Chapter 5
Chapter 6
Chapter 7
Chapter 8

Misr International University
جامعة مصر الدولية

Selected Topics in Electronics

ECE569/EET21E

Dr. Ashraf Eltholth

ECE
Misr International University

Spring 2017

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Outline

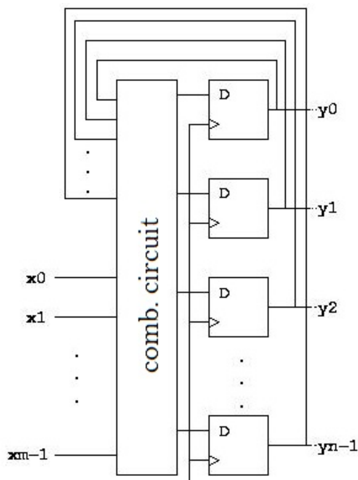
Misr International University
جامعة مصر الدولية

1. Chapter 5
2. Chapter 6
3. Chapter 7
4. Chapter 8

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

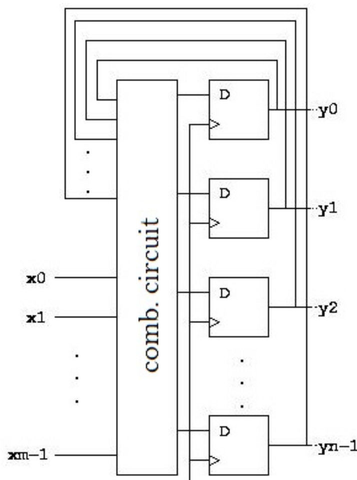
Sequential Circuits

Outlines



Sequential Circuits

Outlines

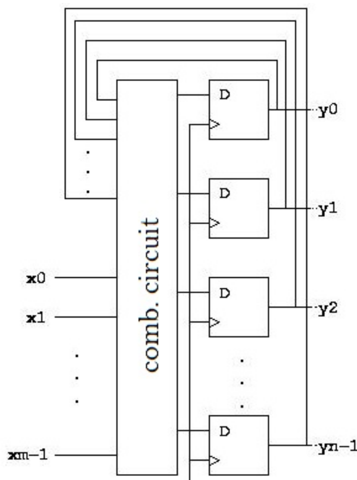


Latches & Flip Flops

- D Latches & D Flip Flops
- Preset & Clear
- JK. FF & T.FF

Sequential Circuits

Outlines



Latches & Flip Flops

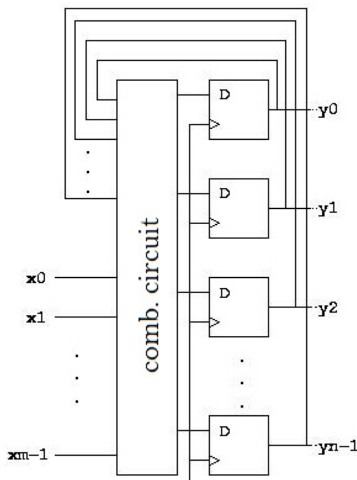
- D Latches & D Flip Flops
- Preset & Clear
- JK. FF & T.FF

Registers

- n-bit Regiters with controls
- Shift Registers
- Register Applications

Sequential Circuits

Outlines



Latches & Flip Flops

- D Latches & D Flip Flops
- Preset & Clear
- JK. FF & T.FF

Registers

- n-bit Regiters with controls
- Shift Registers
- Register Applications

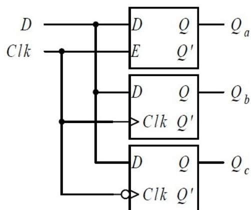
Counters

- Behavioral Counters
- Structural Counters

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

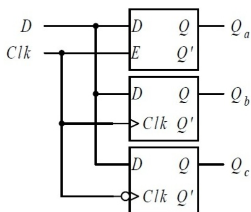
Latchs vs. Flip Flops

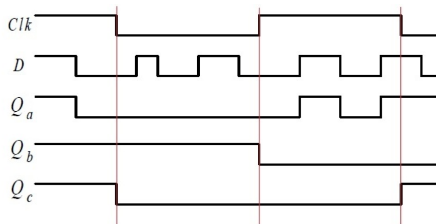
Remember



Latchs vs. Flip Flops

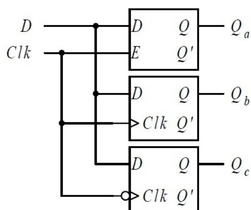
Remember

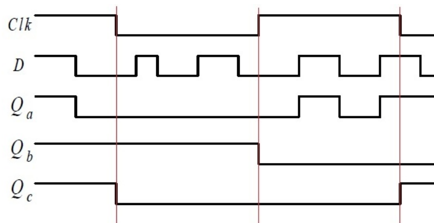




Latchs vs. Flip Flops

Remember



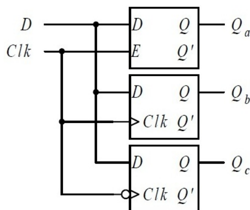


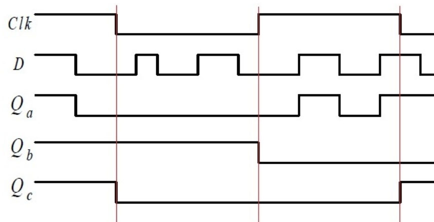

```
entity Latch is


end entity;
Architecture bhv of Latch is
begin




end Architecture;
```

```
entity DFF is


end entity;
Architecture beh of DFF is
begin




end Architecture;
```

Latchs vs. Flip Flops

Remember





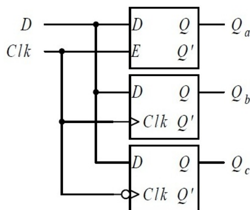

```
entity Latch is
port (D,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture bhv of Latch is
begin



end Architecture;
```

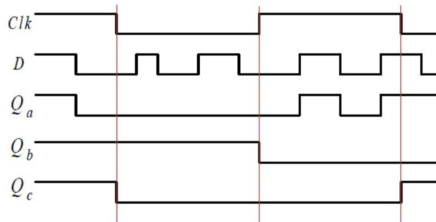
```
entity DFF is
port (D,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of DFF is
begin



end Architecture;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Latchs vs. Flip Flops

Remember



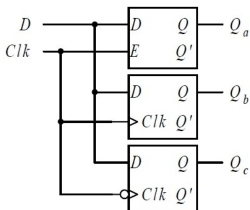


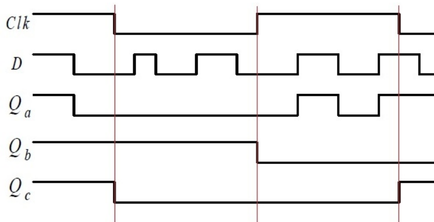

```
entity Latch is
port (D,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture bhv of Latch is
begin


    if (CLK = '1') then
      Q <= D ;
    end if;

end Architecture;
```

```
entity DFF is
port (D,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of DFF is
begin



end Architecture;
```

Latchs vs. Flip Flops

Remember





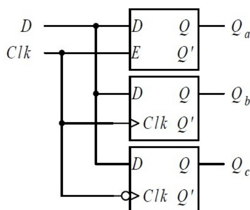

```
entity Latch is
port (D,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture bhv of Latch is
begin


    if (CLK = '1') then
      Q <= D ;
    end if;

end Architecture;
```

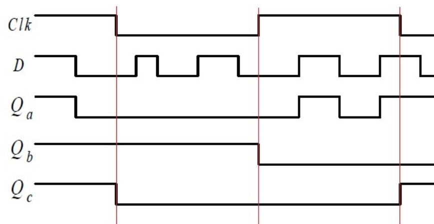
```
entity DFF is
port (D,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of DFF is
begin


    if rising_edge(CLK) then
- if falling_edge(CLK) then
      Q <= D ;
    end if;

end Architecture;
```

Misr International University
جامعة مصر الدولية

Latchs vs. Flip Flops

Remember






```
entity Latch is
port (D,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture bhv of Latch is
begin
  Process(CLK, D)
  begin
    if (CLK = '1') then
      Q <= D ;
    end if;
end Process;
end Architecture;
```

```
entity DFF is
port (D,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of DFF is
begin
  Process(CLK)
  begin
    if rising_edge(CLK) then
- if falling_edge(CLK) then
      Q <= D ;
    end if;
end Process;
end Architecture;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Synchrounous vs. Async. Controls

Preset & Clear

Synchrounous vs. Async. Controls

Preset & Clear

```
entity syn is
port (D,CLK : in std_logic;
   preset : in std_logic;
   clear : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of syn is
begin

end Architecture;
```

```
entity asyn is
port (D,CLK : in std_logic;
   preset : in std_logic;
   clear : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of asyn is
begin

end Architecture;
```

Synchrounous vs. Async. Controls

Preset & Clear

```
entity syn is
port (D,CLK : in std_logic;
   preset : in std_logic;
   clear : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of syn is
begin
  Process(CLK)
  begin


  end Process;
end Architecture;
```

```
entity asyn is
port (D,CLK : in std_logic;
   preset : in std_logic;
   clear : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of asyn is
begin
  Process(CLK,preset,clear)
  begin


  end Process;
end Architecture;
```

Synchrounous vs. Async. Controls

Preset & Clear

```
entity syn is
port (D,CLK : in std_logic;
   preset : in std_logic;
   clear : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of syn is
begin
  Process(CLK)
  begin
    if rising_edge(CLK) then
      if preset = '1' then
        Q <= '1' ;
      elsif clear = '1' then
        Q <= '0' ;
      else
        Q <= D ;
      end if;
    end if;
  end Process;
end Architecture;
```

```
entity asyn is
port (D,CLK : in std_logic;
   preset : in std_logic;
   clear : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of asyn is
begin
  Process(CLK,preset,clear)
  begin

  end Process;
end Architecture;
```

Synchrounous vs. Async. Controls

Preset & Clear

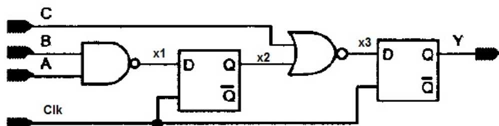
```
entity syn is
port (D,CLK : in std_logic;
   preset : in std_logic;
   clear : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of syn is
begin
  Process(CLK)
  begin
    if rising_edge(CLK) then
      if preset = '1' then
        Q <= '1' ;
      elsif clear = '1' then
        Q <= '0' ;
      else
        Q <= D ;
      end if;
    end if;
  end Process;
end Architecture;
```

```
entity asyn is
port (D,CLK : in std_logic;
   preset : in std_logic;
   clear : in std_logic;
        Q : out std_logic);
end entity;
Architecture beh of asyn is
begin
  Process(CLK,preset,clear)
  begin
    if preset = '1' then
      Q <= '1' ;

    elsif clear = '1' then
      Q <= '0' ;

    elsif rising_edge(CLK)
       then Q <= D ;
    end if;
  end Process;
end Architecture;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Latchs vs. Flip Flops

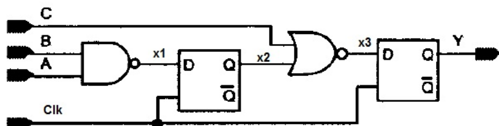
Example




```
entity example is


end entity;
Architecture behav of example is

begin









end Architecture;
```

Latchs vs. Flip Flops

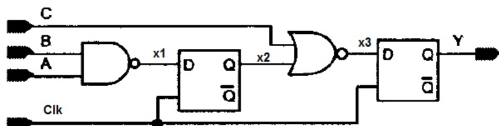
Example




```
entity example is
port (A,B,C,CLK : in std_logic;
        Y : out std_logic);
end entity;
Architecture behav of example is

begin






end Architecture;
```

Latchs vs. Flip Flops

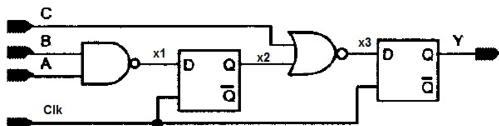
Example




```
entity example is
port (A,B,C,CLK : in std_logic;
        Y : out std_logic);
end entity;
Architecture behav of example is
signal x1, x2, x3 : std_logic :='0';
begin



end Architecture;
```

Latchs vs. Flip Flops

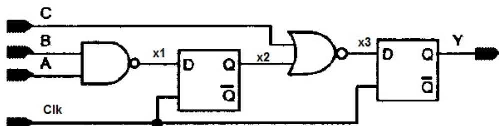
Example




