Lecture 8

Introduction to VHDL: Synthesis (Chapter 2 + more)

Khaza Anuarul Hoque
ECE 4250/7250

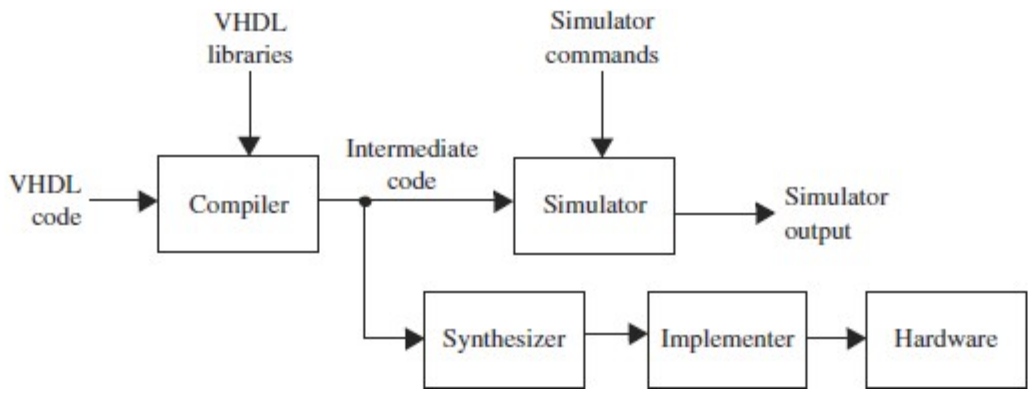
Compilation, Simulation, and Synthesis of VHDL Code (continued)

- **Analysis (compilation)**: checks source code; if a syntax or semantic error occurs, then the compiler gives an error message. Checks references to libraries.
- **Elaboration**: converts code for simulator use; creates drivers for signals, creates ports for component instances, allocates memory.
- **Simulation**: consists of an initialization phase and actual simulation. The initialization phase is used to give an initial value to the signal. During simulation, the VHDL statements are executed and corresponding actions are scheduled (scheduling transactions).

Compilation, Simulation, and Synthesis of VHDL Code (continued)

- Simulation and synthesis process:

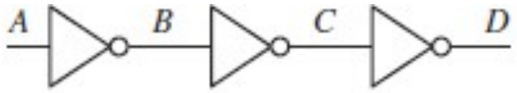
FIGURE 2-28: Compilation, Simulation, and Synthesis of VHDL Code



Compilation, Simulation, and Synthesis of VHDL Code (continued)

- **Discrete event simulation**: time simulated in discrete steps
- **Event**: when a signal value changes
- **Delta delays** ensure signals are processed in the correct order

FIGURE 2-25: Illustration of Delta Delays during Simulation of Concurrent Statements



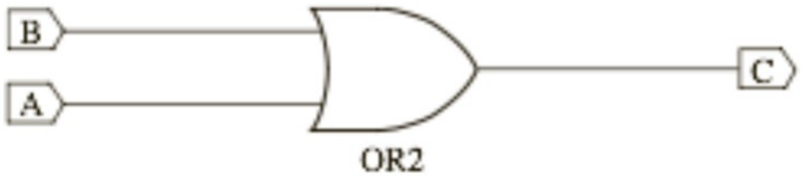
```
1 B <= not A;
2 C <= not B;
3 D <= not C after 5 ns;
```

ns	delta	A	B	C	D
0	+0	0	1	0	1
3	+0	1	1	0	1
3	+1	1	0	0	1
3	+2	1	0	1	1
8	+0	1	0	1	0

Simple Synthesis Examples

```
entity Q1 is
  port(A, B: in bit;
       C: out bit);
end Q1;

architecture circuit of Q1 is
begin
  process(A)
  begin
    C <= A or B after 5 ns;
  end process;
end circuit;
```



Simple Synthesis Examples (continued)

