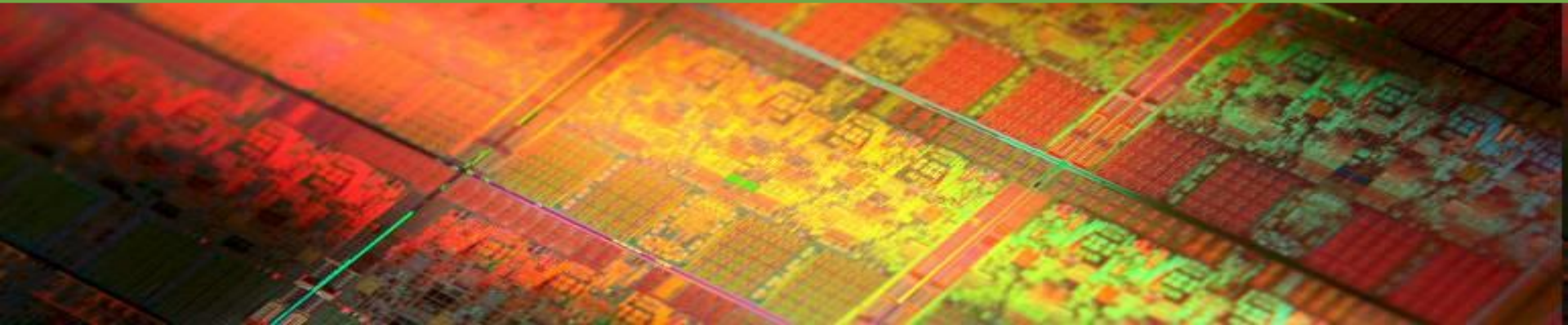


EE 2000

Logic Circuit Design (Fall 2018)



Lecture 3: VHDL Process & Register

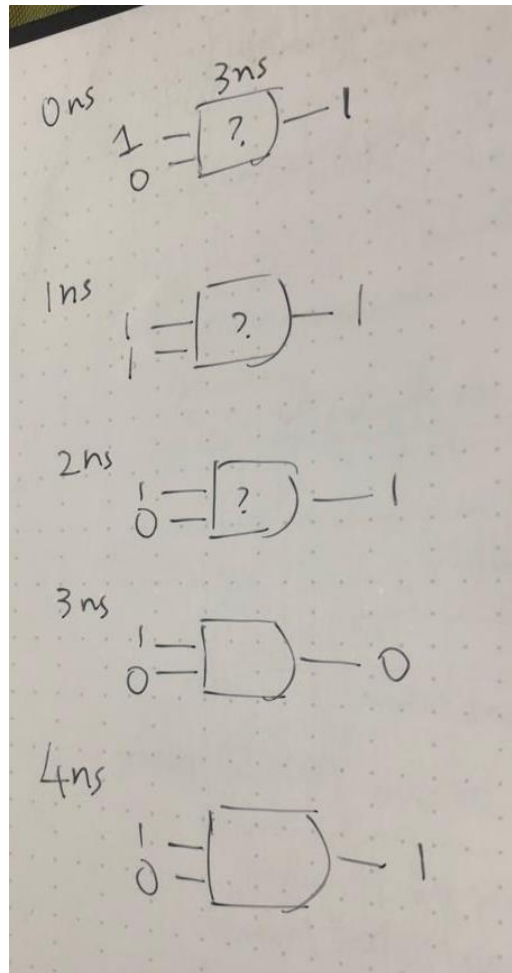
Instructor: Dr. Ray C.C. Cheung

Outline

- Introduction
- VHDL Process
- VHDL MUX
- VHDL Register & Counter
- VHDL Synthesis
- More VHDL Examples
- Summary

Gate Delay

- What is the meaning of 3ns gate delay?



