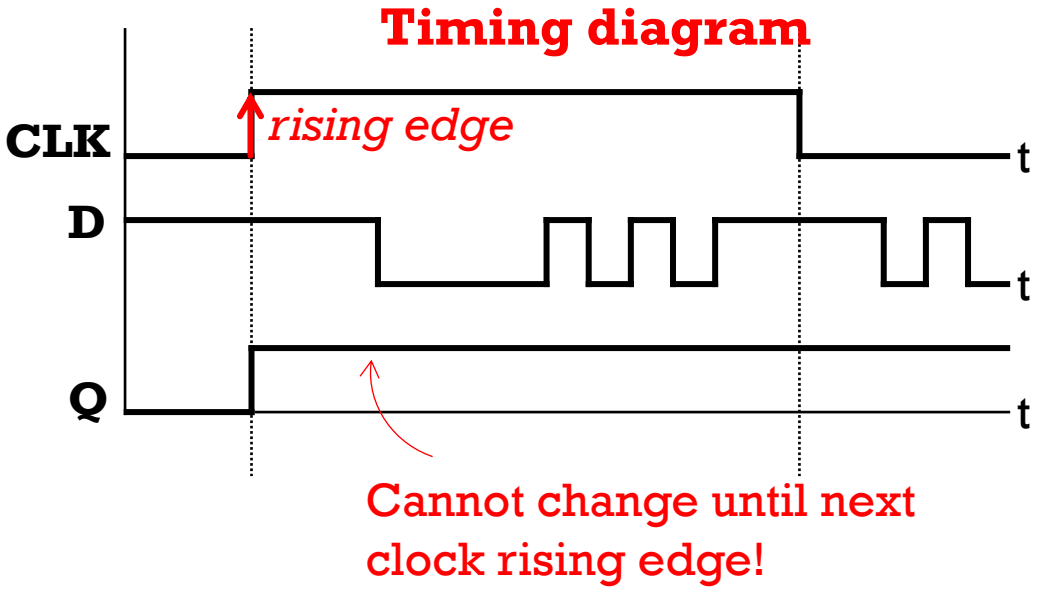
Otherwise, Q holds old value.

At clock rising edge, Q<=D.

Tend to be stable

D FLIP-FLOPS





NOTE: ① Q<= D just at clock positive edge!
② Q does NOT change for remainder of clock cycle!

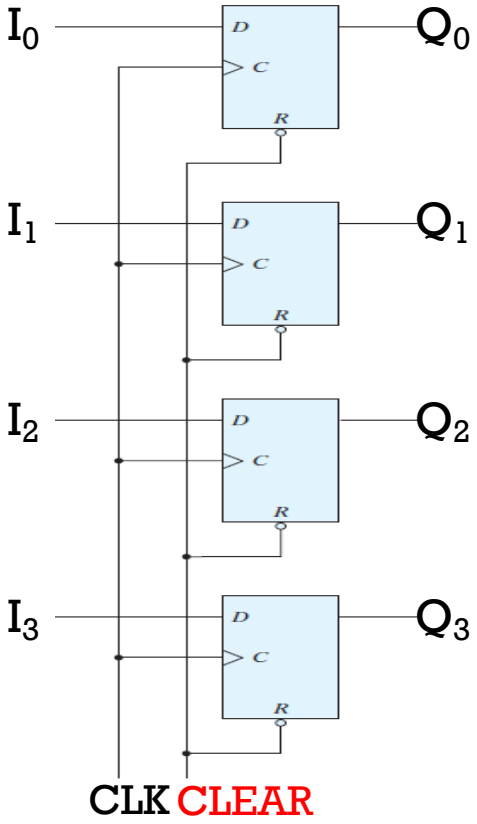
D flip-flop can only store single bit.
Q:How about multi bits (vector) storage?

A:Use register.

REGISTER

Register consists of a group of flip-flops with combinational logic.

Simplest case → 4-bit register



Synchronous Reset

When at rising edge of CLK,

CLEAR = 1 → $Q_n = 0$

CLEAR = 0 → $Q_n = I_n$

Problem?

If I_n changes, then Q_n changes

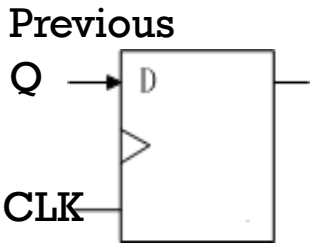
Cannot store data!