```
entity example is
port (A,B,C,CLK : in std_logic;
        Y : out std_logic);
end entity;
Architecture behav of example is
signal x1, x2, x3 : std_logic :='0';
begin
  x1 <= A nand B ;
  x3 <= x2 nor C ;






end Architecture;
```

Latchs vs. Flip Flops

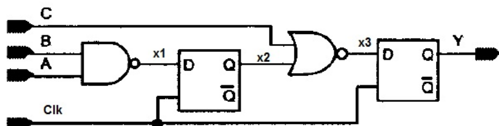
Example




```
entity example is
port (A,B,C,CLK : in std_logic;
        Y : out std_logic);
end entity;
Architecture behav of example is
signal x1, x2, x3 : std_logic :='0';
begin
  x1 <= A nand B ;
  x3 <= x2 nor C ;
  Process(CLK, x1, x3)
  begin




  end Process;
end Architecture;
```

Latchs vs. Flip Flops

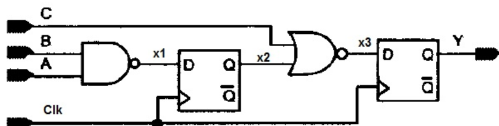
Example




```
entity example is
port (A,B,C,CLK : in std_logic;
        Y : out std_logic);
end entity;
Architecture behav of example is
signal x1, x2, x3 : std_logic :='0';
begin
  x1 <= A nand B ;
  x3 <= x2 nor C ;
  Process(CLK, x1, x3)
  begin
    if (CLK = '1') then
      x2 <= x1 ;
      Y <= x3 ;
    end if;
  end Process;
end Architecture;
```

Latchs vs. Flip Flops

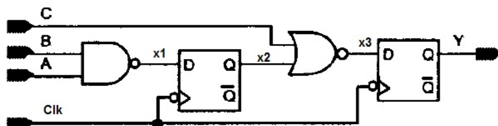
Example




```
entity example is
port (A,B,C,CLK : in std_logic;
        Y : out std_logic);
end entity;
Architecture behav of example is
signal x1, x2, x3 : std_logic :='0';
begin
  x1 <= A nand B ;
  x3 <= x2 nor C ;
  Process(CLK)
  begin
    if rising_edge(CLK) then
      x2 <= x1 ;
      Y <= x3 ;
    end if;
  end Process;
end Architecture;
```

Latchs vs. Flip Flops

Example




```
entity example is
port (A,B,C,CLK : in std_logic;
        Y : out std_logic);
end entity;
Architecture behav of example is
signal x1, x2, x3 : std_logic :='0';
begin
  x1 <= A nand B ;
  x3 <= x2 nor C ;
  Process(CLK)
  begin
    if falling_edge(CLK) then
      x2 <= x1 ;
      Y <= x3 ;
    end if;
  end Process;
end Architecture;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

JK Flip Flops

Remember

```
entity JK_FF is


end entity;
Architecture behav of JK_FF is


begin










end Architecture;
```

JK Flip Flops

Remember

```
entity JK_FF is
port (J,K,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture behav of JK_FF is


begin











end Architecture;
```

JK Flip Flops

Remember

```
entity JK_FF is
port (J,K,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture behav of JK_FF is
signal Q_temp : std_logic :='0';
signal jk : std_logic_vector(1 downto 0);
begin




end Architecture;
```

JK Flip Flops

Remember

```
entity JK_FF is
port (J,K,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture behav of JK_FF is
signal Q_temp : std_logic :='0';
signal jk : std_logic_vector(1 downto 0);
begin
  jk <= J & K ;









end Architecture;
```

Miu
Misr International University
جامعة مصر الدولية

JK Flip Flops

Remember

```
entity JK_FF is
port (J,K,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture behav of JK_FF is
signal Q_temp : std_logic :='0';
signal jk : std_logic_vector(1 downto 0);
begin
  jk <= J & K ;
  Process(CLK)
  begin





  end Process;

end Architecture;
```

Miu
Misr International University
جامعة مصر الدولية

Misr International University
جامعة مصر الدولية

JK Flip Flops

Remember

```
entity JK_FF is
port (J,K,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture behav of JK_FF is
signal Q_temp : std_logic :='0';
signal jk : std_logic_vector(1 downto 0);
begin
  jk <= J & K ;
  Process(CLK)
  begin
    if rising_edge(CLK) then









    end if;
  end Process;

end Architecture;
```

JK Flip Flops

Remember

```
entity JK_FF is
port (J,K,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture behav of JK_FF is
signal Q_temp : std_logic :='0';
signal jk : std_logic_vector(1 downto 0);
begin
  jk <= J & K ;
  Process(CLK)
  begin
    if rising_edge(CLK) then
      if (jk="10") then
        Q_temp <= '1';
      elsif (jk="01") then
        Q_temp <= '0';
      if (jk="11") then
        Q_temp <= not(Q_temp);
      end if;
    end if;
  end Process;

end Architecture;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Misr International University
جامعة مصر الدولية

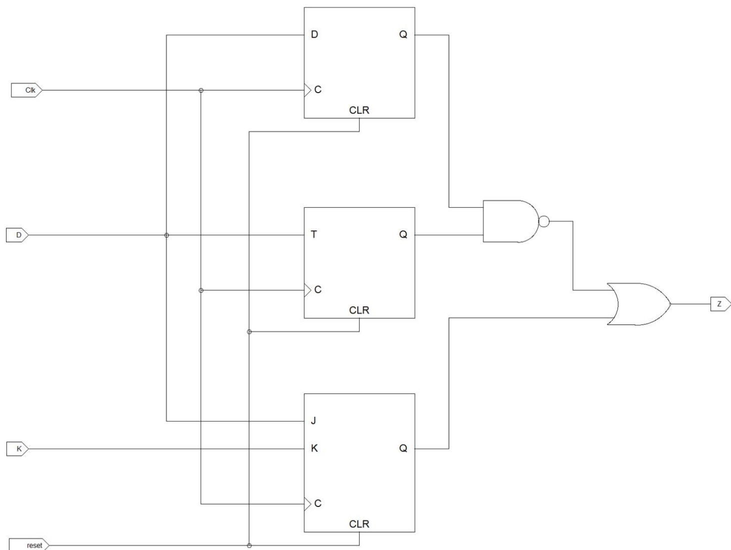
JK Flip Flops

Remember

```
entity JK_FF is
port (J,K,CLK : in std_logic;
        Q : out std_logic);
end entity;
Architecture behav of JK_FF is
signal Q_temp : std_logic :='0';
signal jk : std_logic_vector(1 downto 0);
begin
  jk <= J & K ;
  Process(CLK)
  begin
    if rising_edge(CLK) then
      if (jk="10") then
        Q_temp <= '1';
      elsif (jk="01") then
        Q_temp <= '0';
      if (jk="11") then
        Q_temp <= not(Q_temp);
      end if;
    end if;
  end Process;
    Q <= Q_temp;
end Architecture;
```

Flip Flops

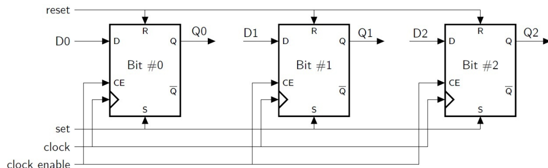
Check Your Understanding



Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Registers

General

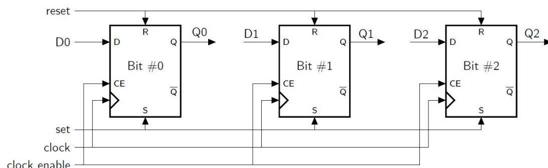



```
entity REG is



end entity;
Architecture beh of REG is
begin
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Registers

General

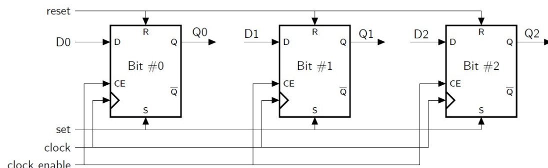



```
entity REG is
port (CLK, clk_en, reset, set : in std_logic;
        D: in std_logic_vector(7 downto 0);
        Q : out std_logic_vector(7 downto 0));
end entity;
Architecture beh of REG is
begin
  Process(CLK, reset, set)
  begin




  end Process;
```

Misr International University
جامعة مصر الدولية

Registers

General



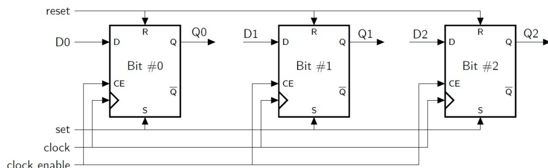

```
entity REG is
port (CLK, clk_en, reset, set : in std_logic;
        D: in std_logic_vector(7 downto 0);
        Q : out std_logic_vector(7 downto 0));
end entity;
Architecture beh of REG is
begin
  Process(CLK, reset, set)
  begin



      if rising_edge(CLK) then
        Q <= D ;
      end if;

  end Process;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

MIU
Misr International University
جامعة مصر الدولية

Registers

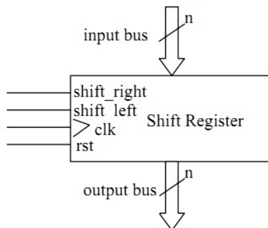
General

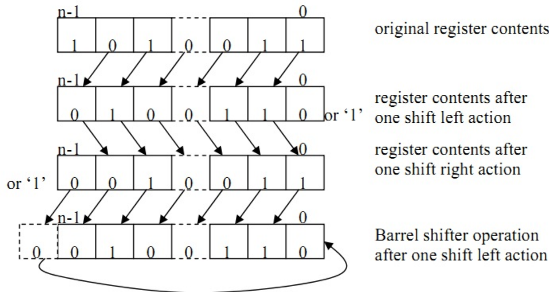



```
entity REG is
port (CLK, clk_en, reset, set : in std_logic;
        D: in std_logic_vector(7 downto 0);
        Q : out std_logic_vector(7 downto 0));
end entity;
Architecture beh of REG is
begin
  Process(CLK, reset, set)
  begin
    if reset = '1' then Q <= (others => '0') ;
    elsif set = '1' then Q <= (others => '1') ;
    elsif clk_en = '1' then
      if rising_edge(CLK) then
        Q <= D ;
      end if;
    end if;
  end Process;
```

Shift Registers

Key Component





Shift Registers

Key Component

```
entity SREG is



end entity;
Architecture beh of SREG is


begin









end Architecture;
```

Misr International University
جامعة مصر الدولية

Shift Registers

Key Component

```
entity SREG is
port (clk, rst, shift_right, shift_left: in std_logic;
        reg_in: in std_logic_vector(7 downto 0);
        reg_in : out std_logic_vector(7 downto 0));
end entity;
Architecture beh of SREG is
  signal shift_control: std_logic_vector(1 downto 0);
  signal reg_temp: std_logic_vector(7 downto 0);
begin
  shift_control <= shift_left & shift_right;
  Process(rst, clk)
  begin







  end Process;
end Architecture;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Misr International University
جامعة مصر الدولية

Shift Registers

Key Component

```
entity SREG is
port (clk, rst, shift_right, shift_left: in std_logic;
        reg_in: in std_logic_vector(7 downto 0);
        reg_in : out std_logic_vector(7 downto 0));
end entity;
Architecture beh of SREG is
  signal shift_control: std_logic_vector(1 downto 0);
  signal reg_temp: std_logic_vector(7 downto 0);
begin
  shift_control <= shift_left & shift_right;
  Process(rst, clk)
  begin
    if rst = '1' then reg_temp <= (others => '0') ;
    elsif rising_edge(CLK) then





    end if;
  end Process;
end Architecture;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Misr International University
جامعة مصر الدولية

Shift Registers

Key Component

```
entity SREG is
port (clk, rst, shift_right, shift_left: in std_logic;
        reg_in: in std_logic_vector(7 downto 0);
        reg_in : out std_logic_vector(7 downto 0));
end entity;
Architecture beh of SREG is
  signal shift_control: std_logic_vector(1 downto 0);
  signal reg_temp: std_logic_vector(7 downto 0);
begin
  shift_control <= shift_left & shift_right;
  Process(rst, clk)
  begin
    if rst = '1' then reg_temp <= (others => '0') ;
    elsif rising_edge(CLK) then
      case shift_control is





      end case;
    end if;
  end Process;
end Architecture;
```

Shift Registers

Key Component