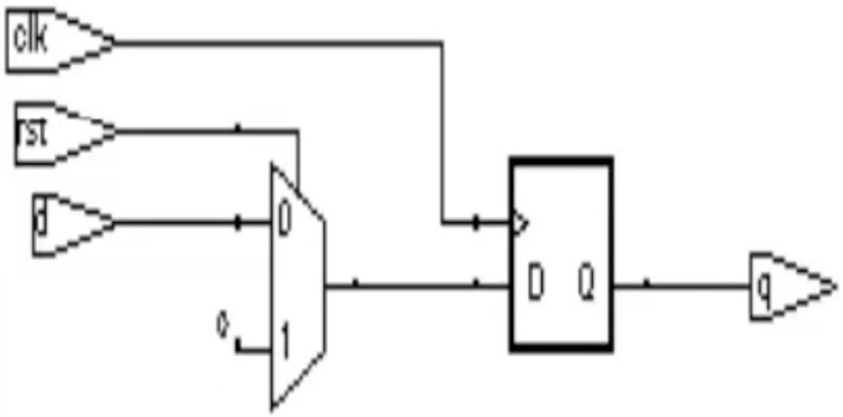
```
entity no_syn is
  port(A,B, CLK: in bit;
         D: out bit);
end no_syn;

architecture no_synthesis of no_syn is
  signal C: bit;
begin
  process(Clk)
  begin
    if (Clk='1' and Clk'event) then
      C <= A and B;
    end if;
  end process;
end no_synthesis;
```

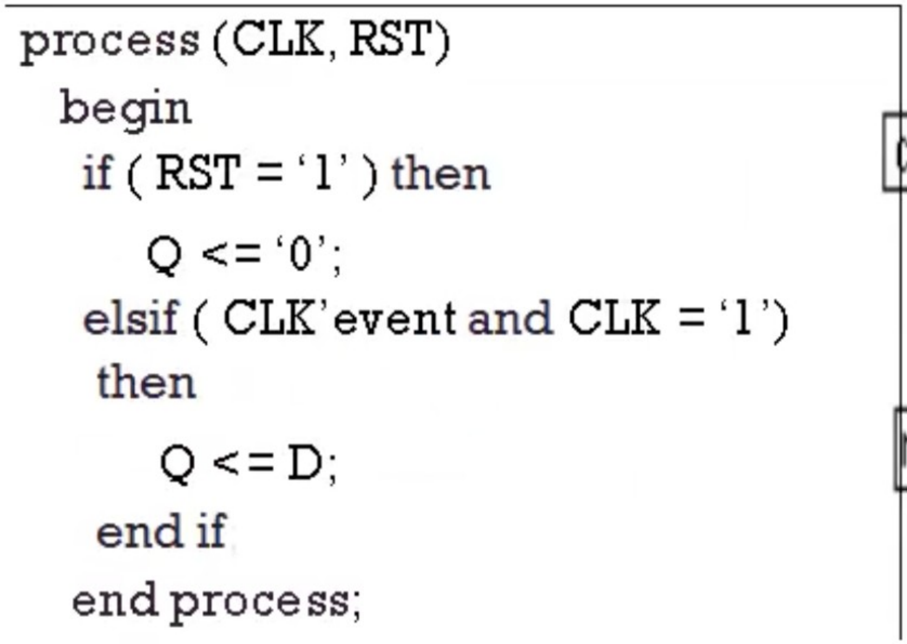
✗

Synchronous reset

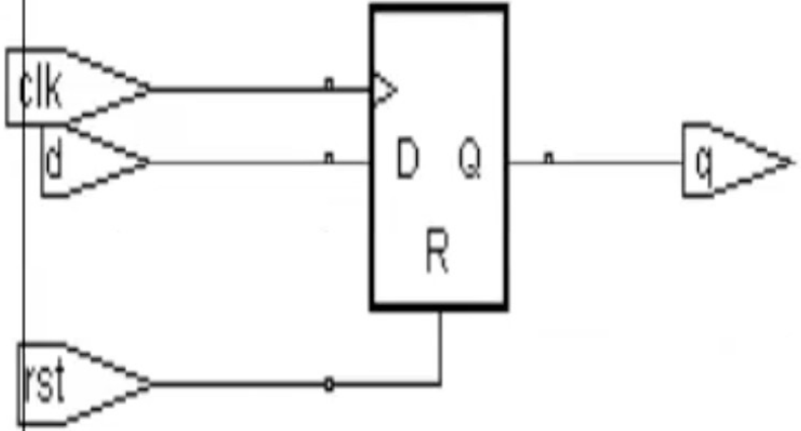
```
process (CLK)
  begin
    if ( CLK' event and CLK = '1')
then
      if ( RST = '1' ) then
          Q <= '0';
       else
          Q <= D;
       end if;
      end if;
 end process;
```