Using Generic

- Useful for bit-width definition

Signal X ... (N-1 downto 0)

Y (N-1 downto 0)

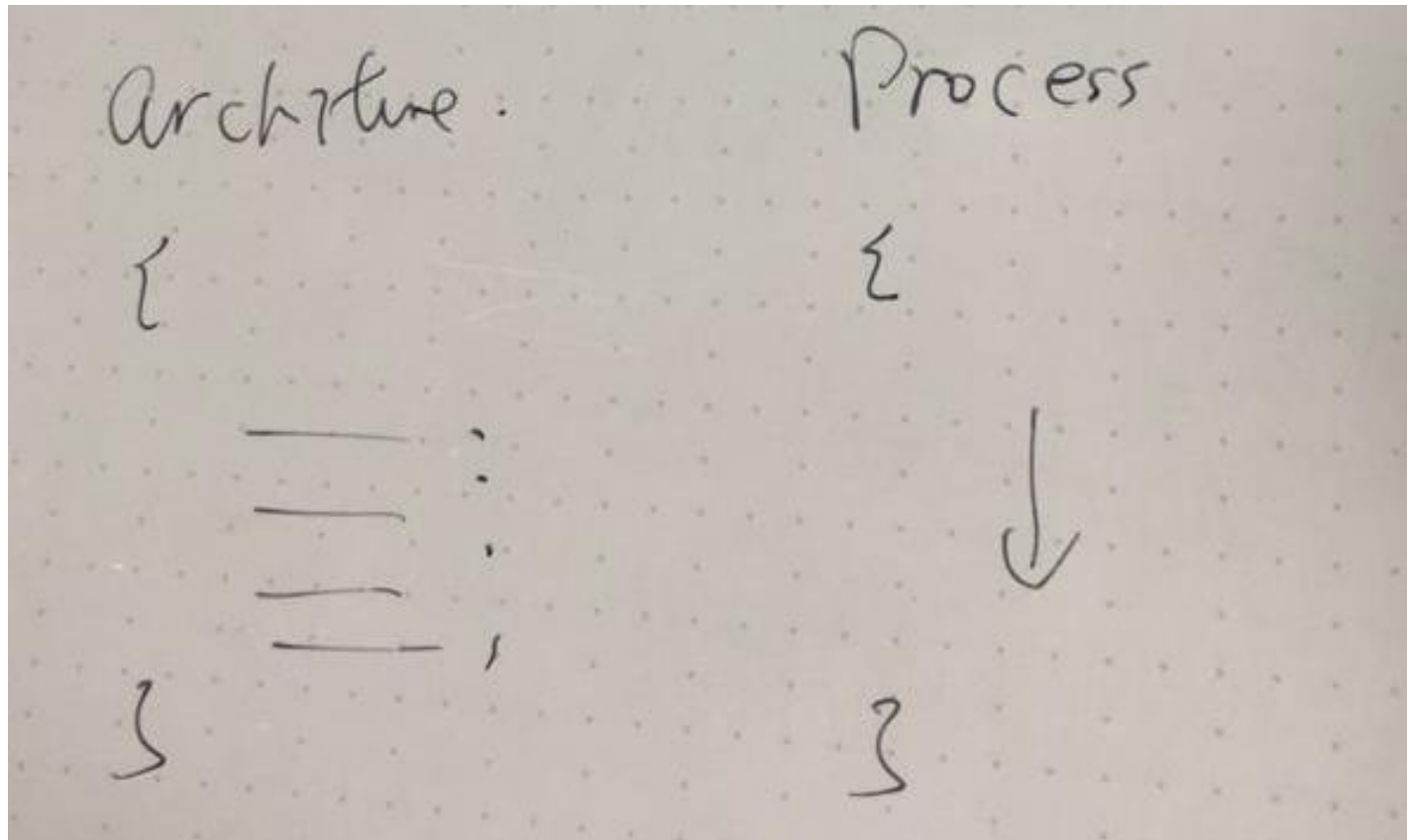
Z ...