```
entity SREG is
port (clk, rst, shift_right, shift_left: in std_logic;
        reg_in: in std_logic_vector(7 downto 0);
        reg_in : out std_logic_vector(7 downto 0));
end entity;
Architecture beh of SREG is
  signal shift_control: std_logic_vector(1 downto 0);
  signal reg_temp: std_logic_vector(7 downto 0);
begin
  shift_control <= shift_left & shift_right;
  Process(rst, clk)
  begin
    if rst = '1' then reg_temp <= (others => '0') ;
    elsif rising_edge(CLK) then
      case shift_control is
        when "00" => reg_temp <= reg_in;
        when "01" => reg_temp <= '0' & reg_temp(7 downto 1);
        when "10" => reg_temp <= reg_temp(6 downto 0) &  '0';
        when others => reg_temp <= reg_temp;
      end case;
    end if;
  end Process;
end Architecture;
```

Shift Registers

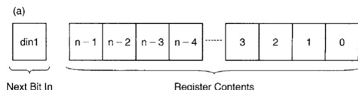
Serial to Parallel & Parallel to Serial Conversion

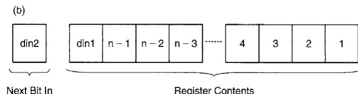
```
entity s2p is
port (clk, rst: in std_logic;
        reg_in: in std_logic;
        reg_out : out std_logic_vector(7 downto 0);
        out_ready : out std_logic);
end entity;
```

Shift Registers

Serial to Parallel & Parallel to Serial Conversion

```
entity s2p is
port (clk, rst: in std_logic;
        reg_in: in std_logic;
        reg_out : out std_logic_vector(7 downto 0);
        out_ready : out std_logic);
end entity;
```

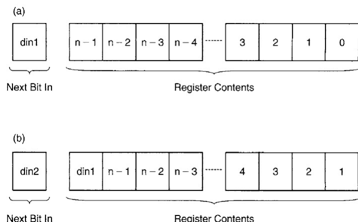




Shift Registers

Serial to Parallel & Parallel to Serial Conversion

```
entity s2p is
port (clk, rst: in std_logic;
        reg_in: in std_logic;
        reg_out : out std_logic_vector(7 downto 0);
        out_ready : out std_logic);
end entity;
```



Check Your Understanding

Write the corresponding VHDL codes?

Shift Registers

Serial to Parallel & Parallel to Serial Conversion

Misr International University
جامعة مصر الدولية

Shift Registers

Serial to Parallel & Parallel to Serial Conversion

```
entity p2s is
port (clk, rst: in std_logic;
        reg_in: in std_logic_vector(7 downto 0);
        reg_out : out std_logic);
end entity;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Misr International University
جامعة مصر الدولية

Shift Registers

Serial to Parallel & Parallel to Serial Conversion

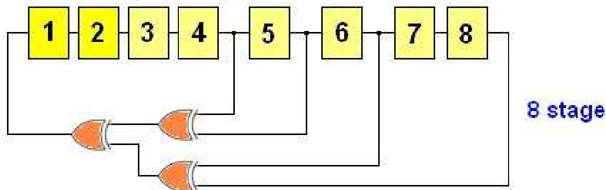
```
entity p2s is
port (clk, rst: in std_logic;
        reg_in: in std_logic_vector(7 downto 0);
        reg_out : out std_logic);
end entity;
```

Check Your Understanding

Write the corresponding VHDL codes?

Shift Registers

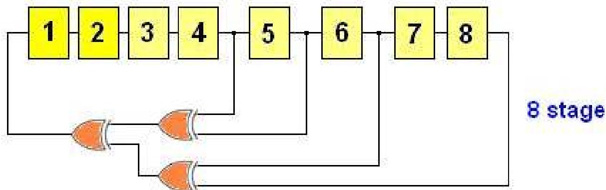
Linear Feedback Shift Register




```
entity lfsr is

port (clk, rst, enable: in std_logic;
        reg_out : out std_logic_vector(7 downto 0));
end entity;
```

Shift Registers

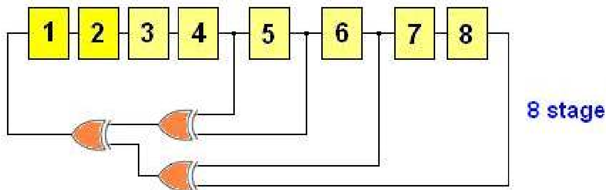
Linear Feedback Shift Register




```
entity lfsr is
generic(reg:std_logic_vector(7 downto 0):="00111010");
port (clk, rst, enable: in std_logic;
        reg_out : out std_logic_vector(7 downto 0));
end entity;
```

Shift Registers

Linear Feedback Shift Register




```
entity lfsr is
generic(reg:std_logic_vector(7 downto 0):="00111010");
port (clk, rst, enable: in std_logic;
        reg_out : out std_logic_vector(7 downto 0));
end entity;
```

Check Your Understanding

Consider the case when it is required to repeat the first 512 output symbols only, write the corresponding VHDL code.

Counters

Behavioral

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity COUNT8 is
 port(
      DIN       : in    std_logic_vector(7 downto 0);
      CLK       : in    std_logic;
      LOAD      : in    std_logic;
      DOUT      : out   std_logic_vector(7 downto 0));
end COUNT8;

architecture behavior of COUNT8 is
begin
        -- notice the process statement and the variable COUNT
        clk_proc:process(CLK)
        variable COUNT:unsigned(7 downto 0) := "00000000";
        begin
                if (CLK'EVENT AND CLK = '1') then
                        if LOAD = '1' then
                                COUNT := DIN;
                        else COUNT := COUNT + 1;
                        end if;
                end if;
                DOUT <= COUNT after 50 ns;
        end process clk_proc;
end behavior;
```

Counters

Behavioral

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

entity Counter2_VHDL is
   port( Clock,Reset,Clock_enable_B: in std_logic;
         Output: out std_logic_vector(0 to 3));
end Counter2_VHDL;
architecture Behavioral of Counter2_VHDL is
   signal temp: std_logic_vector(0 to 3);
begin   process(Clock,Reset)
   begin
      if Reset='1' then
         temp <= "0000";
      elsif(Clock'event and Clock='1') then
         if Clock_enable_B='0' then
            if temp="1001" then
               temp<="0000";
            else
               temp <= temp + 1;
            end if;
         end if;
      end if;
   end process;
   Output <= temp;
end Behavioral;
```

Chapter 5
Sequential Circuits
Chapter 6
Chapter 7
Chapter 8

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Outline

1 Chapter 5
2 Chapter 6
3 Chapter 7
4 Chapter 8

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Finite State Machines

- Finite State Machine is a tool to model the desired behavior of a sequential system.

Finite State Machines

- Finite State Machine is a tool to model the desired behavior of a sequential system.
- A FSM consists of several states, Inputs into the machine are combined with the current state of the machine to determine the new state or next state of the machine.

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Finite State Machines

- Finite State Machine is a tool to model the desired behavior of a sequential system.
- A FSM consists of several states, Inputs into the machine are combined with the current state of the machine to determine the new state or next state of the machine.
- Depending on the state of the machine, outputs are generated based on either the state or the state and inputs of the machine.

Finite State Machines

- Finite State Machine is a tool to model the desired behavior of a sequential system.
- A FSM consists of several states, Inputs into the machine are combined with the current state of the machine to determine the new state or next state of the machine.
- Depending on the state of the machine, outputs are generated based on either the state or the state and inputs of the machine.
- The designer has to develop a finite state model of the system behavior and then designs a circuit that implements this model.

Finite State Machines

- Finite State Machine is a tool to model the desired behavior of a sequential system.
- A FSM consists of several states, Inputs into the machine are combined with the current state of the machine to determine the new state or next state of the machine.
- Depending on the state of the machine, outputs are generated based on either the state or the state and inputs of the machine.

- The designer has to develop a finite state model of the system behavior and then designs a circuit that implements this model.
  - Design the State Diagram for the required Functionality.

Chapter 5

Chapter 6
Finite State Machines

Chapter 7

Chapter 8

Finite State Machines

- Finite State Machine is a tool to model the desired behavior of a sequential system.
- A FSM consists of several states, Inputs into the machine are combined with the current state of the machine to determine the new state or next state of the machine.
- Depending on the state of the machine, outputs are generated based on either the state or the state and inputs of the machine.

- The designer has to develop a finite state model of the system behavior and then designs a circuit that implements this model.
  - Design the State Diagram for the required Functionality.
  - Implement that FSM design using "Flip Flops" (and Visa Versa)

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Finite State Machines

- Finite State Machine is a tool to model the desired behavior of a sequential system.
- A FSM consists of several states, Inputs into the machine are combined with the current state of the machine to determine the new state or next state of the machine.
- Depending on the state of the machine, outputs are generated based on either the state or the state and inputs of the machine.

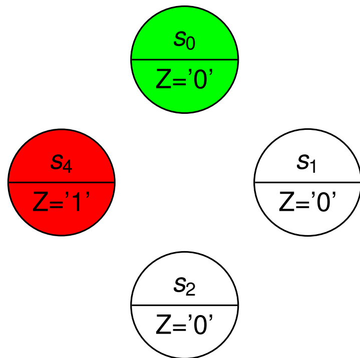
- The designer has to develop a finite state model of the system behavior and then designs a circuit that implements this model.
  - Design the State Diagram for the required Functionality.
  - Implement that FSM design using "Flip Flops" (and Visa Versa)
  - Represent vthat FSM using VHDL code (and Visa Versa)

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Finite State Machine

State Diagram

The FSM diagram consists of State Bubbles and conditions controlling transitions among states.

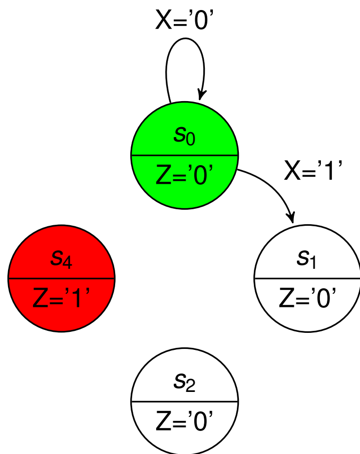


Finite State Machine

State Diagram

The FSM diagram consists of State Bubbles and conditions controlling transitions among states.

- the lines connecting state bubbles starts from the current state and ends at the next state
- The conditions related to s_0 state that if X=1 then the system will change its state to s_1 otherwise it will remain s_0. and the output will remain **Z=0**.

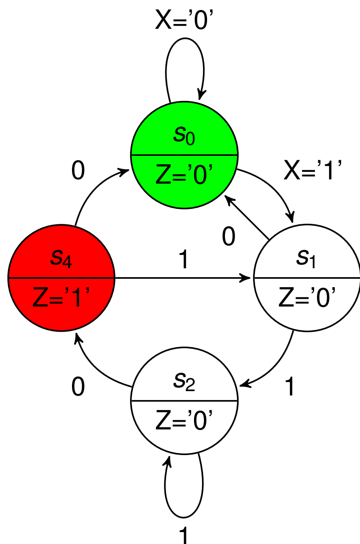


Finite State Machine

State Diagram

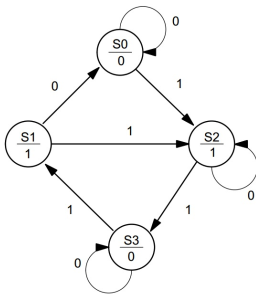
The FSM diagram consists of State Bubbles and conditions controlling transitions among states.

- the lines connecting state bubbles starts from the current state and ends at the next state
- The conditions related to s_0 state that if X=1 then the system will change its state to s_1 otherwise it will remain s_0. and the output will remain **Z=0**.



