University Of Diyala
College Of Engineering
Computer Engineering Department

Modeling Sequential Logic in VHDL

Dr. Yasir Amer Abbas

Third stage

2017

In this lecture

- Synchronous sequential logic circuits rely on storage elements for their operations.
- Flips-flops (FFs) and latches are two commonly used one-bit elements.
- Others as registers, shift registers, and counters.
- Only synchronous sequential logic is considered.

LATCHES & FLIP-FLOPS

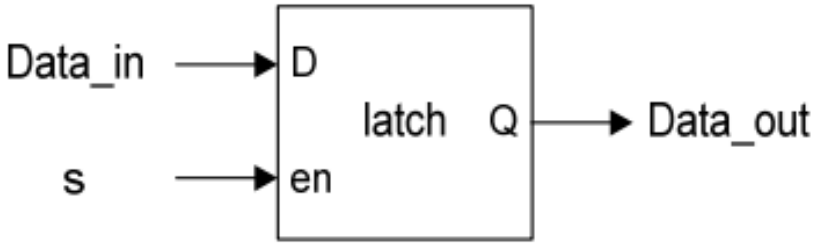
- A latch is a level-sensitive memory device (transparent).
  - As long as the pulse remains at the active high level, any changes in the data input will change the state of the latch.
- A flip-flop (FF) is an edge-triggered memory device.
  - An edge-triggered FF ignores the pulse while it is at a constant level (non-transparent).
  - Triggers only during a transition of the clock signal.
  - Could on the positive edge of the clock (posedge), or negative edge (negedge).

Simple Latch

D Latch

- A latch is inferred because the IF statement is incomplete.
- The notion of implied memory is instantiated in this case.

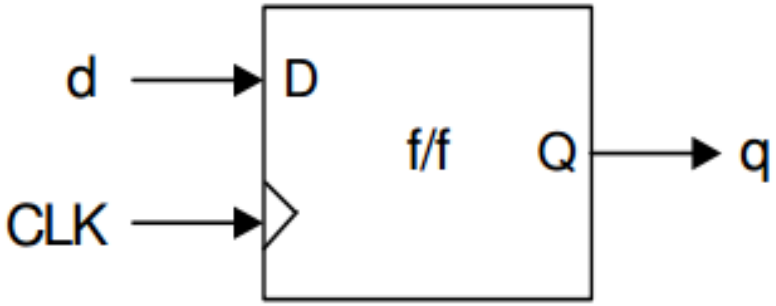
```
…
always  (S or Data_In)
if ( S )
Data_out = Data_In;
…..
```



Simple Latch

```
library ieee ;
use ieee.std_logic_1164.all;
entity D_latch is
 port(
            data_in: in std_logic;
             enable: in std_logic;
            data_out: out std_logic );
end D_latch;
 architecture behv of D_latch is
 begin -- compare this to D flipflop
            process(data_in, enable)
                        begin
                        if (enable='1') then        -- no clock signal here
                         data_out <= data_in;
                        end if;
            end process;
end behv;
```

Basic positive-edge triggered D flip-flop



FLIP-FLOP IS THE BASIC COMPONENT

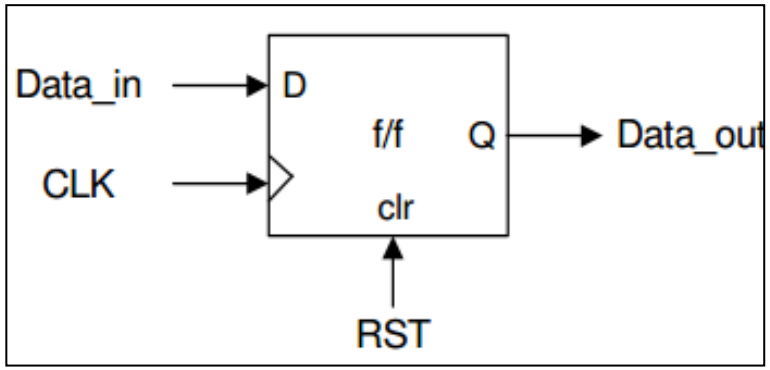
```
library ieee ;
use ieee.std_logic_1164.all;

entity dff is
port(
        data_in: in std_logic;
        clock: in std_logic;
        data_out: out std_logic );
end dff;
Architecture behv of dff is
        begin
        process(data_in, clock) .
        begin                   -- clock rising edge
        if (clock='1' and clock'event) then
        data_out <= data_in;
        end if;
end process;
end behv;
```