$\vdots$ (N-1 ... 0)

Generic $N = 32;$

Architecture vs. Process

- Concurrent vs. Sequential



Introduction

- The process is the key structure in behavioral VHDL modeling
- General Rules:
 - Statements within a process are executed sequentially
 - All processes in a VHDL description file (.VHD) are executed concurrently
 - Does not matter how many process are there
- Signal assignment will be evaluated when events occur on the signals inside the process' sensitivity list(a, b, c)

VHDL Process

- Basic form:
 - process(sensitivity-list)
 - begin
 - Sequential-statements
 - end process;
- Example
 - process(A, B, C, D)
 - begin
 - C <= A and B;
 - E <= C or D;
 - end process;

HDL Adders

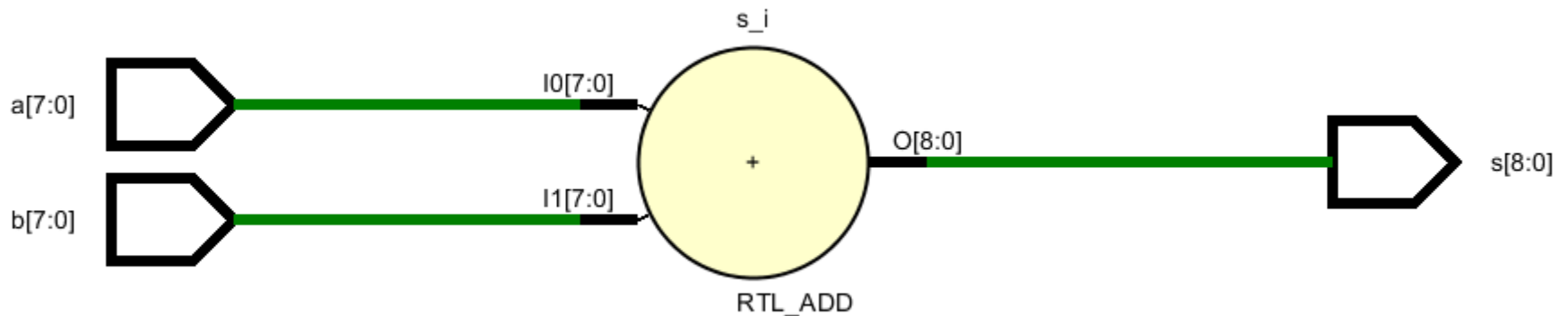
- 8-bit parallel adder circuit using HDL languages.
 - Will add 8-bit values A[8..1] and B[8..1] to produce the 9-bit sum.

- VHDL

```
34 entity Adder is
35     port (
36         a    : in integer range 0 to 255;
37         b    : in integer range 0 to 255;
38         s    : out integer range 0 to 511
39     );
40 end Adder;
41
42 architecture Behavioral of Adder is
43
44     begin
45
46         s <= a + b;
47
48 end Behavioral;
```

Schematic

- How many bits are there for the inputs, output?
- [7:0] and [8:0]
- Can you set the output as [7:0]?



In-Class Exercise

- Please work on the testbench for this Adder design
- You have 5-mins to complete this task.

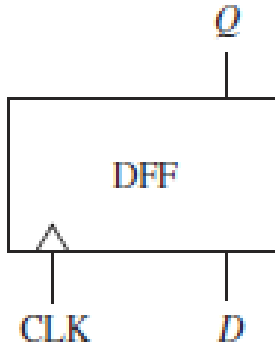
```
34 entity Adder is
35     port (
36         a    : in integer range 0 to 255;
37         b    : in integer range 0 to 255;
38         s    : out integer range 0 to 511
39     );
40 end Adder;
41
42 architecture Behavioral of Adder is
43
44     begin
45
46         s <= a + b;
47
48     end Behavioral;
```

Process Statement

- Asynchronous and concurrent
- Can contain simple signal assignments and sequential statements, but not conditional or selected signal statements
- Optional label
- First line is the **sensitivity list**
- Ports and signals are not changed until instructions inside have been performed