REGISTER WITH PARALLEL LOAD

Parallel load

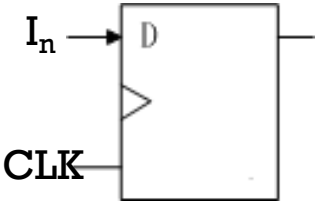
LOAD =0→Q_n does not change

→D = Previous Q

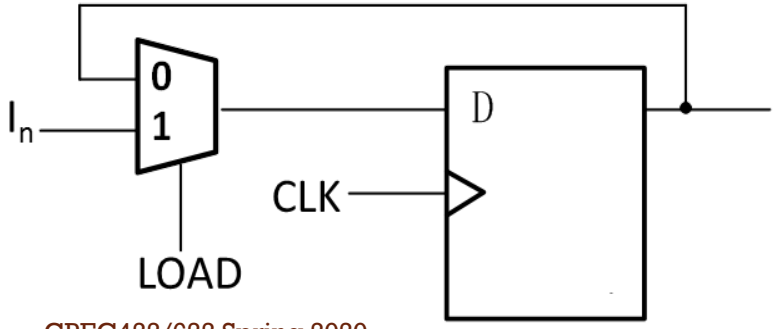


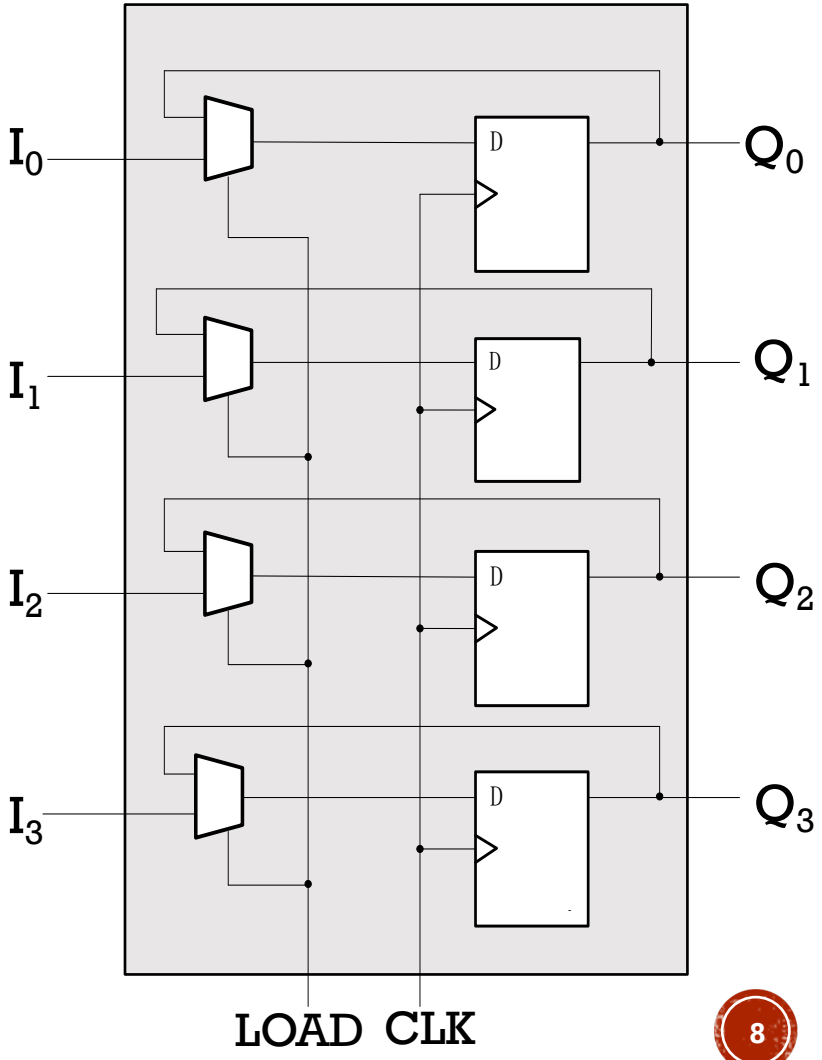
LOAD=1→Q_n= I_n

→D = I_n



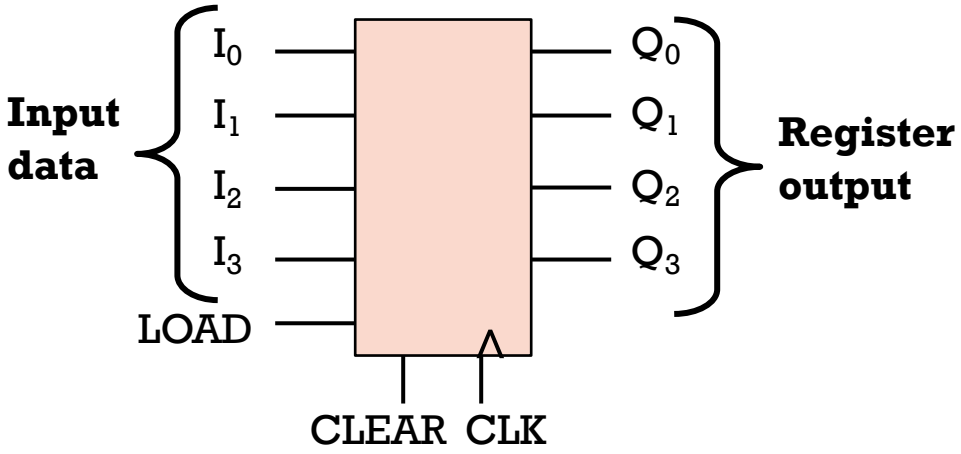
Just use a **2:1 MUX**!





PARALLEL LOAD REGISTER IN VHDL

As a block component:



Library

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

Entity

```
entity parallel_load_reg is
-- Port ( );
port(
      Clock : in std_logic;
      Load: in std_logic;
      Input: in std_logic_vector(3 downto 0);
      Q: out std_logic_vector(3 downto 0)
     );
end parallel_load_reg;
```

Architecture

```
architecture parallel_load_reg of parallel_load_reg is

begin
P_load:process(Clock)
       begin
       if(rising_edge(Clock)) then
         if(Load='1') then
           Q<=Input;
         end if;
       end if;
       end process;

end parallel_load_reg;
```

ANOTHER EXAMPLE 3-BIT COUNTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity counter is
  port(
       Enable: in std_logic;
       Clock: in std_logic;
       Reset: in std_logic;
       Output: out std_logic_vector(2 downto 0)
       );
end counter;

architecture counter of counter is

signal temp: std_logic_vector(2 downto 0);

begin
   process(Clock,Reset)
   begin
      if Reset='1' then
         temp <= "000";
      elsif(rising_edge(Clock)) then
         if Enable='1' then
           if temp="111" then
              temp<="000";
           else
               temp <= temp + 1;
           end if;
         end if;
      end if;
   end process;

   Output <= temp;

end counter;
```

If-then-else statement

IF-THEN-ELSE STATEMENTS

Syntax:

```
if <condition> then
  statements
  ...

[
elsif <condition> then
  statements
  ...

else
  statements
  ...

]
endif;
```

Examples:

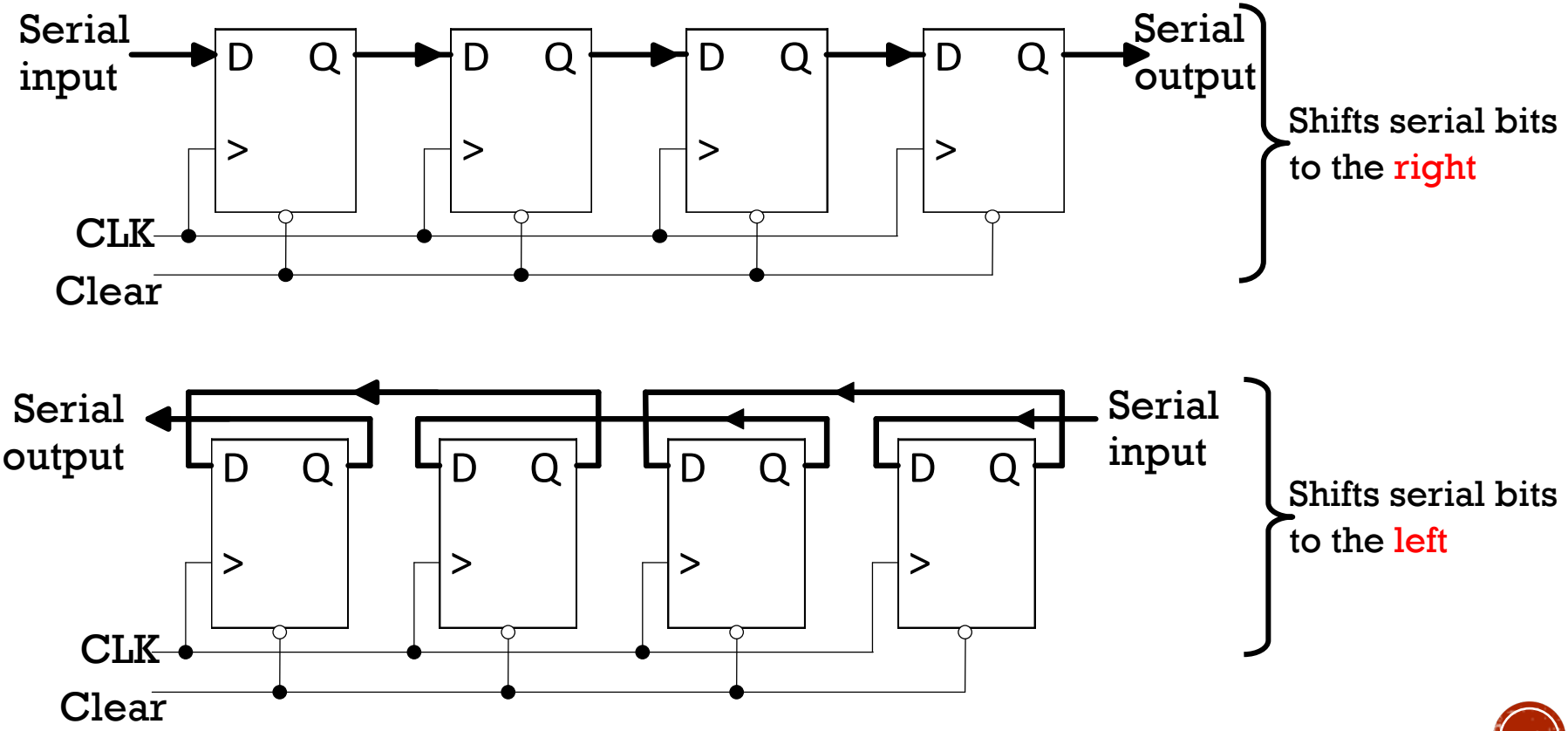
```
if boolean_v then
  output_1 <= '1';