VHDL CODE FOR RISING EDGE D FLIP FLOP

```
Library IEEE;
USE IEEE.Std_logic_1164.all;

entity RisingEdge_DFlipFlop is
port(
        Q : out std_logic;
        Clk :in std_logic;
        D :in std_logic );

end RisingEdge_DFlipFlop;

Architecture Behavioral of RisingEdge_DFlipFlop is
begin
        process(Clk)
                begin
                if(rising_edge(Clk)) then
                Q <= D;
                end if;
        end process;
 end Behavioral;
```

VHDL CODE FOR RISING EDGE D FLIP-FLOP WITH SYNCHRONOUS RESET

```
Library IEEE;
USE IEEE.Std_logic_1164.all;
entity RisingEdge_DFlipFlop_SyncReset is
port(
        Q : out std_logic;
        Clk :in std_logic;
        sync_reset: in std_logic;
        D :in std_logic );
end RisingEdge_DFlipFlop_SyncReset;

Architecture Behavioral of RisingEdge_DFlipFlop_SyncReset is
 begin
        process(Clk)
                begin
                    if(rising_edge(Clk)) then
                        if(sync_reset='1') then
                            Q <= '0';
                        else Q <= D;
                        end if;
                    end if;
        end process;
end Behavioral;
```



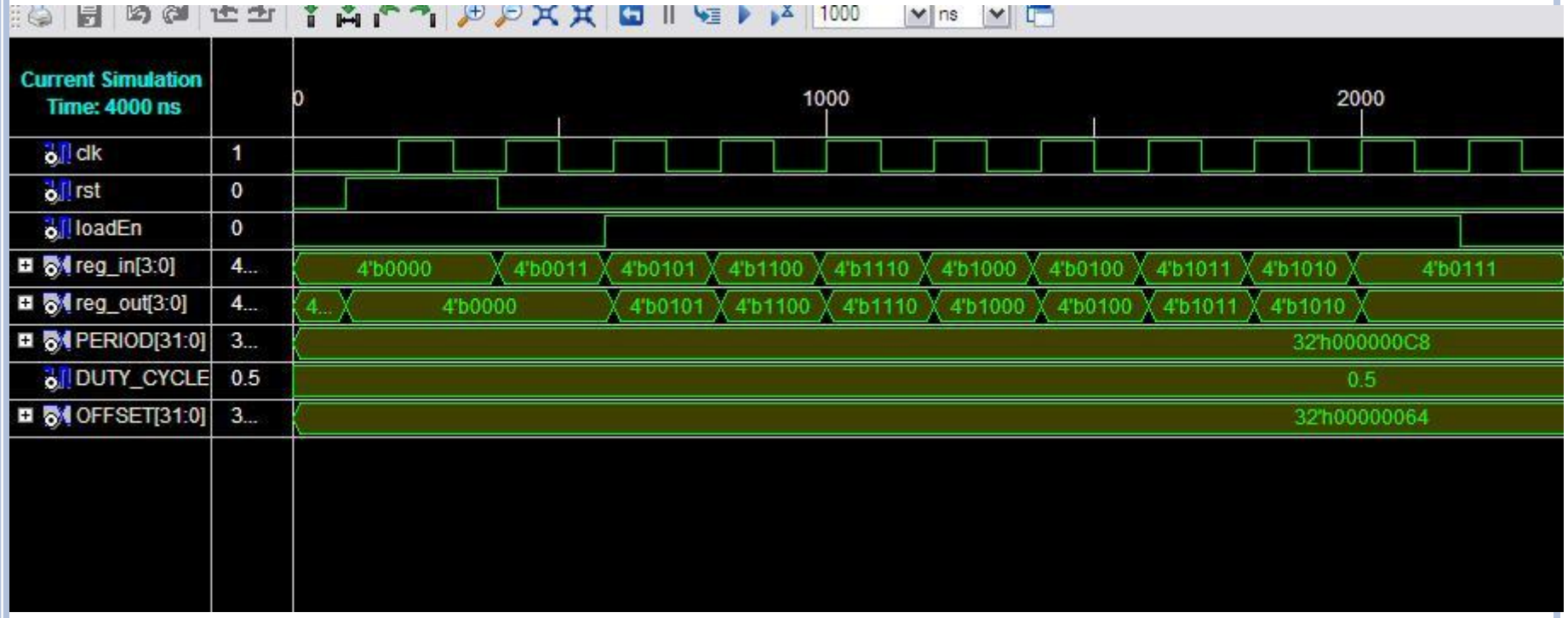
VHDL code for Rising Edge D Flip-Flop with Asynchronous Reset High Level

```
Library IEEE;
USE IEEE.Std_logic_1164.all;

 entity RisingEdge_DFlipFlop_AsyncResetHigh is
 port(
            Q : out std_logic;
            Clk :in std_logic;
            Async_reset: in std_logic;
            D :in std_logic );
end RisingEdge_DFlipFlop_AsyncResetHigh;

Architecture Behavioral of RisingEdge_DFlipFlop_AsyncResetHigh is
begin
process(Clk, Async_reset) .
            begin
                        if(Async_reset='1') then Q <= '0';
                           elsif(rising_edge(Clk)) then
                            Q <= D;
                        end if;
            end process;
end Behavioral;
```