Modeling Flip-Flops

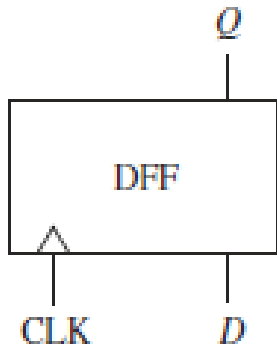
- A flip-flop can change state either on the rising or on the falling edge of the clock input
- This is modeled by a process
- Note what is inside the sensitivity list
- Example, D-type FF:



```
process(CLK)
begin
    if CLK'event and CLK = '1' -- rising edge of CLK
    then Q <= D;
    end if;
end process;
```

Choose your Sensitivity List

- D is not on the sensitivity list
- Changing D will not cause the FF to change its state



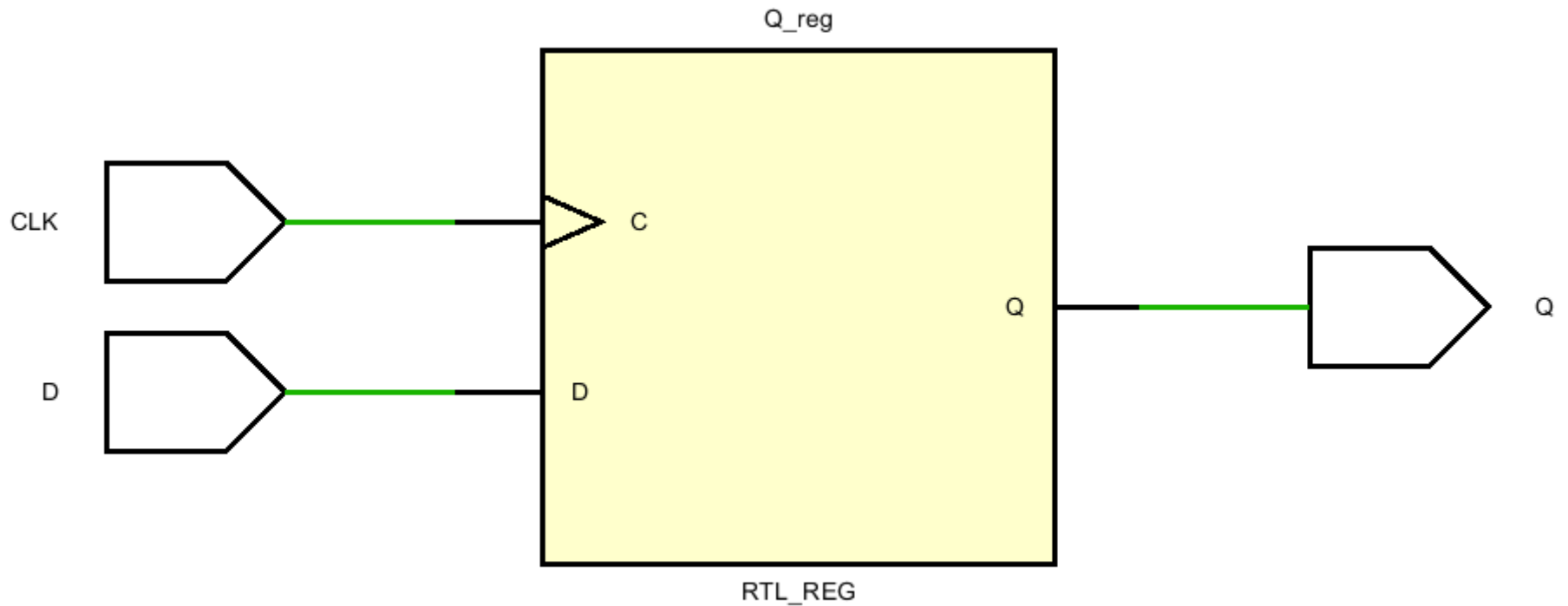
```
process(CLK)
begin
    if CLK'event and CLK = '1' -- rising edge of CLK
    then Q <= D;
    end if;
end process;
```

Full VHDL Code

```
25 entity DFF is
26   -- Port ( );
27   Port (
28       CLK: in std_logic;
29       D: in std_logic;
30       Q: out std_logic
31   );
32
33 end DFF;
34
35 architecture Behavioral of DFF is
36
37   begin
38
39       process (CLK)
40       begin
41           if CLK'event and CLK = '1'
42               then Q <= D;
43           end if;
44       end process;
45
46 end Behavioral;
```

Schematic

- Where is the CLK signal?