Asynchronous reset



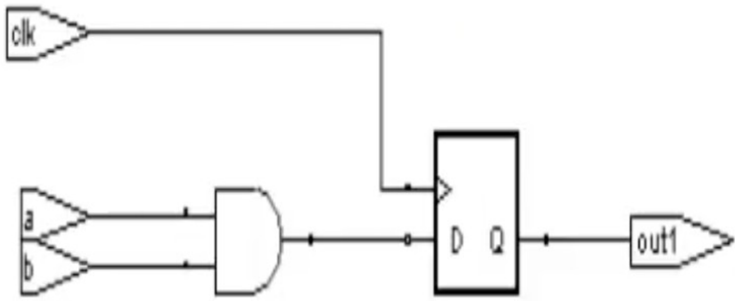
```
process (CLK, RST)
  begin
   if ( RST = '1' ) then
       Q <= '0';
   elsif ( CLK'event and CLK = '1')
    then
        Q <= D;
    end if
  end process;
```



Assignments under a clocked event

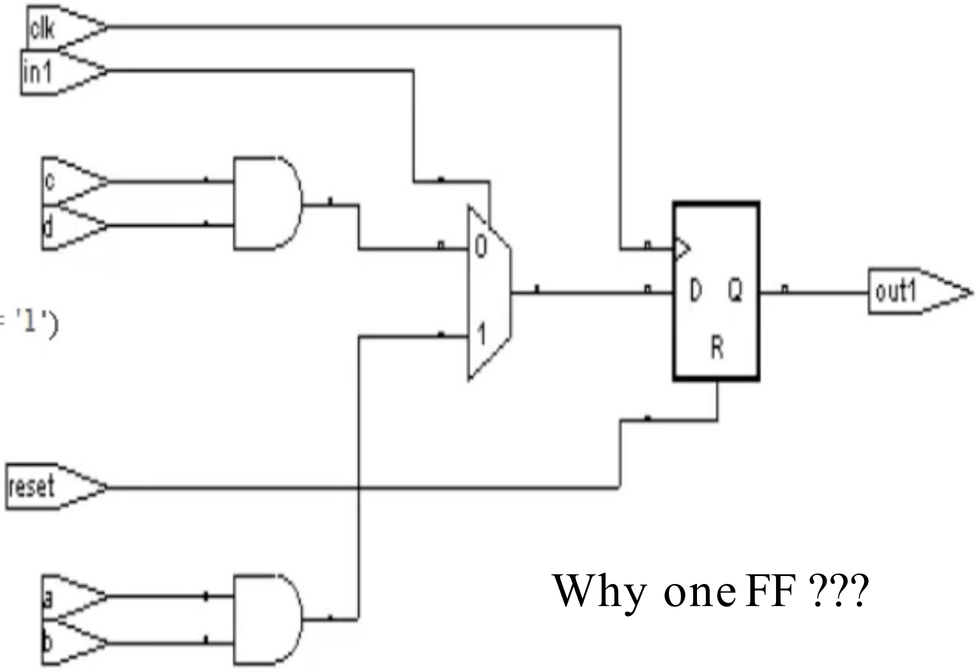
- Any assignment within clock statement will
  - generate a Flip-flop and
  - all other combinational circuitry will be created at the 'D' input of the Flip-flop.

```
process (clk)
 begin
    if (clk'event and clk = '1')
    then
       out1 <= a and b;
    end if;
 end process;
```



Hardware Example

```
process (clk, reset)
 begin
  if (reset = '1' ) then
    out1 <= '0';
  elsif (clk'event and clk = '1')
  then
    if (in1 = '1' ) then
      out1 <= a and b;
    else
      out1 <= c and d;
    end if;
  end if;
end process;
```



Why one FF ???