N-BIT REGISTER

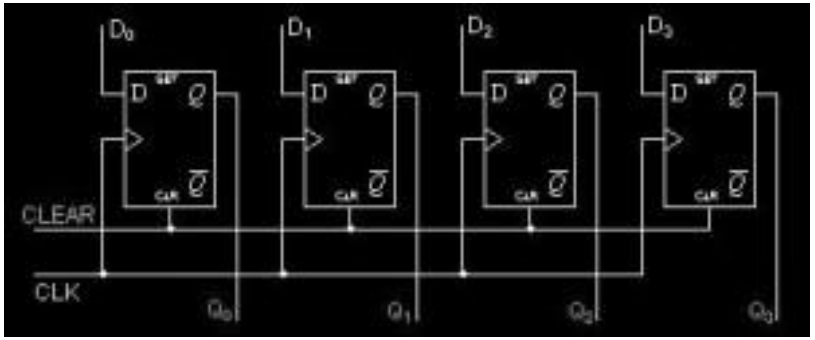
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ckt_reg is
                generic (pattern : natural := 4);
Port ( clk : in STD_LOGIC;
                rst : in STD_LOGIC;
                loadEn : in STD_LOGIC;
                reg _in : in STD_LOGIC_VECTOR (regCount-1 downto 0);
                reg _out : out STD_LOGIC_VECTOR (regCount-1 downto 0));
end ckt_reg ;

architecture Behavioral of ckt_reg is
begin
                process (clk, rst)
                                begin
                                if rst = '1' then
                                reg _out <= '0';
                                elsif clk'event and clk = '1' then
                                if loadEn = '1' then
                                reg _out <= reg_in;
                                end if;
                                end if;
                end process;

end Behavioral;
```

Simulation N bit register

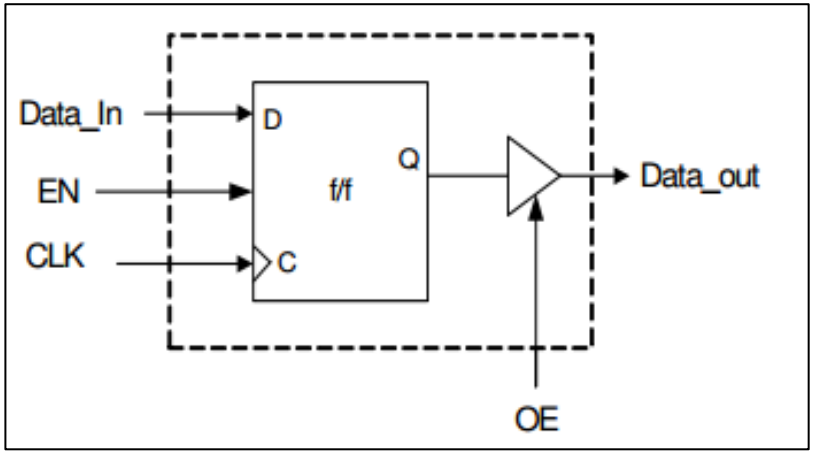


Parallel In – Parallel Out Shift Registers