Moore vs. Mealy FSM

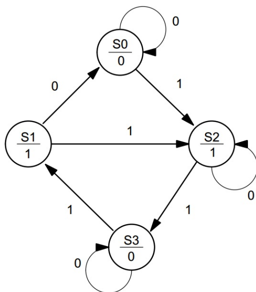
Moore



Present state	Next state		Output (Z)
	X=0	X=1	X=0
S0	S0	S2	0
S1	S0	S2	1
S2	S2	S3	1
S3	S3	S1	0

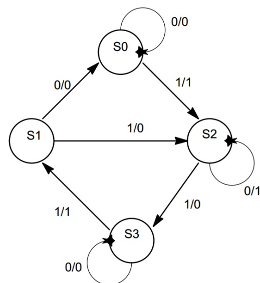
Moore vs. Mealy FSM

Moore



Present state	Next state		Output (Z)
	X=0	X=1	X=0
S0	S0	S2	0
S1	S0	S2	1
S2	S2	S3	1
S3	S3	S1	0

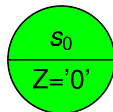
Mealy



Present state	Next state		Output (Z)	
	X=0	X=1	X=0	X=1
S0	S0	S2	0	1
S1	S0	S2	0	0
S2	S2	S3	1	0
S3	S3	S1	0	1

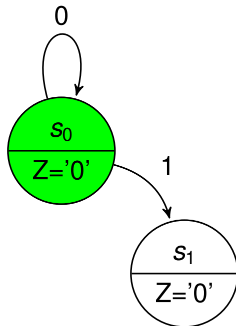
FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'



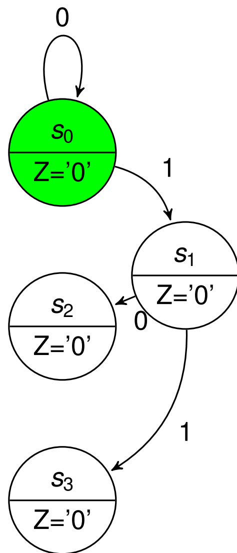
FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'



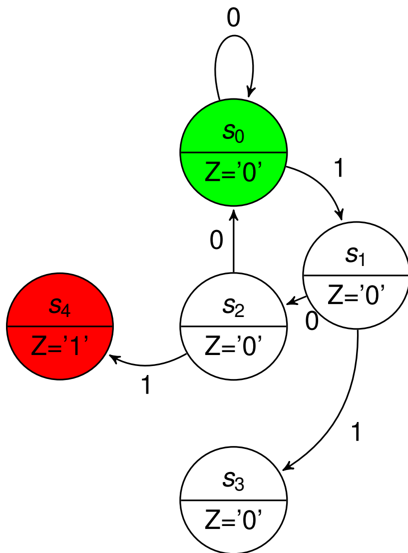
FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'



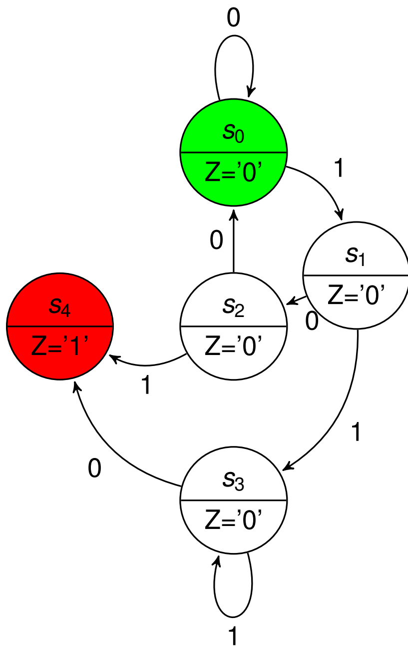
FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'



FSM Example

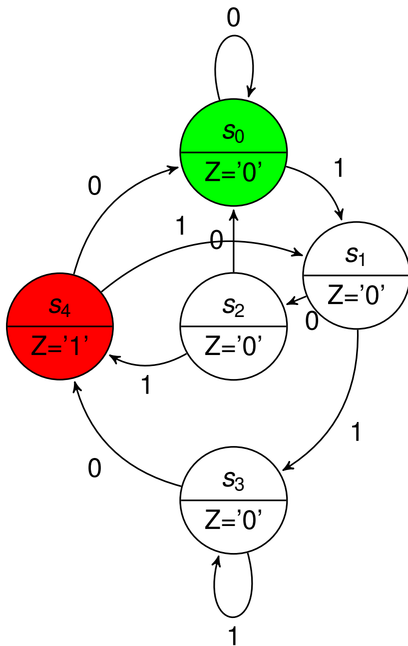
Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'



Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

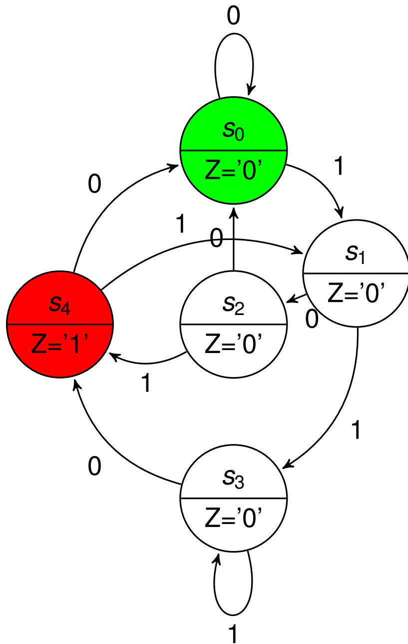


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z

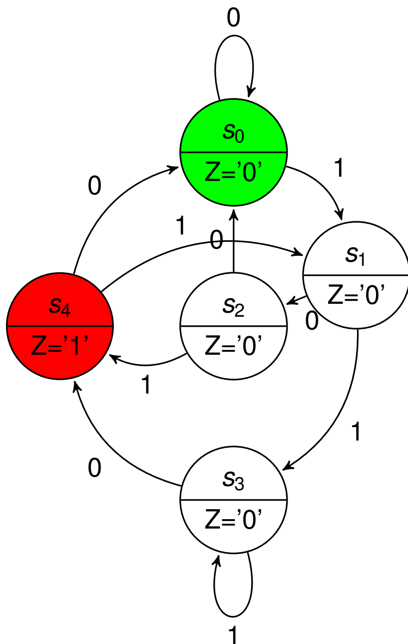


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0

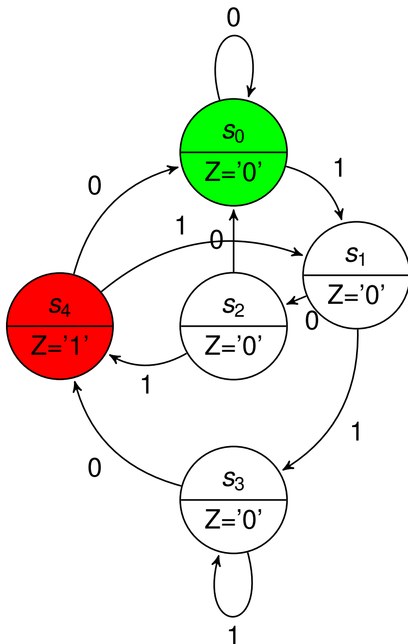


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0

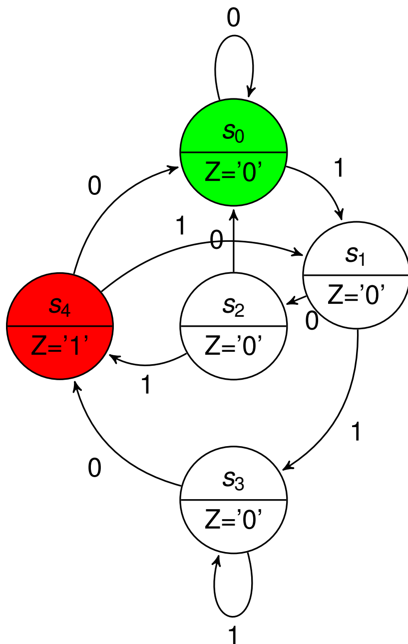


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0

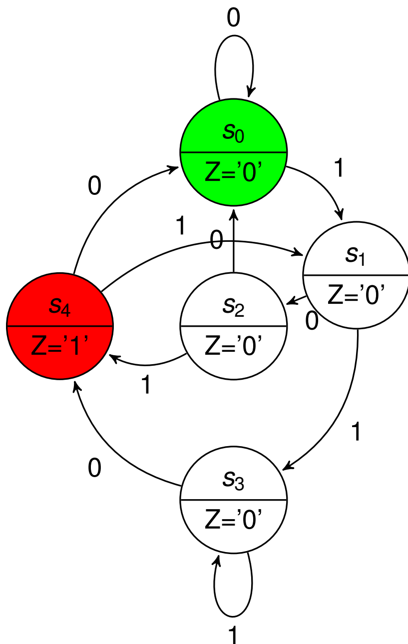


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0

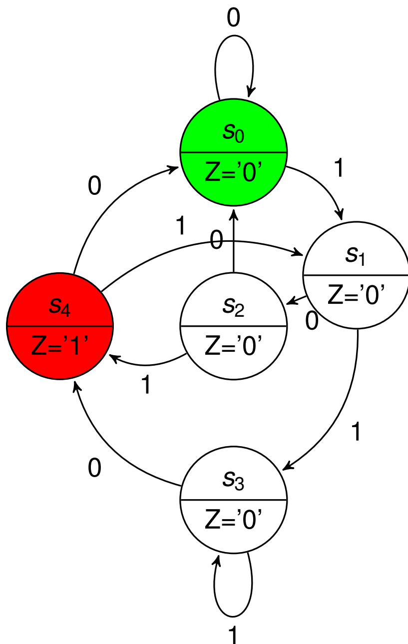


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0

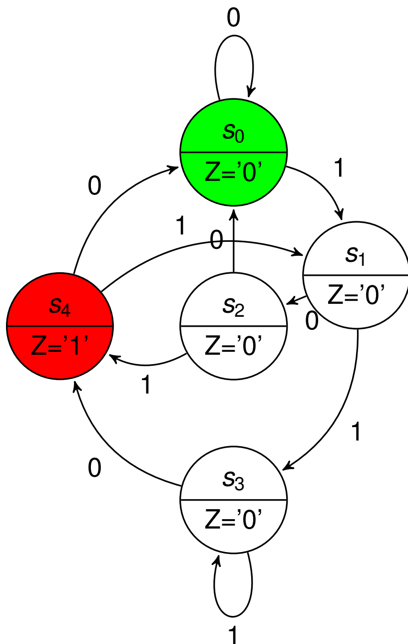


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1Q_2Q_3$	W	$D_1D_2D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0

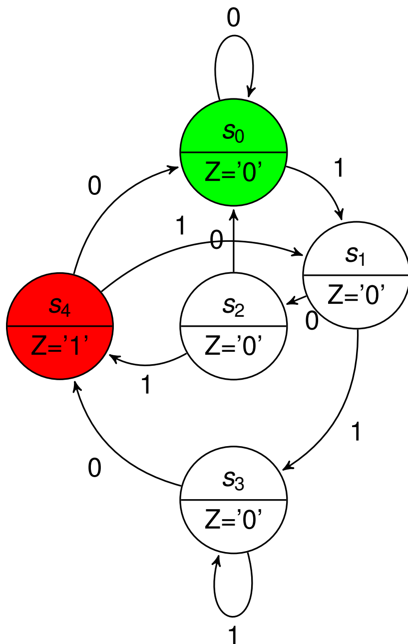


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1Q_2Q_3$	W	$D_1D_2D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0
011	0	100	0

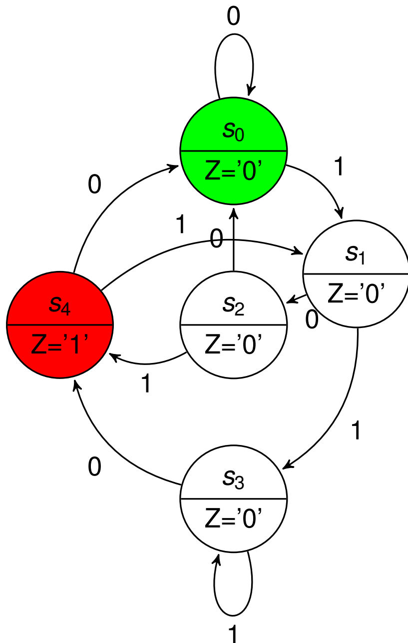


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0
011	0	100	0
011	1	011	0

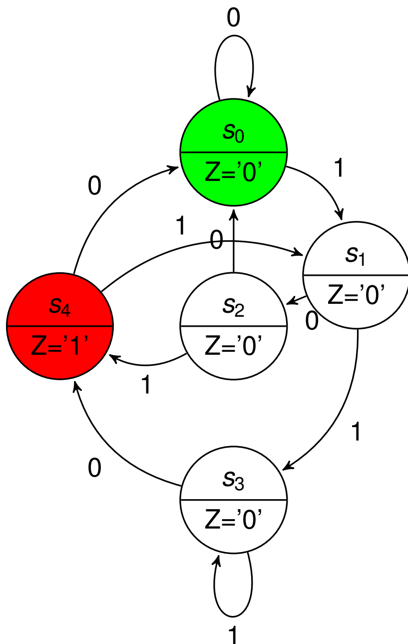


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0
011	0	100	0
011	1	011	0
100	0	000	1

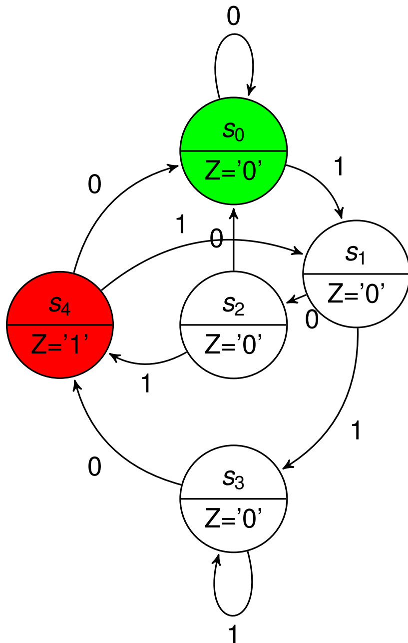


FSM Example

Desigm a module with O/P Z = '1' when the previous values of W are "110" or "101" otherwise Z = '0'

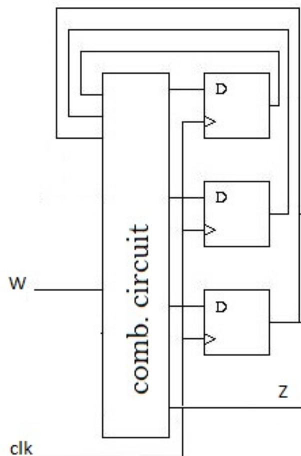
We have **5 States** so, it is required to have **3 Flip Flops** to store the states for the next time slot.