Hardware Example - latch

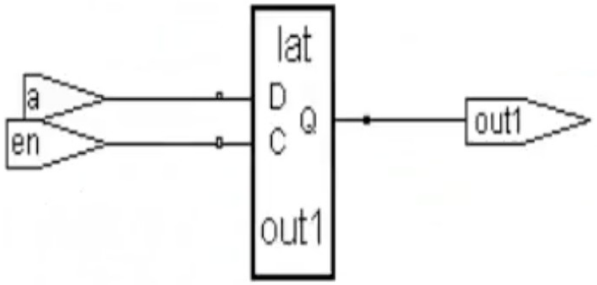
- Incompletely specified Conditional expression infers a latch
- Latch is a combinational circuit which necessarily has feedback to hold the output to previous value for the unspecified states/conditions.
- Avoid the inference of latches in synchronous designs. As latches infer feedback and they cause difficulties in timing analysis and test insertion applications. Most synthesizers provide warnings when latches are inferred.

Hardware Example - latch

<u>incompletely specified</u>

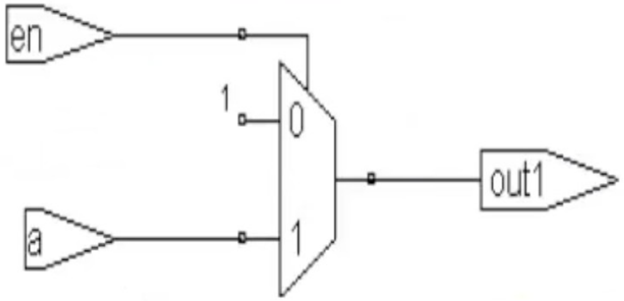
<u>Conditional expression.</u>

```
process (en,a)
    begin
      if en='1' then
         out1 <= a;
      end if
    end process;
```

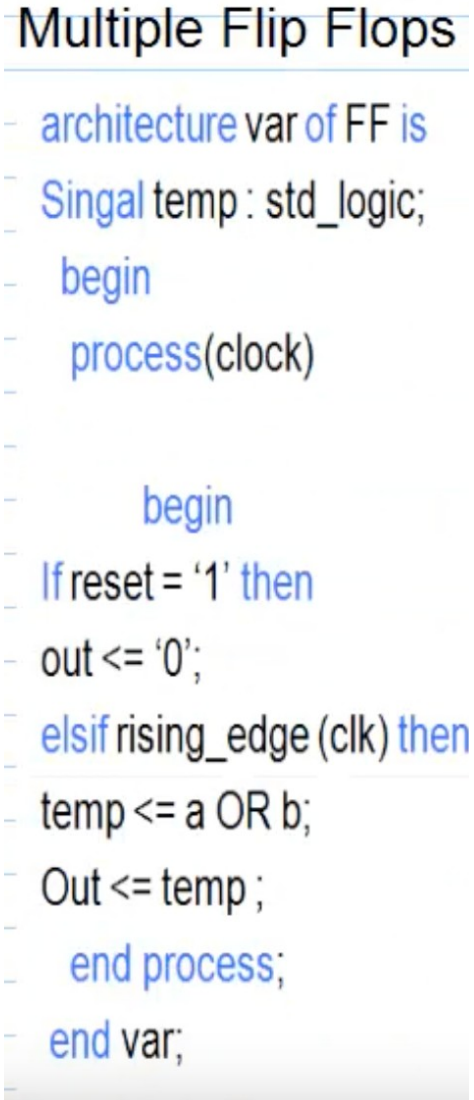


<u>Completely specified</u>
<u>Conditional expression.</u>

```
process (en,a)
     begin
       if en='1'  then
      out1 <= a;
       else
      out1 <='1';
        end if
      end process
```