end if;
---------------------------
if condition_v_1 = '1' then
  out_vector <= "001"

elsif condition_v_2 = '1'
then
  out_vector <= "110"

...

else
  out_vector <= "000"

end if;
```

- Can be nested.
- Performed by checking each condition in the presented order until a "true" is found.

SHIFT REGISTER

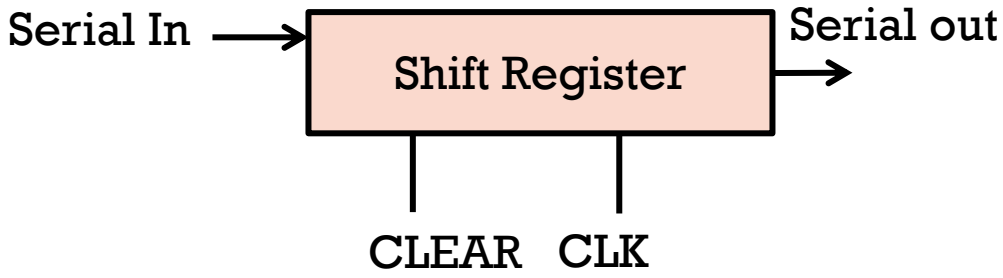
Shift data from one flip-flop to its neighbor (in a selected direction)

Consider:



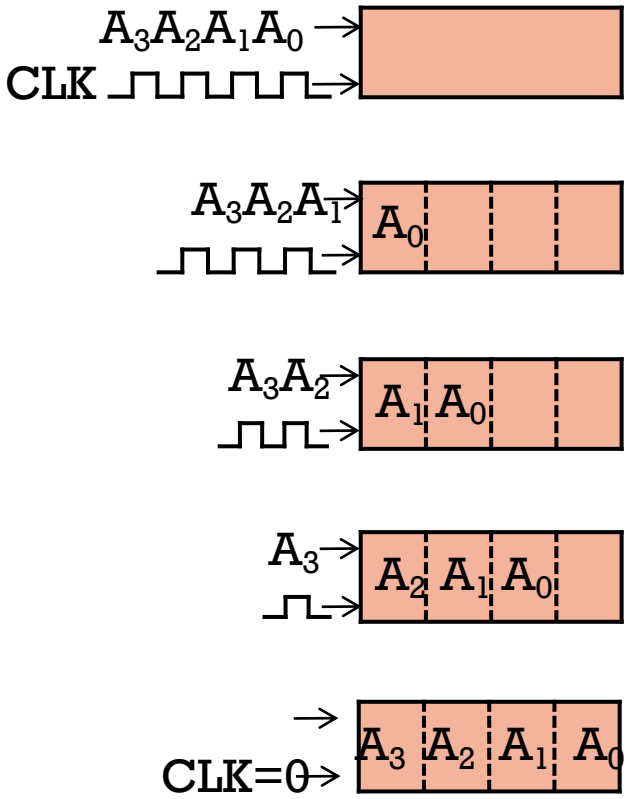
SHIFT REGISTER

As a block component:

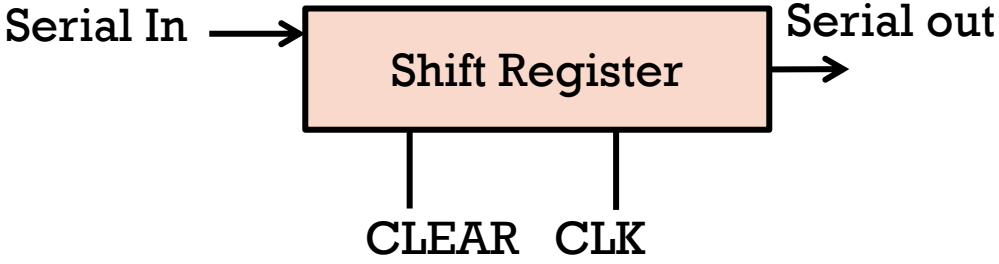


Example:

Read in 4 bits serially and store



SHIFT REGISTER



VHDL implementation of right-shift register.

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

entity shift_register is
-- Port ( );
 port(
      Clock : in std_logic;
      Clear: in std_logic;
      Serial_in: in std_logic;
      Serial_out: out std_logic
     );
end shift_register;
```

```
architecture shift_register of shift_register is
 signal q:std_logic_vector(3 downto 0);
begin

 shift:process(Clock)
       begin
       if(rising_edge(Clock)) then
         if(Clear='1') then
           q<=(others=>'0');
         else
           q(3)<=Serial_in;
           q(2 downto 0)<=q(3 downto 1);
           Serial_out<=q(0);
         end if;
       end if;
       end process;

end shift_register;
```