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0
011	0	100	0
011	1	011	0
100	0	000	1
100	1	001	1



FSM Example

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0
011	0	100	0
011	1	011	0
100	0	000	1
100	1	001	1

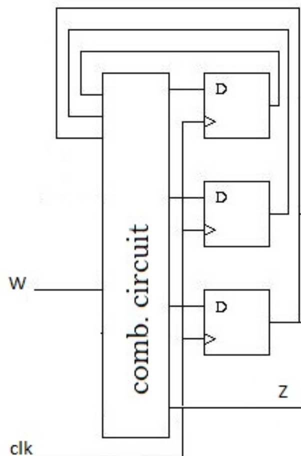


Boolean Equations

$Z = Q_1.\overline{Q_2}.\overline{Q_3}$

FSM Example

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0
011	0	100	0
011	1	011	0
100	0	000	1
100	1	001	1



Boolean Equations

$Z = Q_1.\overline{Q_2}.\overline{Q_3}$

$D_1 = \overline{Q_1}.Q_2.\overline{Q_3} + \overline{Q_1}.Q_2.Q_3$

FSM Example

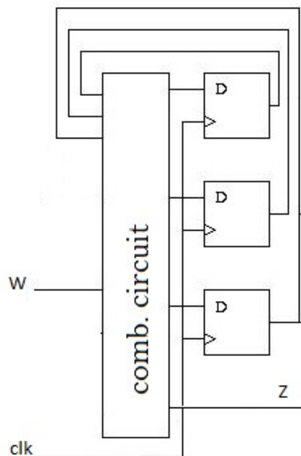
Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0
011	0	100	0
011	1	011	0
100	0	000	1
100	1	001	1

Boolean Equations

$Z = Q_1.\overline{Q_2}.\overline{Q_3}$

$D_1 = \overline{Q_1}.Q_2.\overline{Q_3} + \overline{Q_1}.Q_2.Q_3$

$D_2 = ... + ... + ...$



FSM Example

Current	input	next	output
$Q_1 Q_2 Q_3$	W	$D_1 D_2 D_3$	Z
000	0	000	0
000	1	001	0
001	0	010	0
001	1	011	0
010	0	000	0
010	1	100	0
011	0	100	0
011	1	011	0
100	0	000	1
100	1	001	1

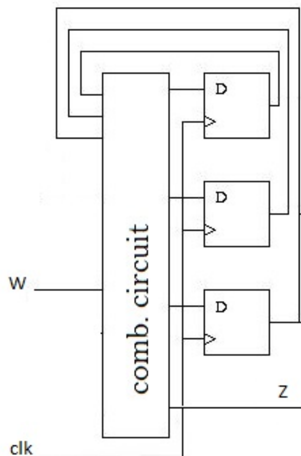
Boolean Equations

$Z = Q_1.\overline{Q_2}.\overline{Q_3}$

$D_1 = \overline{Q_1}.Q_2.\overline{Q_3} + \overline{Q_1}.Q_2.Q_3$

$D_2 = ... + ... + ...$

$D_3 = ... + ... + ... + ...$



Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
```

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin

End behav;
```

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin
Process(CLK, RST)
Begin
if (RST='1') then C_state <= s0;
elsif (rising_edge(CLK)) then C_state <= N_state;
end if;
End Process;

End behav;
```

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin
Process(CLK, RST)
Begin
if (RST='1') then C_state <= s0;
elsif (rising_edge(CLK)) then C_state <= N_state;
end if;
End Process;
Process(W, C_state)
Begin
Case C_state is









End case;
End Process;

End behav;
```

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin
Process(CLK, RST)
Begin
if (RST='1') then C_state <= s0;
elsif (rising_edge(CLK)) then C_state <= N_state;
end if;
End Process;
Process(W, C_state)
Begin
Case C_state is
when s0 => if (W='1') then N_state <= s1; end if;









End case;
End Process;

End behav;
```

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin
Process(CLK, RST)
Begin
if (RST='1') then C_state <= s0;
elsif (rising_edge(CLK)) then C_state <= N_state;
end if;
End Process;
Process(W, C_state)
Begin
Case C_state is
when s0 => if (W='1') then N_state <= s1; end if;
when s1 =>
         if (W='0') then N_state <= s2; else N_state <= s3; end if;







End case;
End Process;

End behav;
```

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin
Process(CLK, RST)
Begin
if (RST='1') then C_state <= s0;
elsif (rising_edge(CLK)) then C_state <= N_state;
end if;
End Process;
Process(W, C_state)
Begin
Case C_state is
when s0 => if (W='1') then N_state <= s1; end if;
when s1 =>
         if (W='0') then N_state <= s2; else N_state <= s3; end if;
when s3 =>
         if (W='0') then N_state <= s4; end if;




End case;
End Process;

End behav;
```

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin
Process(CLK, RST)
Begin
if (RST='1') then C_state <= s0;
elsif (rising_edge(CLK)) then C_state <= N_state;
end if;
End Process;
Process(W, C_state)
Begin
Case C_state is
when s0 => if (W='1') then N_state <= s1; end if;
when s1 =>
         if (W='0') then N_state <= s2; else N_state <= s3; end if;
when s3 =>
         if (W='0') then N_state <= s4; end if;
when s2 =>
         if (W='0') then N_state <= s0; else N_state <= s4; end if;


End case;
End Process;

End behav;
```

Finite State Machine

Example

```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin
Process(CLK, RST)
Begin
if (RST='1') then C_state <= s0;
elsif (rising_edge(CLK)) then C_state <= N_state;
end if;
End Process;
Process(W, C_state)
Begin
Case C_state is
when s0 => if (W='1') then N_state <= s1; end if;
when s1 =>
         if (W='0') then N_state <= s2; else N_state <= s3; end if;
when s3 =>
         if (W='0') then N_state <= s4; end if;
when s2 =>
         if (W='0') then N_state <= s0; else N_state <= s4; end if;
when s4 =>
         if (W='0') then N_state <= s0; else N_state <= s1; end if;
End case;
End Process;

End behav;
```

Finite State Machine

Example

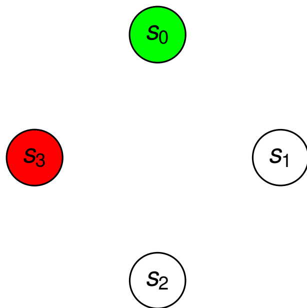
```
entity TOP_MODULE is
    port (CLK, RST, W : IN STD_LOGIC;  Z : OUT STD_LOGIC);
end TOP_MODULE;
Architecture behav of TOP_MODULE is
Type State is (s0,s1,s2,s3,s4);
Signal C_state, N_state : state;
Begin
Process(CLK, RST)
Begin
if (RST='1') then C_state <= s0;
elsif (rising_edge(CLK)) then C_state <= N_state;
end if;
End Process;
Process(W, C_state)
Begin
Case C_state is
when s0 => if (W='1') then N_state <= s1; end if;
when s1 =>
         if (W='0') then N_state <= s2; else N_state <= s3; end if;
when s3 =>
         if (W='0') then N_state <= s4; end if;
when s2 =>
         if (W='0') then N_state <= s0; else N_state <= s4; end if;
when s4 =>
         if (W='0') then N_state <= s0; else N_state <= s1; end if;
End case;
End Process;
Z<='1' when (C_state = s4) else '0';
End behav;
```

Example

state table

Draw the state diagram for the **FSM** represented by the following State Table

State	Next		Output	
Current	in 0	in 1	in 0	in 1
s_0	s_0	s_1	0	1
s_1	s_3	s_2	1	1
s_2	s_1	s_0	1	0
s_3	s_0	s_1	0	0



Note that this is **Mealy FSM**

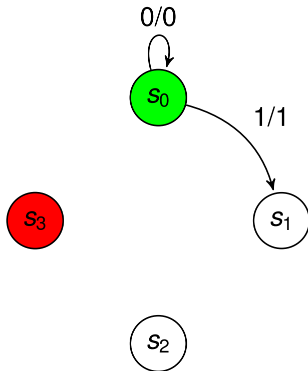
Can you implement the equivalent circuit with different types of Flip Flops?

Example

state table

Draw the state diagram for the **FSM** represented by the following State Table

State	Next		Output	
Current	in 0	in 1	in 0	in 1
s_0	s_0	s_1	0	1
s_1	s_3	s_2	1	1
s_2	s_1	s_0	1	0
s_3	s_0	s_1	0	0



Note that this is **Mealy FSM**

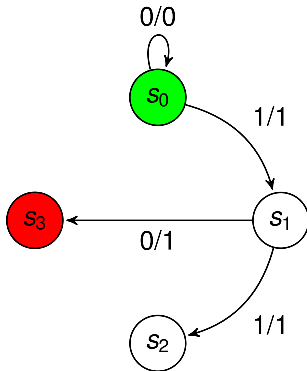
Can you implement the equivalent circuit with different types of Flip Flops?

Example

state table

Draw the state diagram for the **FSM** represented by the following State Table

State	Next		Output	
Current	in 0	in 1	in 0	in 1
s_0	s_0	s_1	0	1
s_1	s_3	s_2	1	1
s_2	s_1	s_0	1	0
s_3	s_0	s_1	0	0



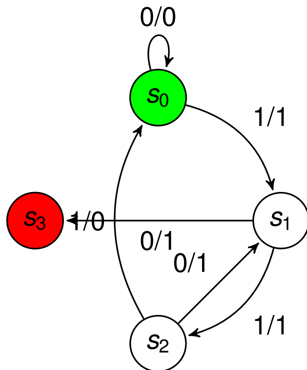
Note that this is **Mealy FSM**
Can you implement the equivalent circuit with different types of Flip Flops?

Example

state table

Draw the state diagram for the **FSM** represented by the following State Table

State	Next		Output	
Current	in 0	in 1	in 0	in 1
s_0	s_0	s_1	0	1
s_1	s_3	s_2	1	1
s_2	s_1	s_0	1	0
s_3	s_0	s_1	0	0



Note that this is **Mealy FSM**

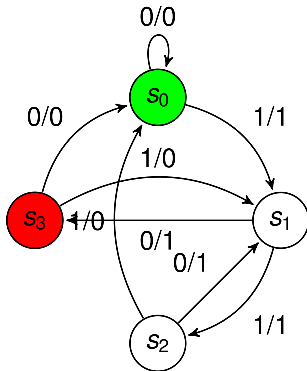
Can you implement the equivalent circuit with different types of Flip Flops?

Example

state table

Draw the state diagram for the **FSM** represented by the following State Table

State	Next		Output	
Current	in 0	in 1	in 0	in 1
s_0	s_0	s_1	0	1
s_1	s_3	s_2	1	1
s_2	s_1	s_0	1	0
s_3	s_0	s_1	0	0



Note that this is **Mealy FSM**

Can you implement the equivalent circuit with different types of Flip Flops?