Hardware Example – Multiple FFs

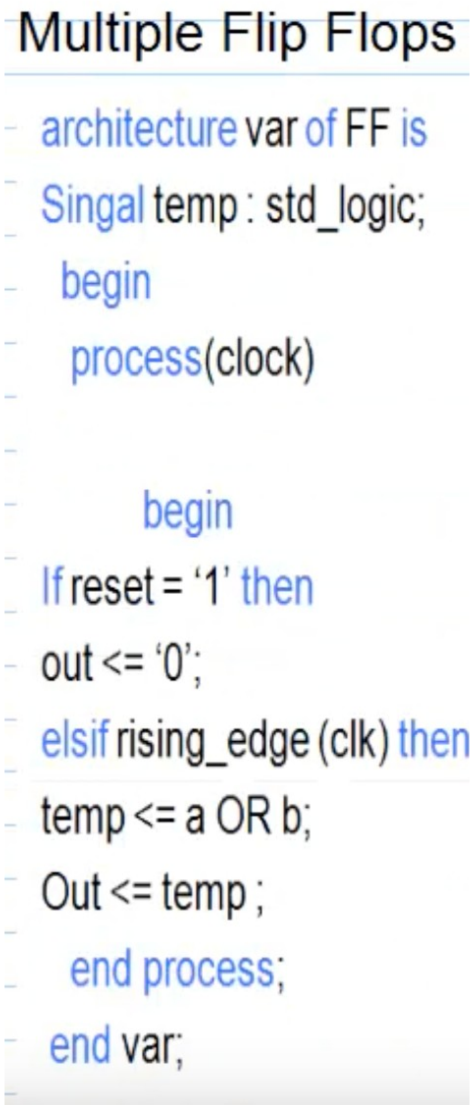

Multiple Flip Flops

```
architecture var of FF is
Singal temp : std_logic;
  begin
   process(clock)

          begin
If reset = '1' then
out <= '0';
elsif rising_edge (clk) then
temp <= a OR b;
Out <= temp ;
   end process;
 end var;
```

?