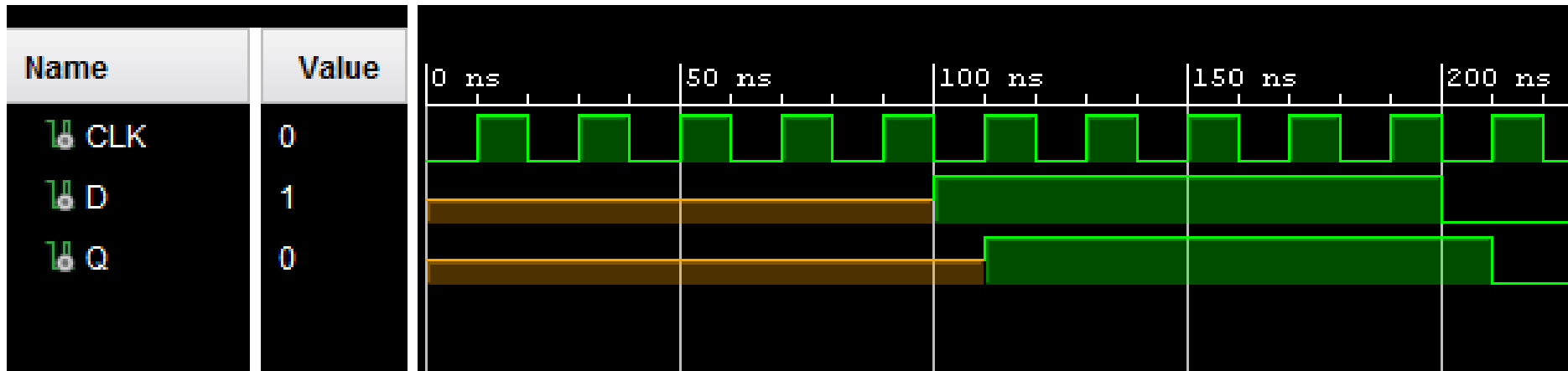
TestBench

- How to generate the CLK?
- Check the port map
- What is the expected waveform?

```
34 entity DFF_tb is
35   -- Port ( );
36 end DFF_tb;
37
38 architecture Behavioral of DFF_tb is
39   component DFF
40     Port (
41       CLK: in std_logic;
42       D: in std_logic;
43       Q: out std_logic);
44   end component;
45   signal CLK, D, Q: std_logic;
46 begin
47   uut: DFF port map (CLK, D, Q);
48
49   clkgen: process
50   begin
51     clk <= '0';
52     wait for 10 ns;
53     clk <= '1';
54     wait for 10 ns;
55   end process;
56
57   sim: process
58   begin
59     wait for 100 ns;
60     D <= '1';
61     wait for 100 ns;
62     D <= '0';
63   end process;
64 end Behavioral;
```