Example

state table

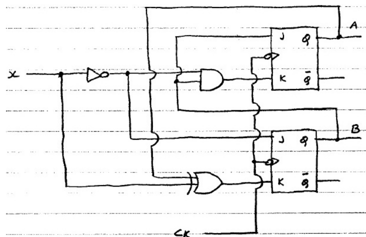
Draw the state diagram for the **FSM** represented by the following State Table

State	Next				
Current	00	01	11	10	**output**
Init	A_0	A_0	A_1	A_1	0
A_0	OK_0	OK_0	A_1	A_1	0
A_1	A_0	A_0	OK_1	OK_1	0
OK_0	OK_0	OK_0	OK_1	A_1	1
OK_1	A_0	OK_0	OK_1	OK_1	1

Note that this is **Moore FSM** Why?
Can you implement the equivalent circuit with different types of Flip Flops?

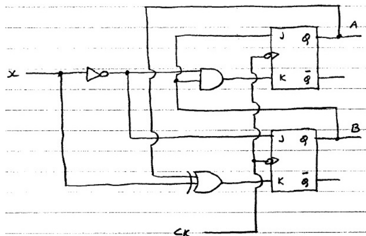
Logic Circuit Example

Consider the Circuit Shown:



Logic Circuit Example

Consider the Circuit Shown:



$J_A = B \quad , \quad K_A = B \cdot \bar{x}$

$J_B = \bar{x} \quad , \quad K_B = \bar{A} \cdot x + A \cdot \bar{x} = A \oplus x$

Excitation Table for the JK Flip-flop

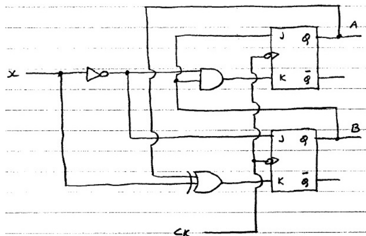
Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Can you extract the FSM?

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Logic Circuit Example

Consider the Circuit Shown:



$J_A = B$, $K_A = B \cdot \bar{x}$

$J_B = \bar{x}$, $K_B = \bar{A} \cdot x + A \cdot \bar{x} = A \oplus x$

Excitation Table for the JK Flip-flop

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

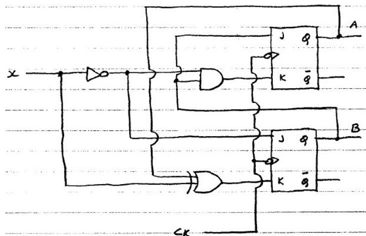
Can you extract the FSM?

A	B	x	J_A	K_A	J_B	K_B	A^+	B^+
0	0	0						

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Logic Circuit Example

Consider the Circuit Shown:



$J_A = B$, $K_A = B \cdot \bar{x}$

$J_B = \bar{x}$, $K_B = \bar{A} \cdot x + A \cdot \bar{x} = A \oplus x$

Excitation Table for the JK Flip-flop

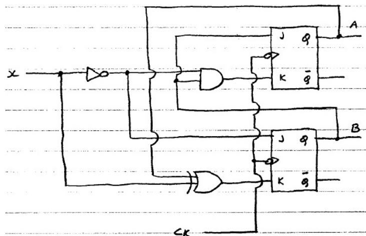
Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Can you extract the FSM?

A	B	x	J_A	K_A	J_B	K_B	A^+	B^+
0	0	0	0	0	1	0		

Logic Circuit Example

Consider the Circuit Shown:



$J_A = B \quad , \quad K_A = B \cdot \bar{x}$

$J_B = \bar{x} \quad , \quad K_B = \bar{A} \cdot x + A \cdot \bar{x} = A \oplus x$

Excitation Table for the JK Flip-flop

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Can you extract the FSM?

A	B	x	J_A	K_A	J_B	K_B	A^+	B^+
0	0	0	0	0	1	0	0	1
0	0	1						

Chapter 5

Chapter 6

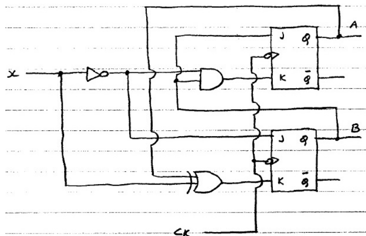
Finite State Machines

Chapter 7

Chapter 8

Logic Circuit Example

Consider the Circuit Shown:



$J_A = B$, $K_A = B \cdot \bar{x}$

$J_B = \bar{x}$, $K_B = \bar{A} \cdot x + A \cdot \bar{x} = A \oplus x$

Excitation Table for the JK Flip-flop

Q	Q^+	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

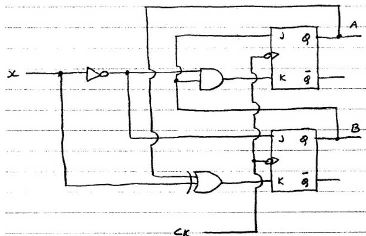
Can you extract the FSM?

A	B	x	J_A	K_A	J_B	K_B	A^+	B^+
0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	1		

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Logic Circuit Example

Consider the Circuit Shown:



$J_A = B \quad , \quad K_A = B \cdot \bar{x}$

$J_B = \bar{x} \quad , \quad K_B = \bar{A} \cdot x + A \cdot \bar{x} = A \oplus x$

Excitation Table for the JK Flip-flop

Q	Q^+	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

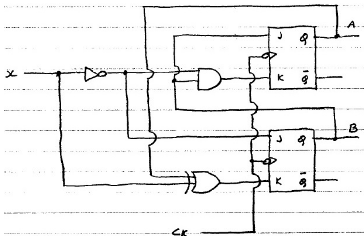
Can you extract the FSM?

A	B	x	J_A	K_A	J_B	K_B	A^+	B^+
0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Logic Circuit Example

Consider the Circuit Shown:



$J_A = B$, $K_A = B \cdot \bar{x}$

$J_B = \bar{x}$, $K_B = \bar{A} \cdot x + A \cdot \bar{x} = A \oplus x$

Excitation Table for the JK Flip-flop

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Can you extract the FSM?

A	B	x	J_A	K_A	J_B	K_B	A^+	B^+
0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0
0	1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1	0
1	0	0	0	0	1	1	1	1
1	0	1	0	0	0	0	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	0	0	0	1	1

Chapter 5

Chapter 6

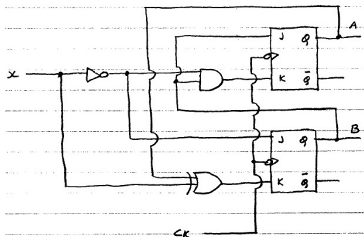
Finite State Machines

Chapter 7

Chapter 8

Logic Circuit Example

Consider the Circuit Shown:



$J_A = B$, $K_A = B \cdot \bar{x}$

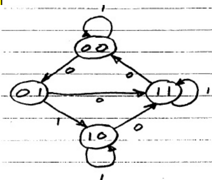
$J_B = \bar{x}$, $K_B = \bar{A} \cdot x + A \cdot \bar{x} = A \oplus x$

Excitation Table for the JK Flip-flop

Q	Q^+	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

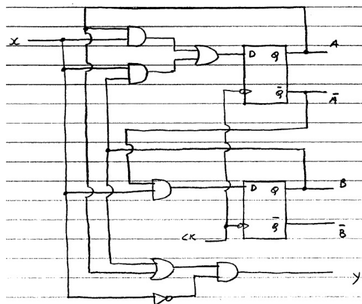
Can you extract the FSM?

A	B	x	J_A	K_A	J_B	K_B	A^+	B^+
0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0
0	1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1	0
1	0	0	0	0	1	1	1	1
1	0	1	0	0	0	0	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	0	0	0	1	1



Logic Circuit Example

Consider the Circuit Shown:



Input Functions

$$DA = A \cdot x + B \cdot x$$

$$DB = \bar{A} \cdot x$$

$$y = (A + B) \cdot \bar{x}$$

Chapter 5

Chapter 6

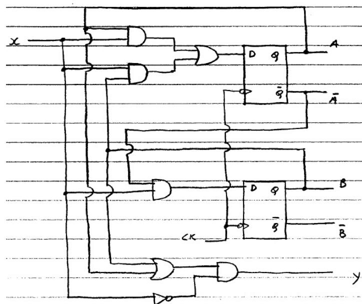
Finite State Machines

Chapter 7

Chapter 8

Logic Circuit Example

Consider the Circuit Shown:



Input Functions

$DA = A \cdot x + B \cdot x$

$DB = \bar{A} \cdot x$

$y = (A + B) \cdot \bar{x}$

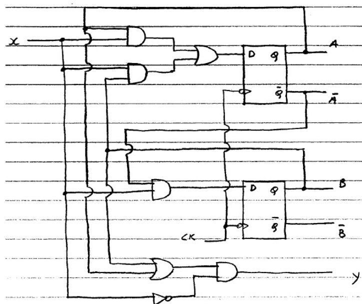
Note that this is **Mealy FSM**

Why?

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Logic Circuit Example

Consider the Circuit Shown:



State Transition Table

		Input	DA	DB	Output
A	B	x	A+	B+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Input Functions

$$DA = A \cdot x + B \cdot x$$

$$DB = \bar{A} \cdot x$$

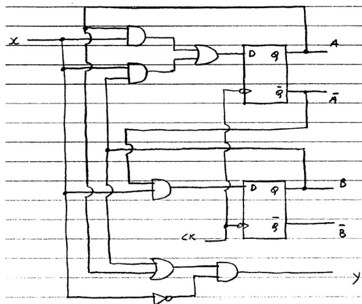
$$y = (A + B) \cdot \bar{x}$$

Note that this is **Mealy FSM**

Why?

Logic Circuit Example

Consider the Circuit Shown:



State Transition Table

		Input	DA	DB	Output
A	B	x	A^+	B^+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

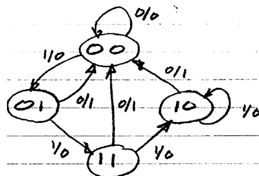
Input Functions

$$DA = A \cdot x + B \cdot x$$

$$DB = \bar{A} \cdot x$$

$$y = (A + B) \cdot \bar{x}$$

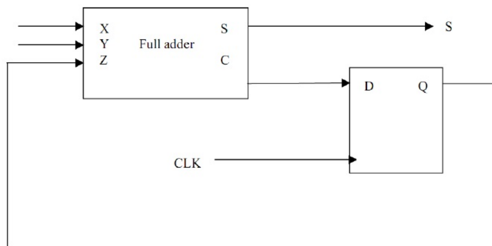
State Transition Diagram



Note that this is **Mealy FSM**
Why?

Logic Circuit Example

The sequential circuit below has one D flip-flop, two inputs (X and Y), and one output (S). The circuit consists of a full adder circuit whose carry output is connected to a D flip-flop.



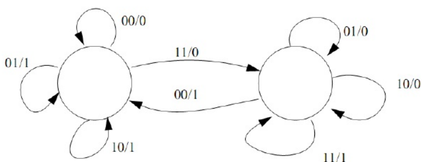
1. Derive the truth table for the above circuit.

2. Draw the state diagram for this circuit (be sure to show the output S in the diagram).

3. Is the finite state machine of the Moore type or the Mealy type? Explain.

Logic Circuit Example

x	y	z	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Mealy. S depends on the state and the inputs.