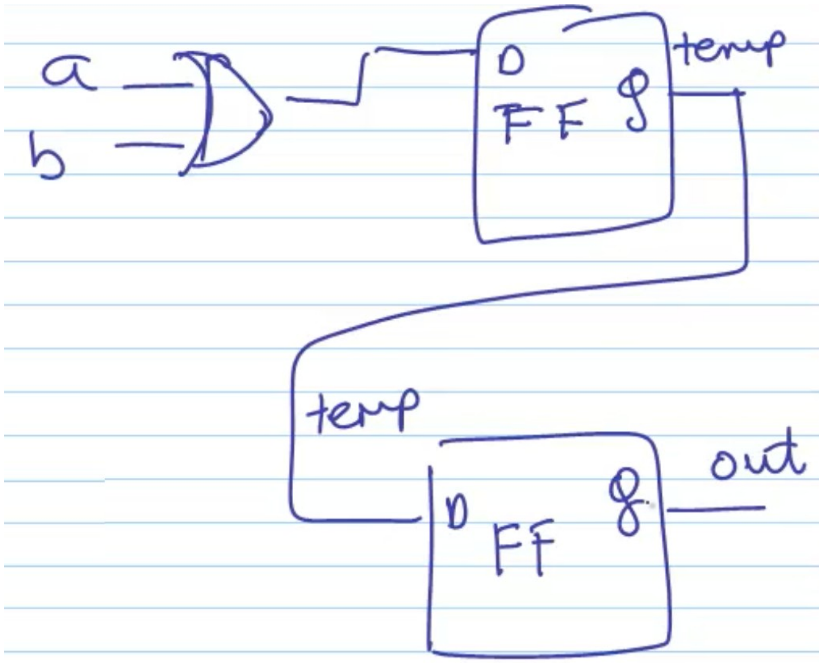
Hardware Example – Multiple FFs



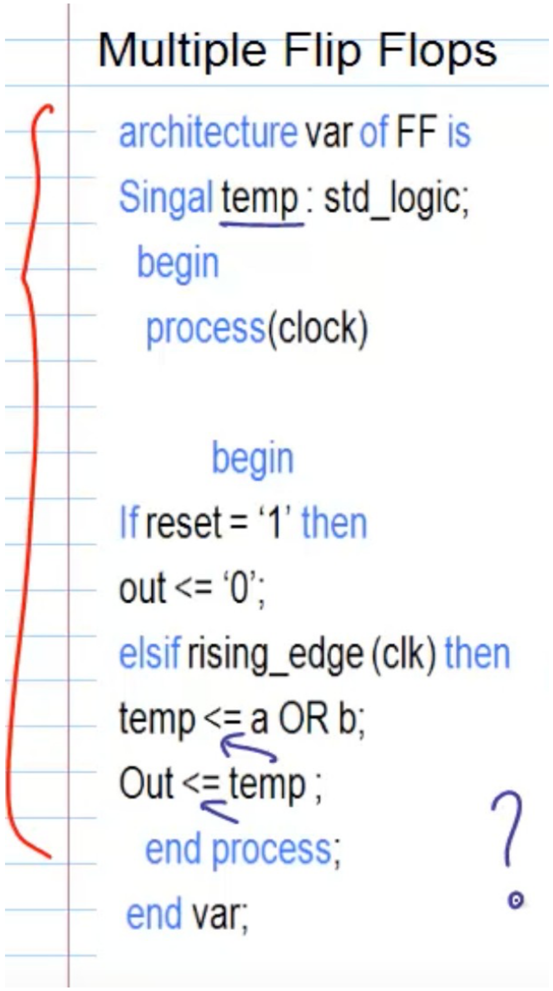
Multiple Flip Flops

```
architecture var of FF is
Singal temp : std_logic;
 begin
  process(clock)


       begin
If reset = '1' then
out <= '0';
elsif rising_edge (clk) then
temp <= a OR b;
Out <= temp ;
  end process;
 end var;
```



Hardware Example – Multiple FFs

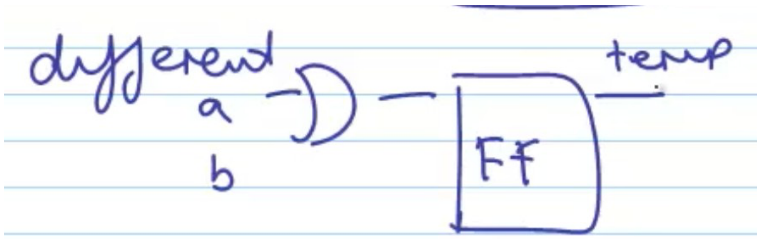


Multiple Flip Flops

```
architecture var of FF is
Singal temp : std_logic;
  begin
   process(clock)

        begin
If reset = '1' then
out <= '0';
elsif rising_edge (clk) then
temp <= a OR b;
Out <= temp ;
   end process;
 end var;
```

?



?

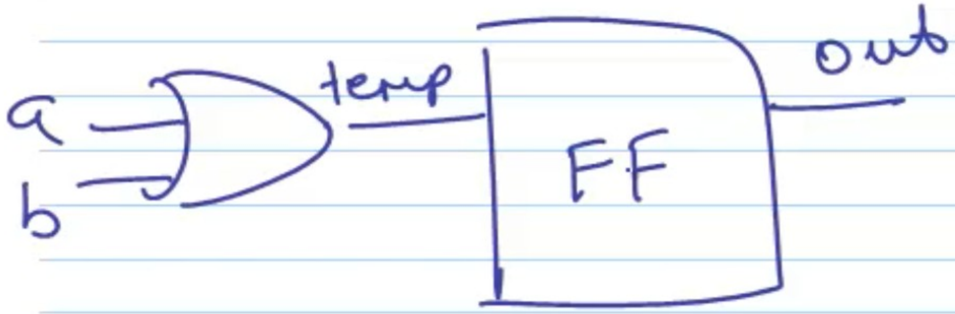
Hardware Example – Multiple FFs

```
architecture var of FF is
 begin
  process(clock)
     variable temp : std_logic;
         begin
         temp := '0';
If reset = '1' then
out <= '0';
elsif rising_edge (clk) then
temp := a OR b;
Out <= temp ;
  end process;
 end var;
```

temp as a variable

temp := a OR b

out ⇐ temp;


a
b
temp
FF
out