Waveform

- Initial State - 'U'
- Check the rising clock edge



Clk Generation

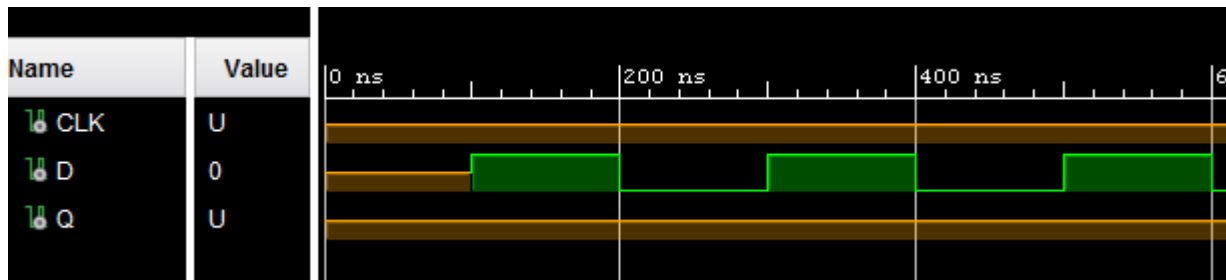
- Remember to provide an initial value to “CLK” signal.
- Do you prefer this shorter version for the clk?

```
38 architecture Behavioral of DFF_tb is
39     component DFF
40         Port (
41             CLK: in std_logic;
42             D: in std_logic;
43             Q: out std_logic);
44     end component;
45     signal CLK: std_logic := '0';
46     signal D, Q: std_logic;
47 begin
48     uut: DFF port map (CLK, D, Q);
49
50     -- clkgen: process
51     -- begin
52     --     clk <= '0';
53     --     wait for 10 ns;
54     --     clk <= '1';
55     --     wait for 10 ns;
56     -- end process;
57
58     clk <= not clk after 10 ns;
59
60     sim: process
61     begin
62         wait for 100 ns;
63         D <= '1';
64         wait for 100 ns;
65         D <= '0';
66     end process;
67 end Behavioral;
```

Incorrect Version

- CLK signal is not initialized

```
38 architecture Behavioral of DFF_tb is
39     component DFF
40     Port (
41         CLK: in std_logic;
42         D: in std_logic;
43         Q: out std_logic);
44     end component;
45     signal CLK, D, Q: std_logic;
46 begin
47     uut: DFF port map (CLK, D, Q);
```



Process Statement

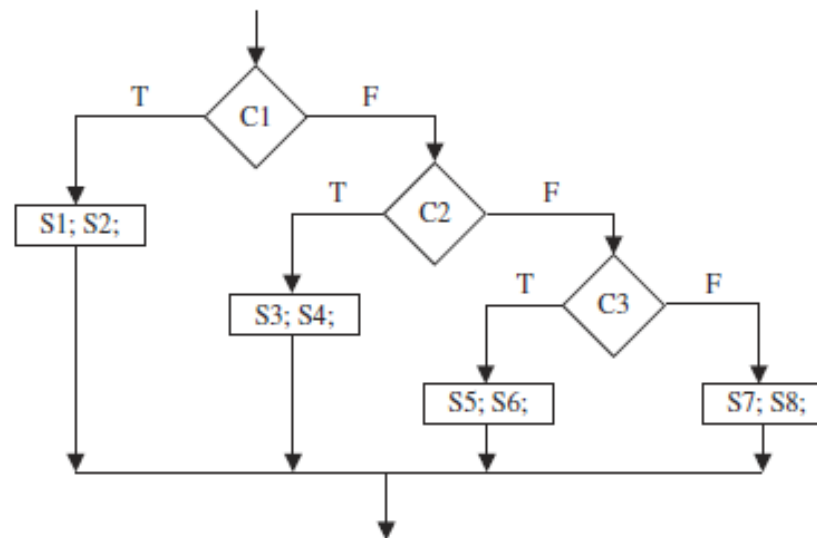
- IF statement
 - Must be inside a process statement
 - One of the most used statements in a process
 - Label is optional and has restrictions
 - Some anomalies occur when used to assign ports
- 'EVENT Attribute
 - Allows us to sense moment in time that an input or buffer port changes state
 - Must always be ANDed with test to determine in what logic state the port ends
 - When used with other level conditions, statement with 'EVENT must be last

Instantiating Entities

- First, write class entity
- Second, declare prototype of class entity in main entity's architecture using component declaration
- Finally, do actual instantiation using port map statement with or without generic map statement

Modeling Flip-Flops

- Flowchart depiction using nested Ifs and Else Ifs:

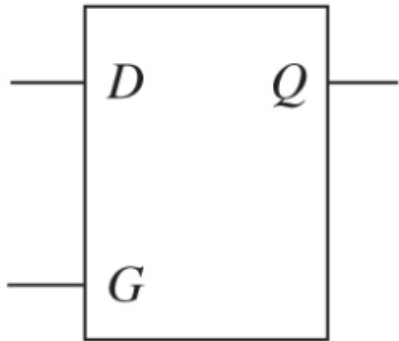


```
if (C1) then S1; S2;  
  else if (C2) then S3; S4;  
    else if (C3) then S5; S6;  
      end if;  
    end if;  
  end if;  
end if;
```

```
if (C1) then S1; S2;  
  elsif (C2) then S3; S4;  
  elsif (C3) then S5; S6;  
  else S7; S8;  
end if;
```