```
library ieee;
use ieee.std_logic_1164.all;

entity pipo is
 port(
                clk : in std_logic;
                D: in std_logic_vector(3 downto 0);
                Q: out std_logic_vector(3 downto 0)
                 );
                end pipo;

architecture arch of pipo is
begin
 process (clk)
                 begin
                 if (CLK'event and CLK='1') then
                 Q <= D;
                 end if;
 end process;

end arch;
```

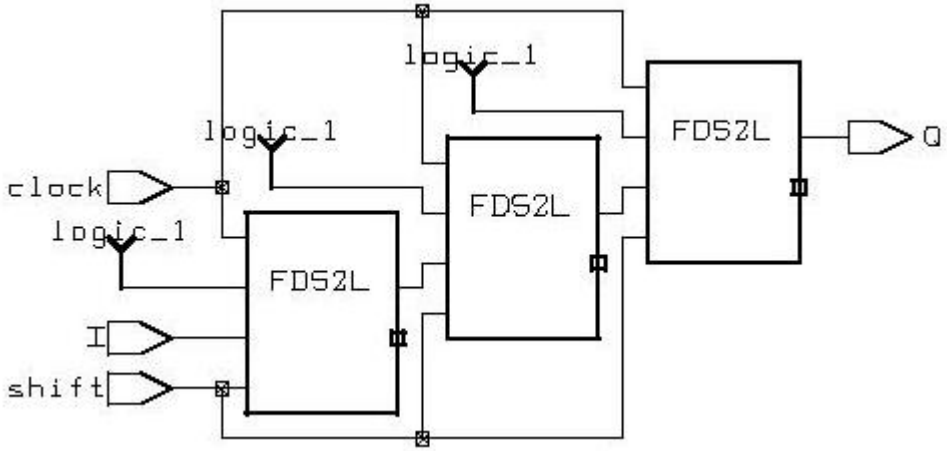


H.W: DESIGN FLIP-FLOP WITH A TRI-STATE OUTPUT USING VHDL CODE

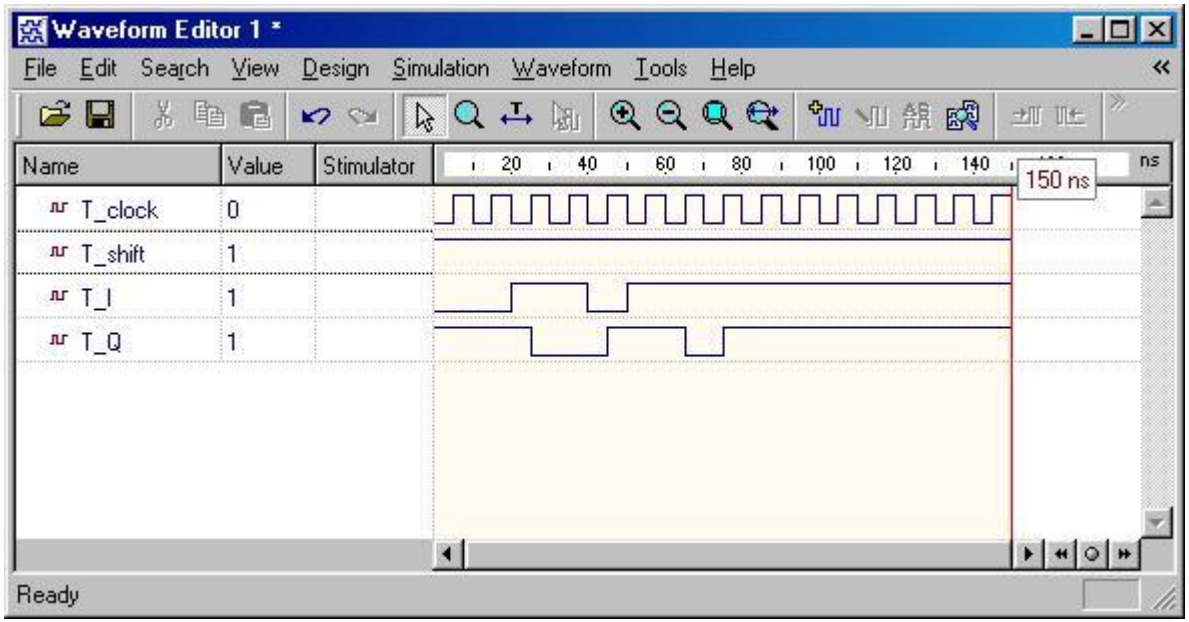


SHIFT REGISTER