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Counter Example

Design a 3 bit counter which counts in the sequence:

001, 011, 010, 110, 111, 101, <u>001</u>, …

(a) Use clocked D flip-flops

(b) Use clocked T flip-flop

(c) Use J-K flip-flop

Reminder: The following table summarizes state changes for D, T, and J-K flip-flops

$Q \rightarrow Q_+$	D flip-flop D	Toggle Flip-flop T	J-K Flip-flop J	 K	 J	 K
$0 \rightarrow 0$	0	0	0 0	1 0	0	x
$0 \rightarrow 1$	1	1	1 1	0 1	1	x
$1 \rightarrow 0$	0	1	0 1	1 1	x	1
$1 \rightarrow 1$	1	0	0 1	0 0	x	0

Counter Example

Step1. Determine the number of flip-flop stages required. In this case it is 3.
Step2. Flip-flop outputs $Q_2Q_1Q_0$ are determined by the specifications. Present state and next state tables are constructed. The present state-next state table (quick and full) is shown below. Make sure that the counter counts in the following order:

$$001 \rightarrow 011 \rightarrow 010 \rightarrow 110 \rightarrow 111 \rightarrow 101$$

Quick table:

Present-state			Next-state		
Q_2	Q_1	Q_0	Q_{2+}	Q_{1+}	Q_{0+}
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	0	0	1

On Karnaugh map, put x for the states that does not exist in this table

Full table:

Present-state			Next-state		
Q_2	Q_1	Q_0	Q_{2+}	Q_{1+}	Q_{0+}
0	0	0	x	x	x
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	x	x	x
1	0	1	0	0	1
1	1	0	1	1	1
1	1	1	1	0	1

Counter Example

Step 3. Find state transition relationships for each pair of present-state-next-state columns, using the state change table for the chosen flip-flop.
Step 4. Use the complete present state as input to truth tables whose outputs are state transition values corresponding to each flip-flop input.
Step 5. With map or Boolean algebra methods find expressions (excitation equations) for each flip-flop input from the truth tables of step 4.
Step 6. Form system outputs from combinations of the flip-flop outputs.
Step 7. Realize the excitation expressions as combinational logic drives for the flip-flop inputs.
Step 8. You can check your design by constructing a timing diagram.

Counter Example

(a) Implementation using D flip-flop. (Since the quick table is used, do not forget to put x to the respective boxes in the Karnaugh map for missing states in the table)

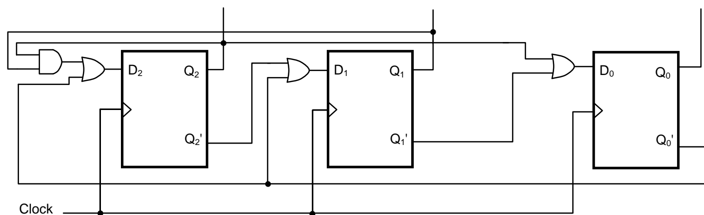
Present-state			Next-state			Flip-flop input		
Q_2	Q_1	Q_0	Q_{2+}	Q_{1+}	Q_{0+}	D_2	D_1	D_0
0	0	1	0	1	1	0	1	1
0	1	1	0	1	0	0	1	0
0	1	0	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	0	1
1	0	1	0	0	1	0	0	1

Q_2 \ Q_1Q_0	00	01	11	10	00	01	11	10	00	01	11	10
0	x	0	0	1	x	1	1	1	x	1	0	0
1	x	0	1	1	x	0	0	1	x	1	1	1

$D_2 = Q_0' + Q_2Q_1$ $D_1 = Q_2' + Q_0'$ $D_0 = Q_1' + Q_2$

Counter Example

$D_2 = Q_0' + Q_2Q_1$ $D_1 = Q_2' + Q_0'$ $D_0 = Q_1' + Q_2$



Counter Example

(b) Implementation using T flip-flop. The following table gives the states of the flip-flop inputs. (use the reminder if you don't remember the state changes).

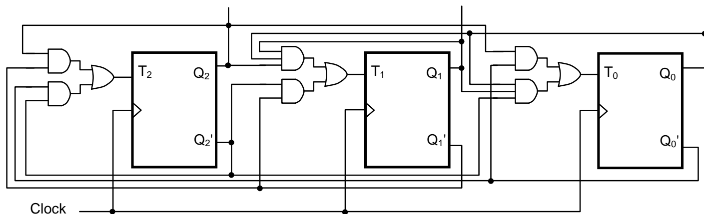
Present-state			Next-state			Flip-flop input		
Q_2	Q_1	Q_0	Q_{2+}	Q_{1+}	Q_{0+}	T_2	T_1	T_0
0	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	1	0	1	0	1	0
1	0	1	0	0	1	1	0	0

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Counter Example

Q_2 \ Q_1Q_0	T_2: 00	01	11	10	T_1: 00	01	11	10	T_0: 00	01	11	10
0	x	0	0	1	x	1	0	0	x	0	1	0
1	x	1	0	0	x	0	1	0	x	0	0	1

$T_2 = Q_2'Q_0' + Q_2Q_1'$ $T_1 = Q_2'Q_1' + Q_2Q_1Q_0$ $T_0 = Q_2Q_0' + Q_2'Q_1Q_0$



State Reduction

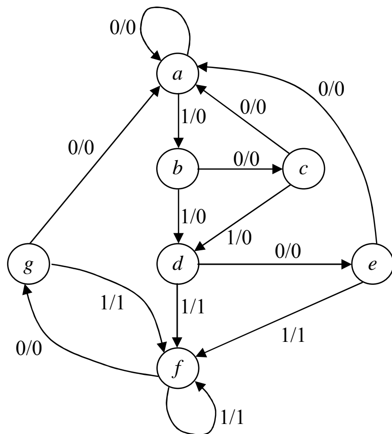
The state diagram of a sequential circuit is given as below.

(a) Tabulate the related state table.

(b) Reduce the state table to a minimum number of states using row matching.

(c)

Draw the reduced state diagram.



State Reduction

Present state	Next state $x=0$	Next state $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

State Reduction

Present state	Next state $x=0$	Next state $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

State Reduction

Present state	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Chapter 5
Chapter 6
Finite State Machines
Chapter 7
Chapter 8

Chapter 5
Chapter 6
Chapter 7
VHDL to Silicon Design Flow
Chapter 8

Outline

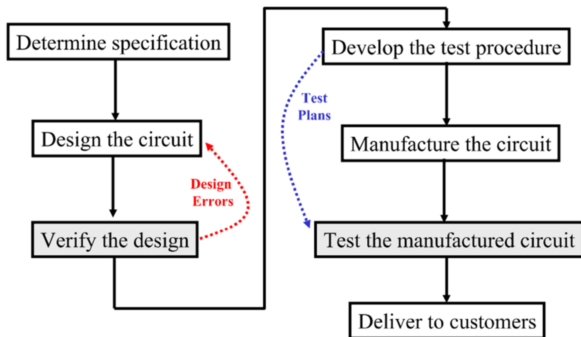
1. Chapter 5
2. Chapter 6
3. Chapter 7
4. Chapter 8

Testing of Logic Circuits

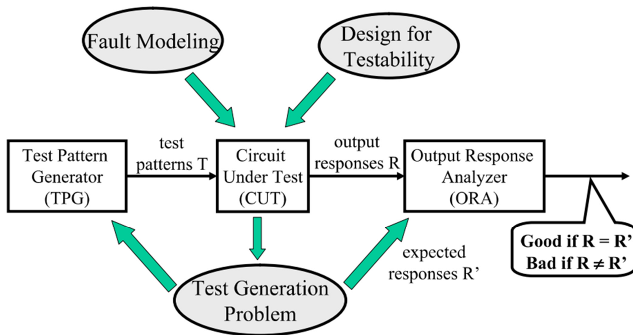
Chapter 11

Ref.1

VLSI Development Flow



Key Issues in Testing



Fault Models

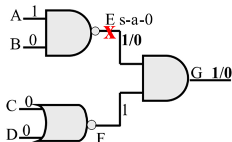
- Stuck-at faults
- Bridging faults
- PLA faults
- Transistor stuck-on/open faults
- Delay faults
- Functional faults
- State transition faults
- Memory faults

Complexity of a test

- Testing is done via applying a test patterns and comparing the O/P with the good O/P.
- A circuit with k inputs requires 2^k test patterns, which is a huge task for large values of k.
- Fortunately, not all the 2^k patterns must be used to cover an acceptable percentage of the probable faults in the circuit.

Test Pattern Generation
Combinational Circuits testing

- Fault Simulation
  - Determine the behavior of faulty circuit for all stuck-at fault wires.
  - Choose test patterns that cover faults



- Path sensitizing
  - deal with several wires that form a path as an entity that can be tested for several faults using a single test

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

w_1 a
w_2 b
w_3 c
d
f

Test $w_1 w_2 w_3$	Fault detected									
	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000										
001										
010										
011										
100										
101										
110										
111										

Chapter 5
Chapter 6
Chapter 7
VHDL to Silicon Design Flow
Chapter 8

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

Test	Fault detected									
$w_1 w_2 w_3$	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		✓						✓		✓
001										
010										
011										
100										
101										
110										
111										

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

Test	Fault detected									
$w_1 w_2 w_3$	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		✓						✓		✓
001		✓		✓				✓		✓
010										
011										
100										
101										
110										
111										

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

Test $w_1 w_2 w_3$	Fault detected									
	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		√						√		√
001		√		√				√		√
010		√				√		√		√
011										
100										
101										
110										
111										

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

Test	Fault detected									
$w_1 w_2 w_3$	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		✓						✓		✓
001		✓		✓				✓		✓
010		✓				✓		✓		✓
011			✓		✓		✓		✓	
100										
101										
110										
111										

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

Test $w_1 w_2 w_3$	Fault detected									
	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		✓						✓		✓
001		✓		✓				✓		✓
010		✓				✓		✓		✓
011			✓		✓		✓		✓	
100	✓								✓	
101										
110										
111										

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

Test $w_1 w_2 w_3$	Fault detected									
	a/0	a/1	b/0	b/1	c/0	c/1	d/0	d/1	f/0	f/1
000		✓						✓		✓
001		✓		✓				✓		✓
010		✓				✓		✓		✓
011			✓		✓		✓		✓	
100	✓								✓	
101	✓								✓	
110										
111										

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

Test $w_1 w_2 w_3$	Fault detected									
	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		✓						✓		✓
001		✓		✓				✓		✓
010		✓				✓		✓		✓
011			✓		✓		✓		✓	
100	✓								✓	
101	✓								✓	
110	✓								✓	
111										

Fault Simulation

- 5 wires (a, b, c, d, f) can be either s_a_0 or s_a_1

Test	Fault detected									
$w_1 w_2 w_3$	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		√						√		√
001		√		√				√		√
010		√				√		√		√
011			√		√		√		√	
100	√								√	
101	√								√	
110	√								√	
111									√	

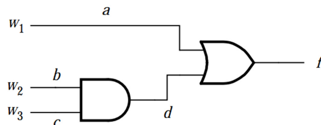
Fault Simulation

- There is only one pattern to detect b/1 ..”001”

Test $w_1 w_2 w_3$	Fault detected									
	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		✓						✓		✓
001		✓		✓				✓		✓
010		✓				✓		✓		✓
011			✓		✓		✓		✓	
100	✓								✓	
101	✓								✓	
110	✓								✓	
111									✓	

Fault Simulation

- There is only one pattern to detect b/1 .."001"



Test	Fault detected									
$w_1 w_2 w_3$	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		✓						✓		✓
001		✓		✓				✓		✓
010		✓				✓		✓		✓
011			✓		✓		✓		✓	
100	✓								✓	
101	✓								✓	
110	✓								✓	
111									✓	

Fault Simulation

- There is only one pattern to detect b/1 .."001"

Test $w_1 w_2 w_3$	Fault detected									
	$a/0$	$a/1$	$b/0$	$b/1$	$c/0$	$c/1$	$d/0$	$d/1$	$f/0$	$f/1$
000		✓						✓		✓
001		✓		✓				✓		✓
010		✓				✓		✓		✓
011			✓		✓		✓		✓	
100	✓								✓	
101	✓								✓	
110	✓								✓	
111									✓	

Fault Simulation

- There is only one pattern to detect b/1 .."001"

Test $w_1 w_2 w_3$	Fault detected									
	a/0		b/0	b/1	c/0	c/1	d/0		f/0	
000										
001				✓						
010						✓				
011			✓		✓		✓		✓	
100	✓								✓	
101	✓								✓	
110	✓								✓	
111									✓	

Fault Simulation

- There is only one pattern to detect b/1 ..”001”

Test $w_1 w_2 w_3$	Fault detected									
	$a/0$		$b/0$	$b/1$	$c/0$	$c/1$	$d/0$		$f/0$	
000										
001				✓						
010						✓				
011			✓		✓		✓		✓	
100	✓								✓	
101	✓								✓	
110	✓								✓	
111									✓	

Dr. Ashraf Eltholth

Chapter 5
Chapter 6
Chapter 7
VHDL to Silicon Design Flow
Chapter 8

Fault Simulation

- There is only one pattern to detect b/1 .."001"

Test $w_1 w_2 w_3$	Fault detected									
	a/0		b/0	b/1	c/0	c/1	d/0		f/0	
000										
001				✓						
010						✓				
011			✓		✓		✓		✓	
100	✓								✓	
101	✓								✓	
110	✓								✓	
111									✓	