VHDL for a Latch

- Which signals are on the sensitivity list
- How do you define the signal G?
- If statement can be written on one single line



```
process(G, D)
begin
    if G = '1' then Q <= D; end if;
end process;
```

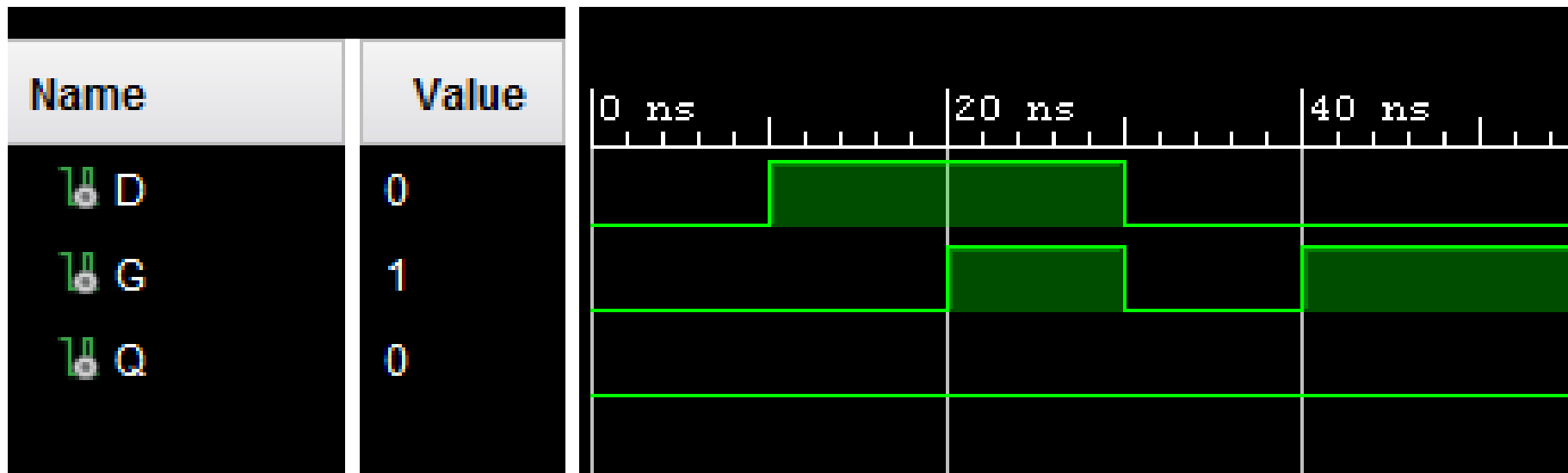
Latch TestBench - Debug

- What is the problem here?

```
34 entity Latch_tb is
35   -- Port ( );
36 end Latch_tb;
37
38 architecture Behavioral of Latch_tb is
39   component Latch
40     port (
41       D, G: in std_logic;
42       Q: out std_logic
43     );
44   end component;
45   signal D, G, Q: std_logic := '0';
46 begin
47   process
48     begin
49     wait for 10 ns;
50     D <= '1';
51
52     wait for 10 ns;
53     G <= '1';
54
55     wait for 10 ns;
56     D <= '0';
57     G <= '0';
58
59     wait for 10 ns;
60     G <= '1';
61
62     wait;
63   end process;
64 end Behavioral;
```

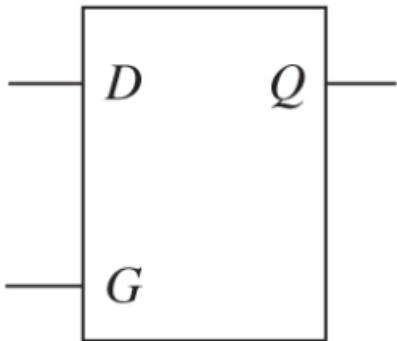
Waveform?