```
library ieee ;
 use ieee.std_logic_1164.all;
 ---------------------------------------------
entity shift_reg is
                port(
                                I: in std_logic;
                                clock: in std_logic;
                                 shift: in std_logic;
                                Q: out std_logic );
                end shift_reg;
---------------------------------------------
Architecture behv of shift_reg is -- initialize the declared signal
                signal S: std_logic_vector(2 downto 0):="111";
                                begin
                                process(I, clock, shift, S)
                                begin          -- everything happens upon the clock changing
                                if clock'event and clock='1' then
                                  if shift = '1' then
                                    S <= I & S(2 downto 1);
                                      end if;
                                    end if;
                    end process; -- concurrent assignment
              Q <= S(0);
 end behv;
```



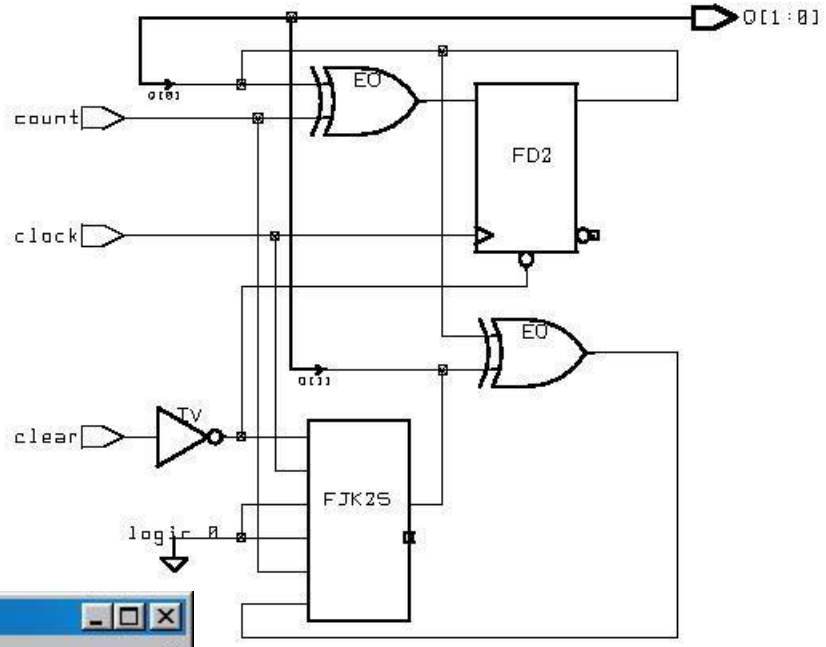
Simulation Waveform Shift Register

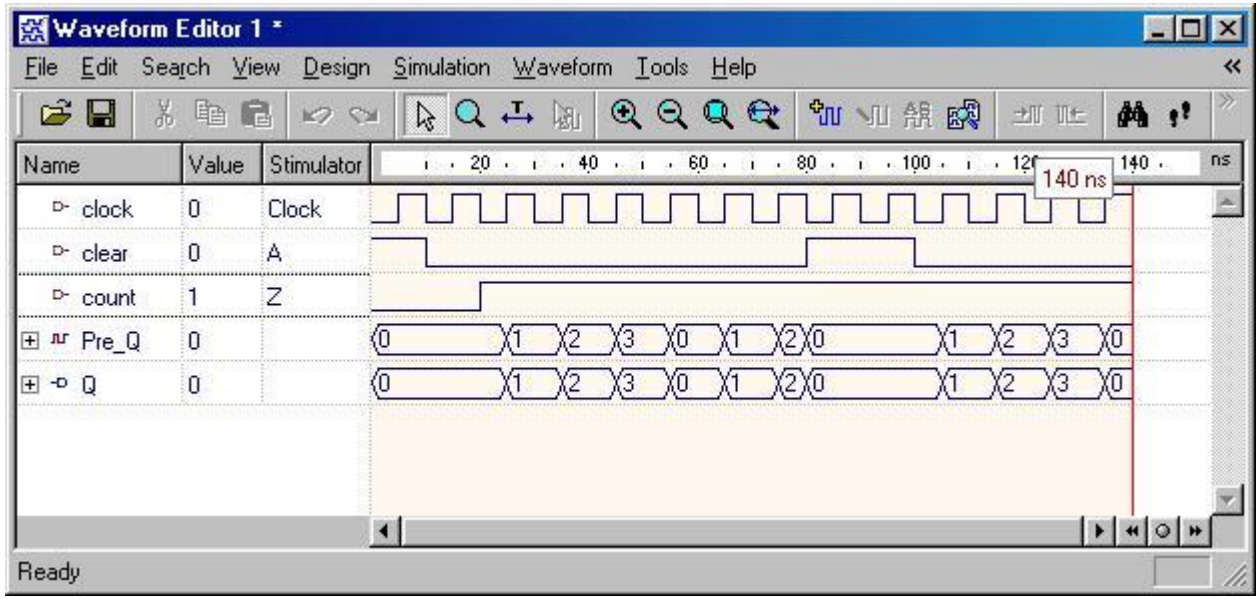


COUNTER