Sequential signal assignment

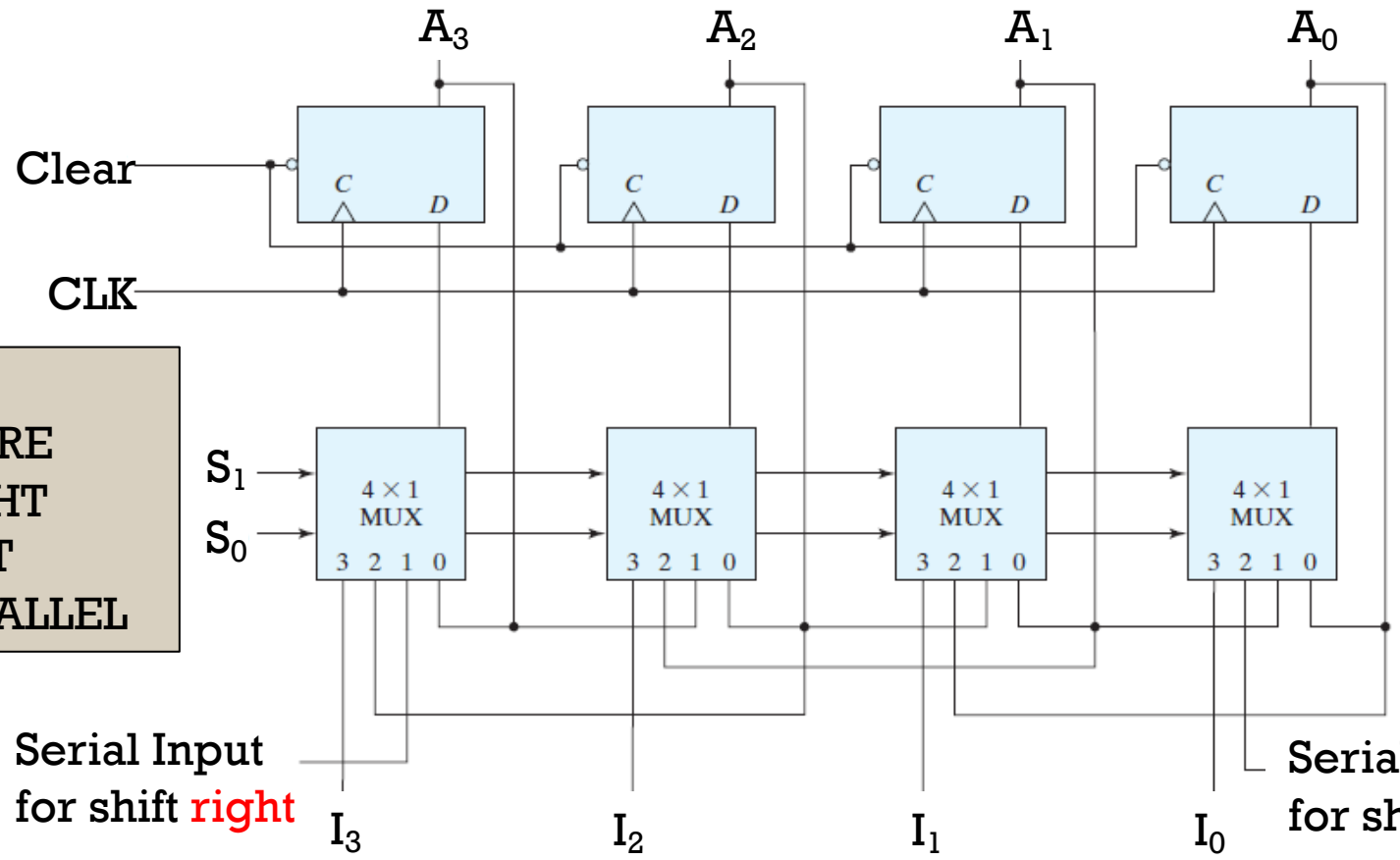
SIGNAL ASSIGNMENT INSIDE PROCESS

- Signal assignment is typically concurrent.
- When used inside a process, the assignments become sequential, but happen with a delay.
- Why? Because signals are declared outside a process. Thus they are updated in the scope where they are declared, when the sequential code reaches its end or encounters a 'wait' or other event that triggers the update.

```
architecture shift_register of shift_register is
 signal q:std_logic_vector(3 downto 0);
begin

 shift:process(Clock)
       begin
       if(rising_edge(Clock)) then
         if(Clear='1') then
           q<=(others=>'0');
         else
           q(3)<=Serial_in;
           q(2 downto 0)<=q(3 downto 1);
           Serial_out<=q(0);
         end if;
       end if;
       end process;

end shift_register;
```

UNIVERSAL SHIFT REGISTER

- Combines all the features (store, shift right, shift left, parallel load)

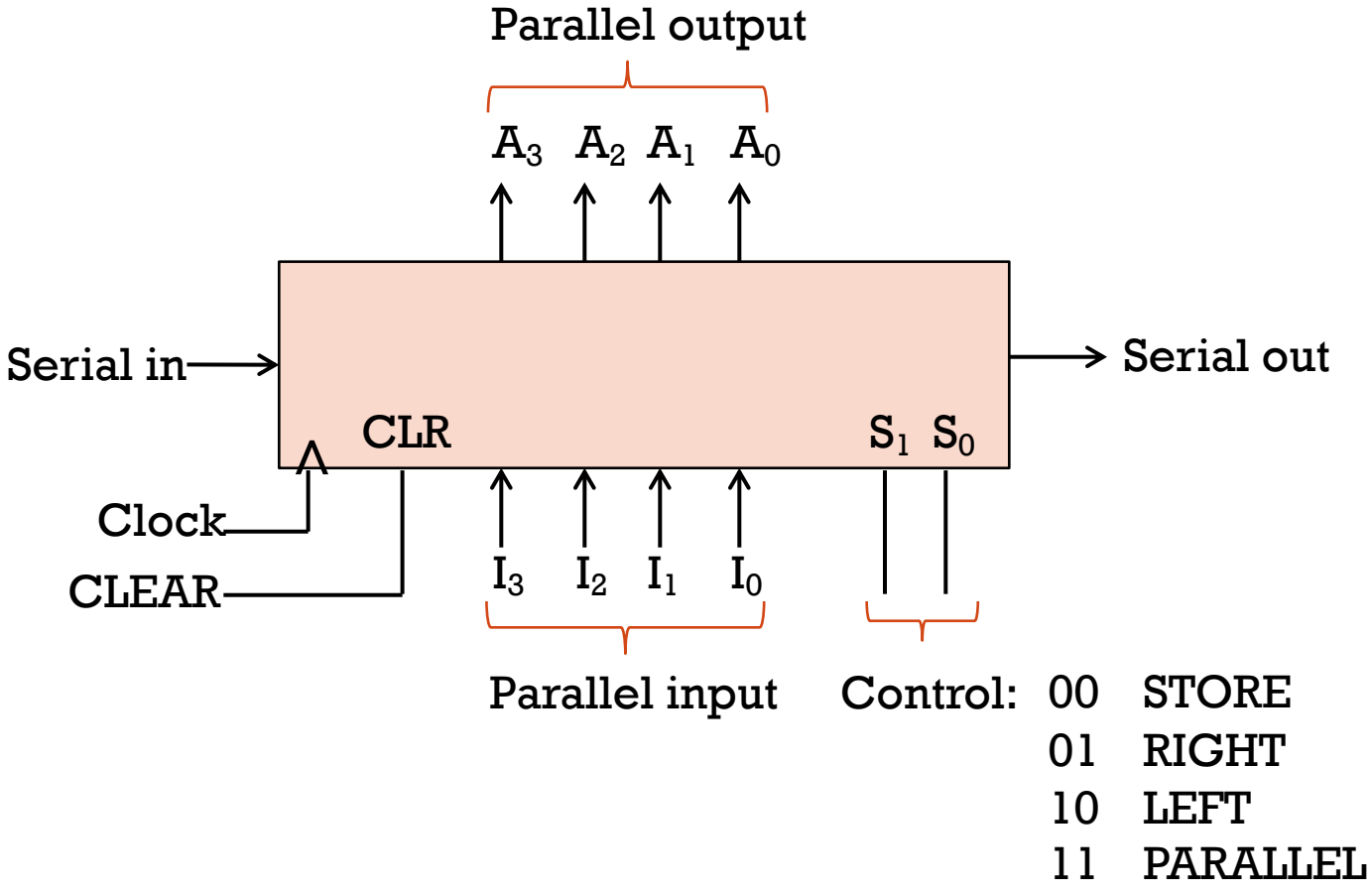


MUX:

00	STORE
01	RIGHT
10	LEFT
11	PARALLEL

UNIVERSAL SHIFT REGISTER

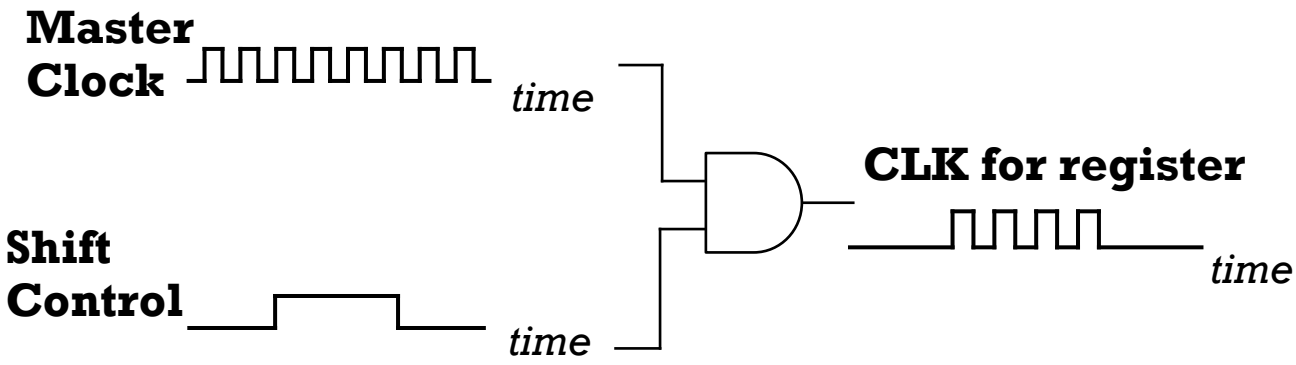
As a block component:



SHIFT CONTROL

Q: How to control when the register shifts?

➡ Control the **clock**



SEQUENTIAL LOGIC

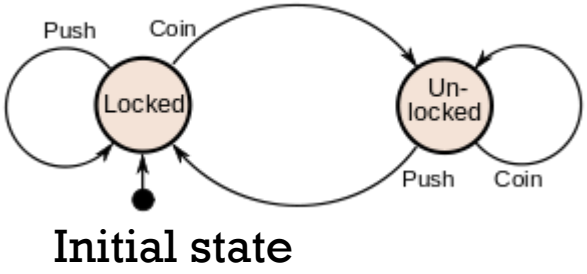
- Register
- **Finite state machine (FSM)**

FINITE STATE MACHINE

A finite state machine (FSM) is a mathematical model of computation. It can represent a serial of behavior of sequential logic.

- FSMs are widely used in automatic control system.

Example：

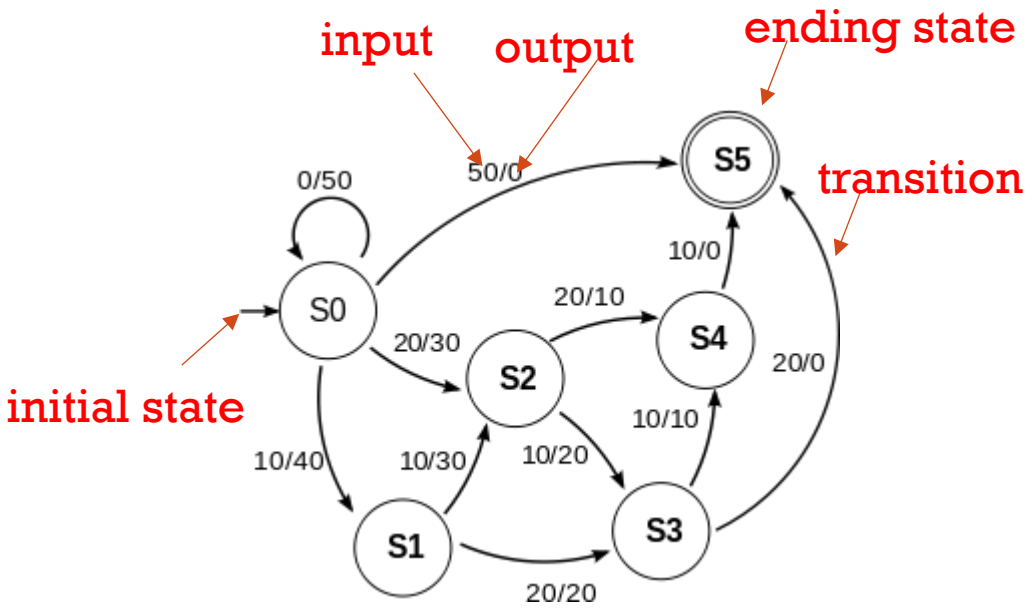


State diagram for a turnstile (wiki)

FINITE STATE MACHINE

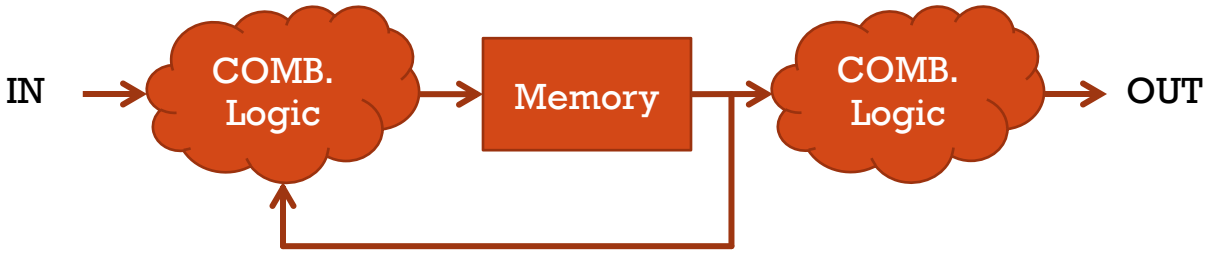
FSM diagram consists of:

1. States
2. Transitions
3. Inputs
4. Outputs



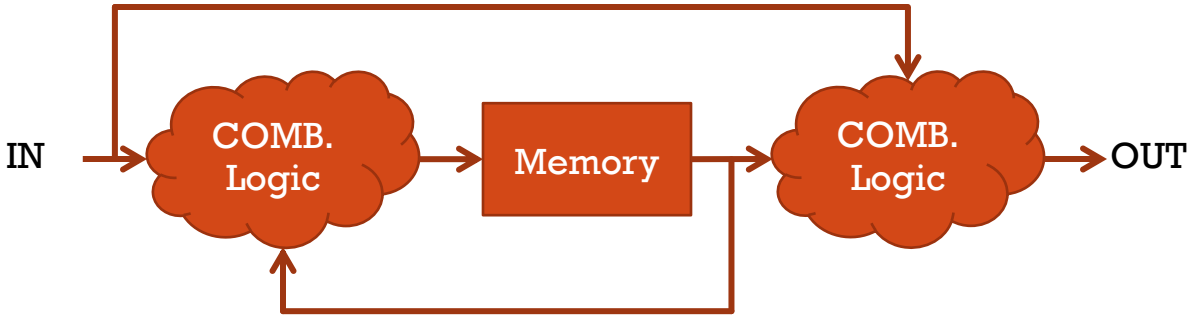
FINITE STATE MACHINE

FSM can be divided into **Moore** and **Mealy** Machines.



MOORE MACHINE

Output depends on present state only



MEALY MACHINE

Output depends on present state and inputs

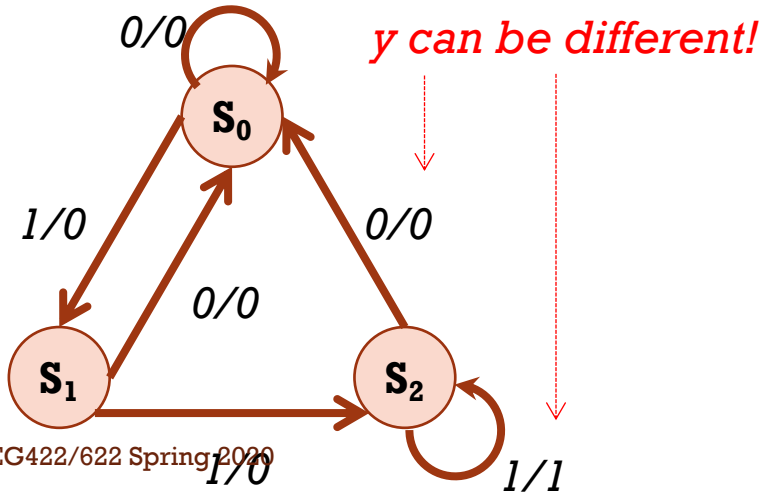
FSM EXAMPLE

Sequence detector

- Reads incoming bits and produces OUTPUT = 1 when it detects 3 or more 1's in a row

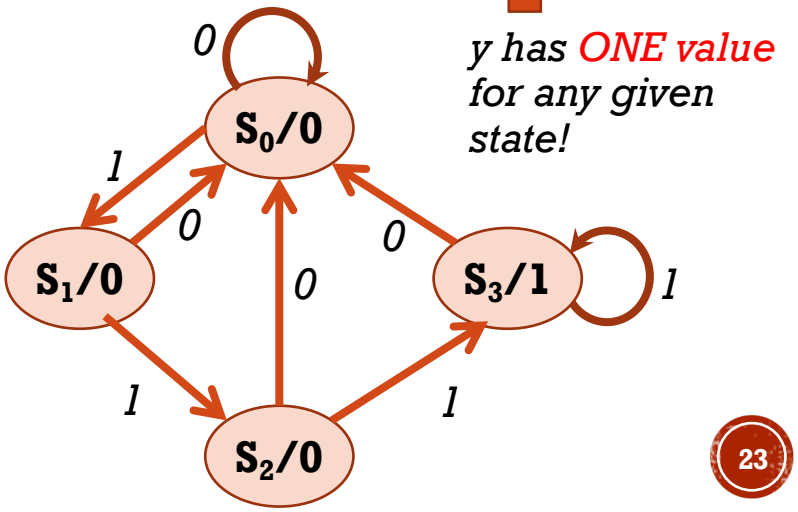
MEALY MACHINE

$S_0 \rightarrow$ Detected 0
$S_1 \rightarrow$ Detected 1
$S_2 \rightarrow$ Detected 11



MOORE MACHINE

$S_0 \rightarrow$ Detected 0 (y = 0)
$S_1 \rightarrow$ Detected 1 (y = 0)
$S_2 \rightarrow$ Detected 11 (y = 0)
$S_3 \rightarrow$ Detected 111 (y = 1)



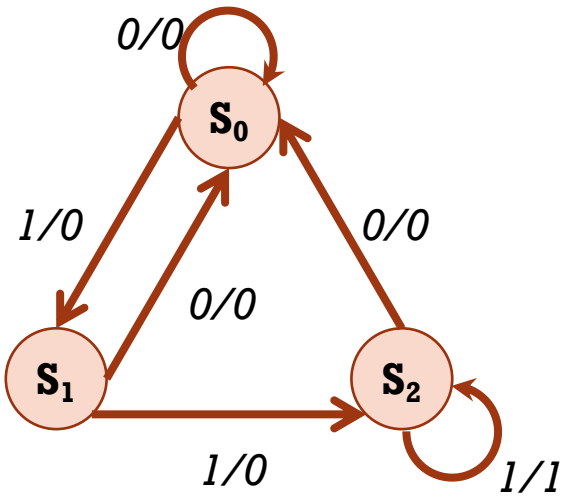
FSM CIRCUIT DESIGN IN VHDL

STEP 1 **State assignment (binary coding)**

Flip-flops: A B

Need two D flip-flops!

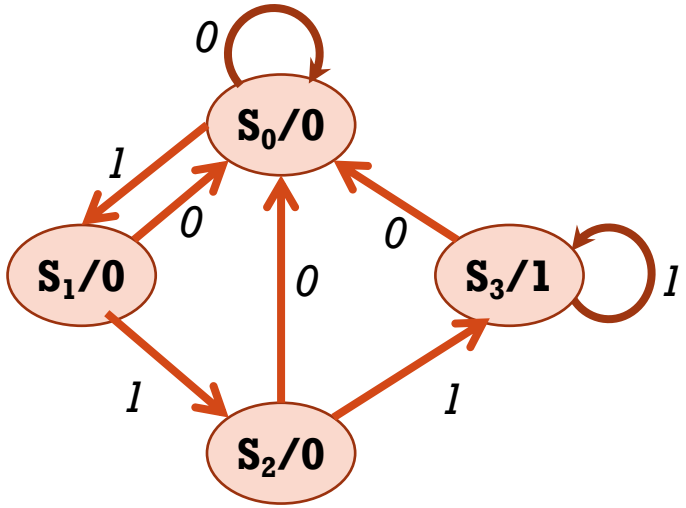
S_0:00
S_1:01
S_2:10



Flip-flops: A B

Need two D flip-flops!