Chapter 5

Chapter 6

Chapter 7

VHDL to Silicon Design Flow

Chapter 8

Fault Simulation

- There is only one pattern to detect b/1 ..”001”

Test $w_1 w_2 w_3$	Fault detected									
	$a/0$		$b/0$	$b/1$		$c/1$				
000										
001				√						
010						√				
011			√							
100	√									
101	√									
110	√									
111										

Chapter 5
Chapter 6
Chapter 7
VHDL to Silicon Design Flow
Chapter 8

Fault Simulation

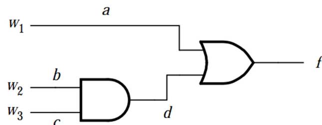
- There is only one pattern to detect b/1 ..”001”

w_1 a
w_2 b
w_3 c
d
f

Test $w_1 w_2 w_3$	Fault detected			
	$a/0$	$b/0$	$b/1$	$c/1$
000				
001			√	
010				√
011		√		
100	√			
101	√			
110	√			
111				

Fault Simulation

- There is only one pattern to detect b/1 ..”001”



Test $w_1 w_2 w_3$	Fault detected									
	a/0		b/0	b/1		c/1				
000										
001				✓						
010						✓				
011			✓							
100	✓									
101	✓									
110	✓									
111										

Chapter 5
Chapter 6
Chapter 7
VHDL to Silicon Design Flow
Chapter 8

Fault Simulation

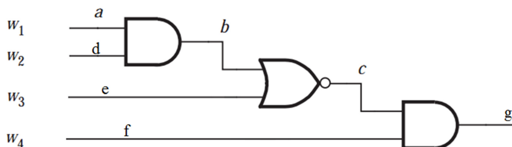
- There is only one pattern to detect b/1 .."001"

Minimal Test Pattern

- 001
- 010
- 011
- 100

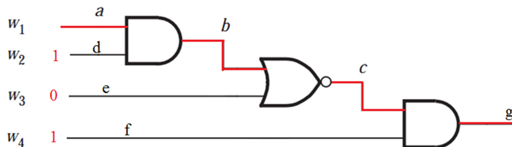
Test $w_1 w_2 w_3$	Fault detected $a/0$	$b/0$	$b/1$	$c/1$
000				
001			√	
010				√
011		√		
100	√			
101	√			
110	√			
111				

Path Sensitizing



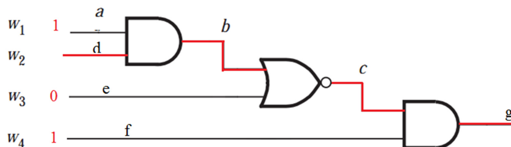
- The circuit contains 4 paths : abcg, dbcg, ecg, fg
- The path is activated by ensuring that other paths in the circuit do not determine the value of the output g.
- To sensitize a path through an input of an AND or NAND gate, all other inputs must be set to 1. To sensitize a path through an input of an OR or NOR gate, all other inputs must be 0.

Path Sensitizing



Path abcg			
w1	g	Faulty result will indicate	I/p
0	1	a/1, b/1, c/0, g/0	0101
1	0	a/0, b/0, c/1, g/1	1101

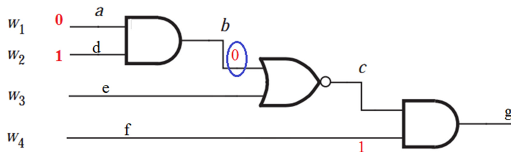
Path Sensitizing



Path abcg			
w1	g	Faulty result will indicate	I/p
0	1	a/1, b/1, c/0, g/0	0101
1	0	a/0, b/0, c/1, g/1	1101

Path dbcg			
w2	g	Faulty result will indicate	I/p
0	1	d/1, b/1, c/0, g/0	1001
1	0	d/0, b/0, c/1, g/1	1101

Path Sensitizing



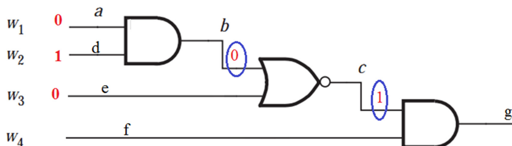
Path abcg			
w1	g	Faulty result will indicate	I/p
0	1	a/1, b/1, c/0, g/0	0101
1	0	a/0, b/0, c/1, g/1	1101

Path dbcg			
w2	g	Faulty result will indicate	I/p
0	1	d/1, b/1, c/0, g/0	1001
1	0	d/0, b/0, c/1, g/1	1101

Path ecg			
w3	g	Faulty result will indicate	I/p
0	1	e/1, c/0, g/0	0101
1	0	e/0, c/1, g/1	0111

Chapter 5
Chapter 6
Chapter 7
VHDL to Silicon Design Flow
Chapter 8

Path Sensitizing



Path abcg			
w1	g	Faulty result will indicate	I/p
0	1	a/1, b/1, c/0, g/0	0101
1	0	a/0, b/0, c/1, g/1	1101

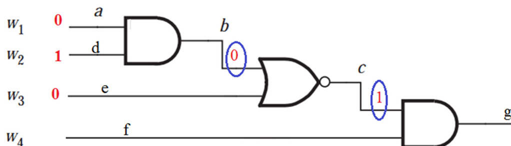
Path dbcg			
w2	g	Faulty result will indicate	I/p
0	1	d/1, b/1, c/0, g/0	1001
1	0	d/0, b/0, c/1, g/1	1101

Path ecg			
w3	g	Faulty result will indicate	I/p
0	1	e/1, c/0, g/0	0101
1	0	e/0, c/1, g/1	0111

Path fg			
w4	g	Faulty result will indicate	I/p
0	0	f/1, g/1	0100
1	1	f/0, g/0	0101

Chapter 5
Chapter 6
Chapter 7
VHDL to Silicon Design Flow
Chapter 8

Path Sensitizing



Path abcg			
w1	g	Faulty result will indicate	I/p
0	1	a/1, b/1, c/0, g/0	0101
1	0	a/0, b/0, c/1, g/1	1101

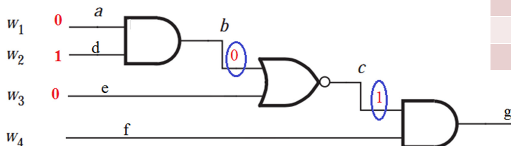
Path dbcg			
w2	g	Faulty result will indicate	I/p
0	1	d/1, b/1, c/0, g/0	1001
1	0	d/0, b/0, c/1, g/1	1101

Path ecg			
w3	g	Faulty result will indicate	I/p
0	1	e/1, c/0, g/0	0101
1	0	e/0, c/1, g/1	0111

Path fg			
w4	g	Faulty result will indicate	I/p
0	0	f/1, g/1	0100
1	1	f/0, g/0	0101

Path Sensitizing

Test Pattern
0100
0101
0111
1001
1101



Path abcg			
w1	g	Faulty result will indicate	I/p
0	1	a/1, b/1, c/0, g/0	0101
1	0	a/0, b/0, c/1, g/1	1101

Path dbcg			
w2	g	Faulty result will indicate	I/p
0	1	d/1, b/1, c/0, g/0	1001
1	0	d/0, b/0, c/1, g/1	1101

Path ecg			
w3	g	Faulty result will indicate	I/p
0	1	e/1, c/0, g/0	0101
1	0	e/0, c/1, g/1	0111

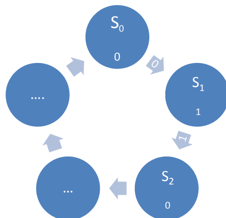
Path fg			
w4	g	Faulty result will indicate	I/p
0	0	f/1, g/1	0100
1	1	f/0, g/0	0101

Testing of Sequential Circuits

- A combinational circuit can be tested by comparing its behavior with the functionality specified in the **truth table**. An equivalent attempt would be to test a sequential circuit by comparing its behavior with the functionality specified in the **state table**.

- Represent the circuit by a state diagram.

- Verify each transition by first taking the machine to a specific state, applying the input to perform the transition and verify the output and the final state.

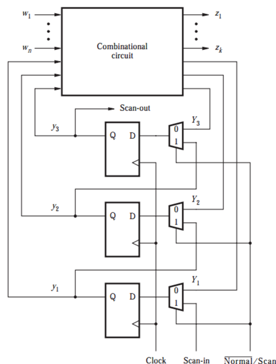


Design for Testability

Scan-Path Technique

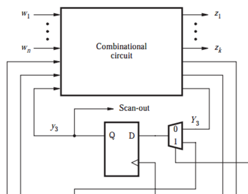
•The operation of the flip-flops is tested by scanning into them a pattern of 0s and 1s, for example, 01011001, in consecutive clock cycles, and observing whether the same pattern is scanned out.

•The combinational circuit is tested by applying test vectors on $w_1w_2 \cdots w_ny_1y_2y_3$ and observing the values generated on $z_1z_2 \cdots z_mY_1Y_2Y_3$.



Design for Testability

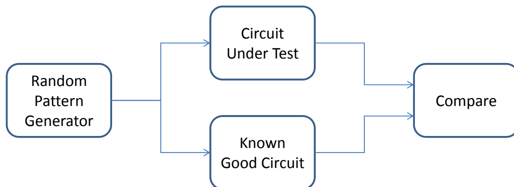
Scan-Path Technique



- The $y_1y_2y_3$ portion of the test vector is scanned into the flip-flops during three clock cycles, using $\overline{Normal}/Scan = 1$.
- The $w_1w_2 \cdots w_n$ portion of the test vector is applied as usual and the normal operation of the sequential circuit is performed for one clock cycle, by setting $\overline{Normal}/Scan = 0$. The outputs $z_1z_2 \cdots z_m$ are observed. The generated values of $Y_1Y_2Y_3$ are loaded into the flip-flops at this time.
- The select input is changed to $\overline{Normal}/Scan = 1$, and the contents of the flip-flops are scanned out during the next three clock cycles, which makes the

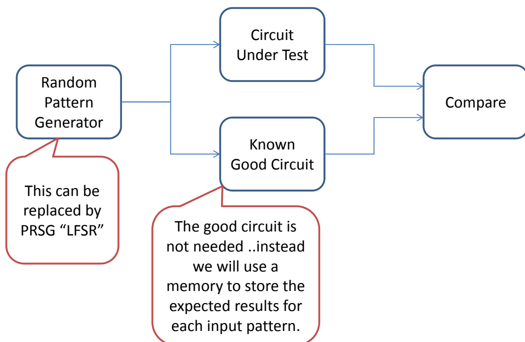
Testing of Sequential Circuits

Random Testing



Testing of Sequential Circuits

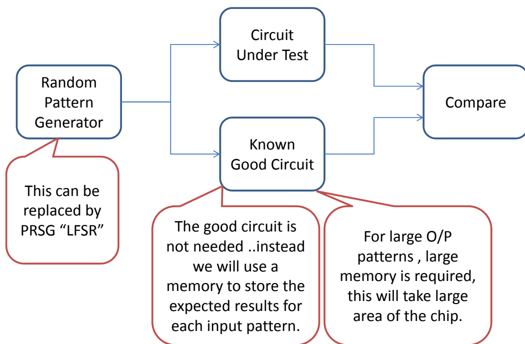
Random Testing



Chapter 5
Chapter 6
Chapter 7
VHDL to Silicon Design Flow
Chapter 8

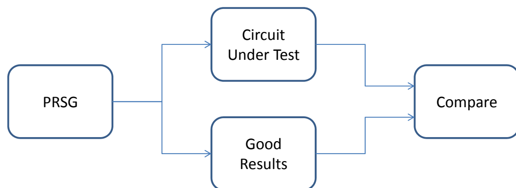
Testing of Sequential Circuits

Random Testing



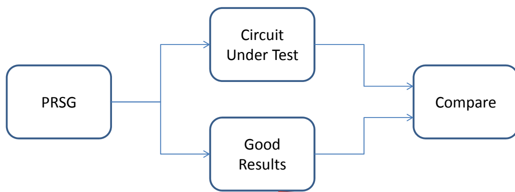
Testing of Sequential Circuits

Random Testing



Testing of Sequential Circuits

Random Testing



Transition count testing
-Count the number of transitions (between 0,1) of the o/p pattern for each i/p pattern
- store the number of transitions only
-For example "0010011101110" contains 6 transitions which can be stored as "110"
-This will same a lot of memory

Chapter 5

Chapter 6

Chapter 7

VHDL to Silicon Design Flow

Chapter 8