```
library ieee ;
use ieee.std_logic_1164.all; use ieee.std_logic_unsigned.all;
---------------------------------------------------
entity counter is
generic(n: natural :=2);
 port(
clock: in std_logic;
 clear: in std_logic;
 count: in std_logic;
 Q: out std_logic_vector(n-1 downto 0) );
 end counter;
---------------------------------------------------
architecture behv of counter is
signal Pre_Q: std_logic_vector(n-1 downto 0);
begin -- behavior describe the counter
process(clock, count, clear)
begin
if clear = '1' then Pre_Q <= Pre_Q - Pre_Q;
elsif (clock='1' and clock'event) then
if count = '1' then Pre_Q <= Pre_Q + 1;
 end if;
end if;
 end process; -- concurrent assignment statement
Q <= Pre_Q;
end behv;
```

COUNTER


O[1:0]
count
clock
clear
logic 0
EO
FD2
EO
TV
FJK2S


Waveform Editor 1 *
File Edit Search View Design Simulation Waveform Tools Help
Name
Value
Stimulator
20
40
60
80
100
120
140
ns
140 ns
clock 0 Clock
clear 0 A
count 1 Z
Pre_Q 0
Q 0
Ready

Finite State Machine (FSM)

· A circuit type with memory.
· Usually as datapath controller unit.
· Via algorithmic state machine (ASM) flowchart, an FSM is readily modeled in HDL.
· There are two basic models of FSMs: Moore and Mealy.
· In a Mealy machine, the next state (NS) and the outputs depend on both the present state (PS) and the inputs.
· The NS of a Moore machine depends on the PS and the inputs, but the outputs depend on only the PS.
· All FSMs have the general feedback structure.
· We will deal only with synchronous FSMs, hence the state transitions of the machine are synchronized by the active edge of a common clock.
In this case, the state register is consisted of edge triggered flip-flops.