S_0:00
S_1:01
S_2:10
S_3:11



FSM CIRCUIT DESIGN IN VHDL

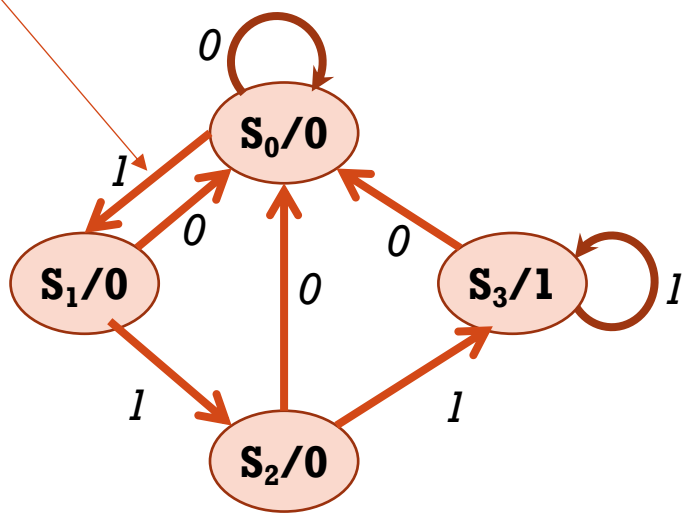
STEP 2

Describe next state and output logic in VHDL

For example, describe S_0 (00) to S_1 (01)

```
if current_state= "00" then
  if input=1  then
    next_state<= "01";
  end if;
end if;
```

Case statements are more frequently used.



CASE STATEMENT

Case statements are quite useful in condition translations (e.g. keyboard scan code interpretations).

Syntax:

```
case <expression> is

 when <choice(s)> =>

  <expression>;
  ...

 when ...

 [when others => ... ]

end case;
```

Example:

```
case scancode is

 when x"14" =>
  integer_signal <= 1;

 when x"18" =>
  integer_signal <= 2;

 when x"19" | x"20" |
 x"21" =>
  integer_signal <= 3;

 when others =>
  integer_signal <= 0;

end case;
```

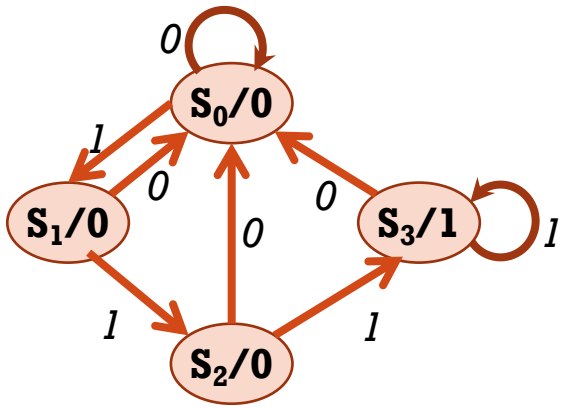
Hexadecimal bits

VHDL EXAMPLE-MOORE FSM

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

entity sequence_detector is
 Port(
      clock:in std_logic;
      reset:in std_logic;
      input:in std_logic;
      output:out std_logic
      );
end sequence_detector;

architecture Behavioral of sequence_detector is
  signal state,next_state: std_logic_vector(1 downto 0):="00";

begin
 sync:process(clock)
 begin
  if(rising_edge(clock)) then
     if(reset='1') then
      state<="00";
     else
      state<=next_state;
     end if;
   end if;
   end process;
```

```
state_logic:process(input,state)
begin
   case state is
     when "00"=>
         if(input='0') then  next_state<="00";
         else  next_state<="01";
         end if;
     when "01"=>
         if(input='0') then  next_state<="00";
         else  next_state<="10";
         end if;
     when "10"=>
         if(input='0') then  next_state<="00";
         else  next_state<="11";
         end if;
     when "11"=>
         if(input='0') then  next_state<="00";
         else  next_state<="11";
         end if;
   end case;
end process;
```

```
output:process(state)
  begin
  case state is
          when "00"=>
             output<='0';
          when "01"=>
             output<='0';
          when "10"=>
             output<='0';
          when "11"=>
             output<='1';
        end case;
end process;
```



S_0 :00
S_1 :01
S_2 :10
S_3 :11

VHDL EXAMPLE-MOORE FSM

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

entity sequence_detector is
 Port(
      clock:in std_logic;
      reset:in std_logic;
      input:in std_logic;
      output:out std_logic
      );
end sequence_detector;

architecture Behavioral of sequence_detector is
  signal state,next_state: std_logic_vector(1 downto 0):="00";
```

```vhdl
begin
 sync:process(clock)
 begin
  if(rising_edge(clock)) then
     if(reset='1') then
      state<="00";
     else
      state<=next_state;
     end if;
   end if;
   end process;
```

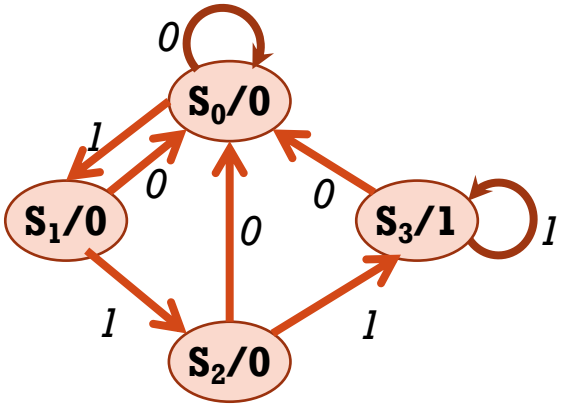
Reset & State change

```vhdl
state_logic:process(input,state)
begin
   case state is
     when "00"=>
         if(input='0') then  next_state<="00";
         else  next_state<="01";
         end if;
     when "01"=>
         if(input='0') then  next_state<="00";
         else  next_state<="10";
         end if;
     when "10"=>
         if(input='0') then  next_state<="00";
         else  next_state<="11";
         end if;
     when "11"=>
         if(input='0') then  next_state<="00";
         else  next_state<="11";
         end if;
   end case;
end process;
```

Next state logic

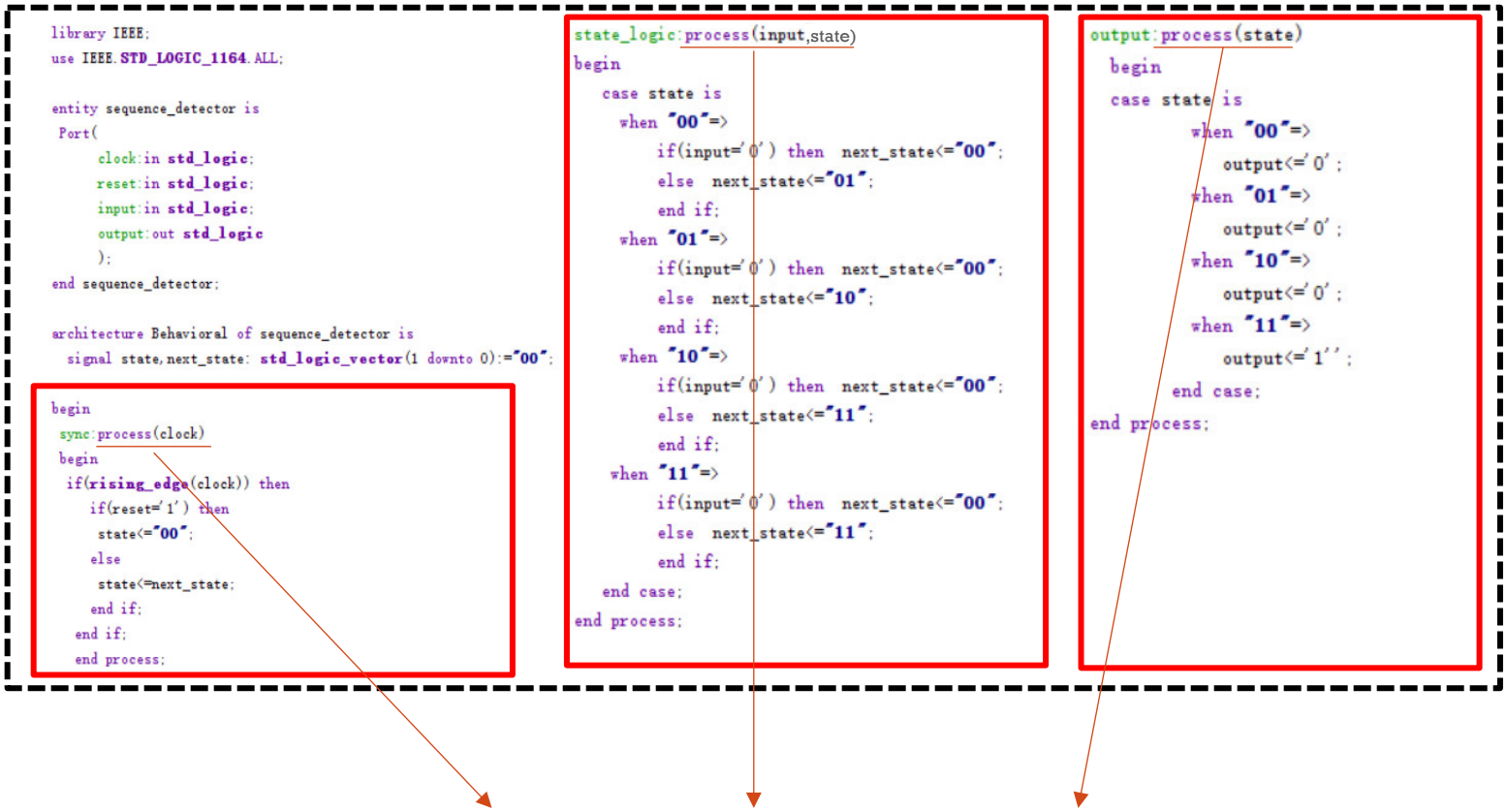
```vhdl
output:process(state)
  begin
  case state is
          when "00"=>
             output<='0';
          when "01"=>
             output<='0';
          when "10"=>
             output<='0';
          when "11"=>
             output<='1';
        end case;
end process;
```

Output logic



S_0 :00
S_1 :01
S_2 :10
S_3 :11

VHDL EXAMPLE-MOORE FSM



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

entity sequence_detector is
 Port(
      clock:in std_logic;
      reset:in std_logic;
      input:in std_logic;
      output:out std_logic
      );
end sequence_detector;

architecture Behavioral of sequence_detector is
  signal state,next_state: std_logic_vector(1 downto 0):="00";

begin
 sync:process(clock)
 begin
  if(rising_edge(clock)) then
     if(reset='1') then
      state<="00";
     else
      state<=next_state;
     end if;
   end if;
   end process;

state_logic:process(input,state)
begin
   case state is
     when "00"=>
         if(input='0') then  next_state<="00";
         else  next_state<="01";
         end if;
     when "01"=>
         if(input='0') then  next_state<="00";
         else  next_state<="10";
         end if;
     when "10"=>
         if(input='0') then  next_state<="00";
         else  next_state<="11";
         end if;
     when "11"=>
         if(input='0') then  next_state<="00";
         else  next_state<="11";
         end if;
   end case;
end process;

output:process(state)
  begin
  case state is
          when "00"=>
              output<='0';
          when "01"=>
              output<='0';
          when "10"=>
              output<='0';
          when "11"=>
              output<='1';
        end case;
end process;
```

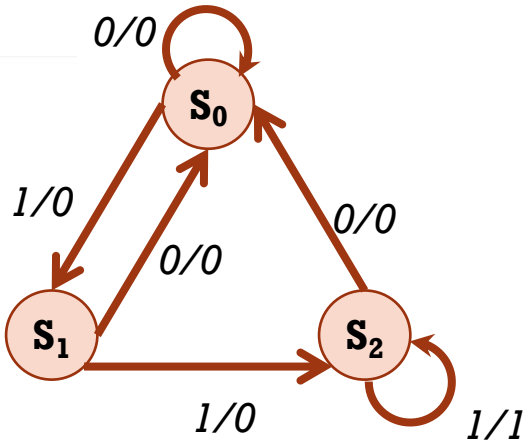
Pay attention to the sensitive signals inside the process!

VHDL EXAMPLE-MEALY FSM

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

entity sequence_detector is
 Port(
      clock:in std_logic;
      reset:in std_logic;
      input:in std_logic;
      output:out std_logic
      );
end sequence_detector;

architecture Behavioral of sequence_detector is
  signal state,next_state: std_logic_vector(1 downto 0):="00";

begin
 sync:process(clock)
 begin
  if(rising_edge(clock)) then
     if(reset='1') then
      state<="00";
     else
      state<=next_state;
     end if;
   end if;
   end process;
```

```
   state_logic:process(state,input)
   begin
      case state is
        when "00"=>
            if(input='0') then
            next_state<="00";output<='0';
            else
            next_state<="01";output<='0';
            end if;
        when "01"=>
            if(input='0') then
            next_state<="00";output<='0';
            else
            next_state<="10";output<='0';
            end if;
        when "10"=>
            if(input='0') then
            next_state<="00";output<='0';
            else
            next_state<="10";output<='1';
            end if;
         when others=>
            next_state<="00";output<='0';
      end case;
   end process;

end Behavioral;
```

Output & next state



S_0 :00
S_1 :01
S_2 :10

Why need others?

If undefined state like "11" appears accidently, it resets FSM back in order.

HOMEWORK

- Implement the **universal shift register** described in this lecture. Submit your VHDL source files and testbench.

- Implement the FSM for multiplication control. FSM diagram will be given. Submit your VHDL source files and testbench.