- Is this what you expected?



Sensitivity List

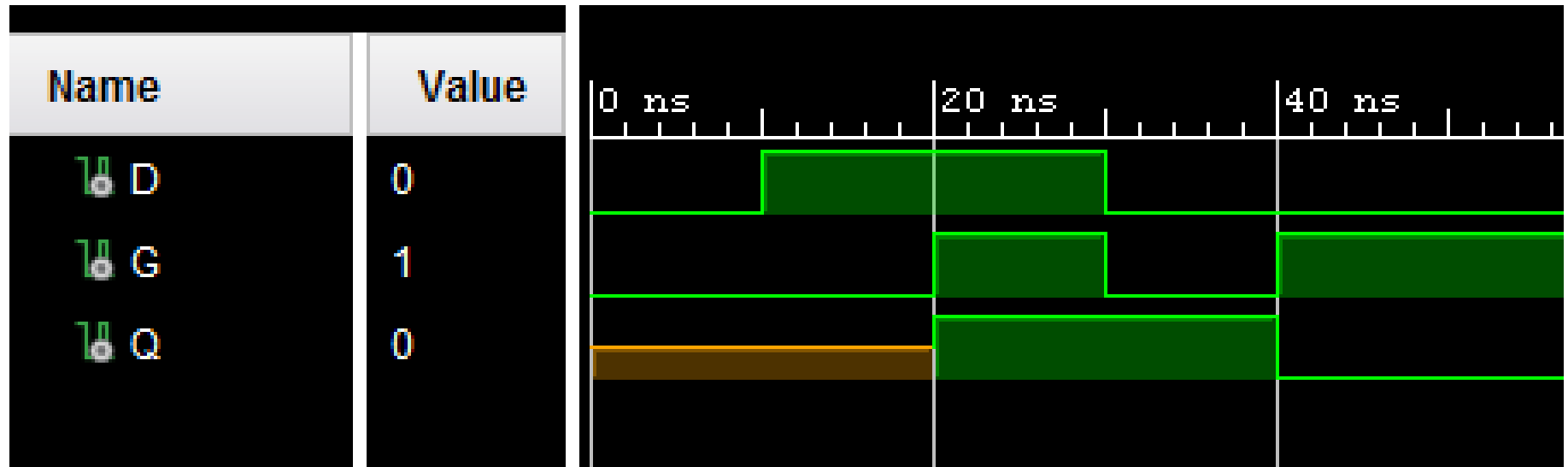
- If $G = '1'$, a change in D causes Q to change
- If G changes to $'0'$, Q remains unchanged



```
process(G, D)
begin
    if G = '1' then Q <= D; end if;
end process;
```

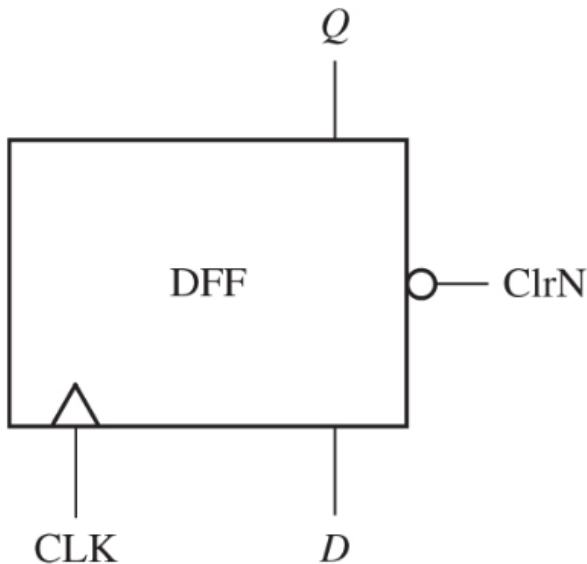
Correct Latch

- When $G = '1'$, D will be latched to Q



D Flip-Flop with Asynchronous Clear

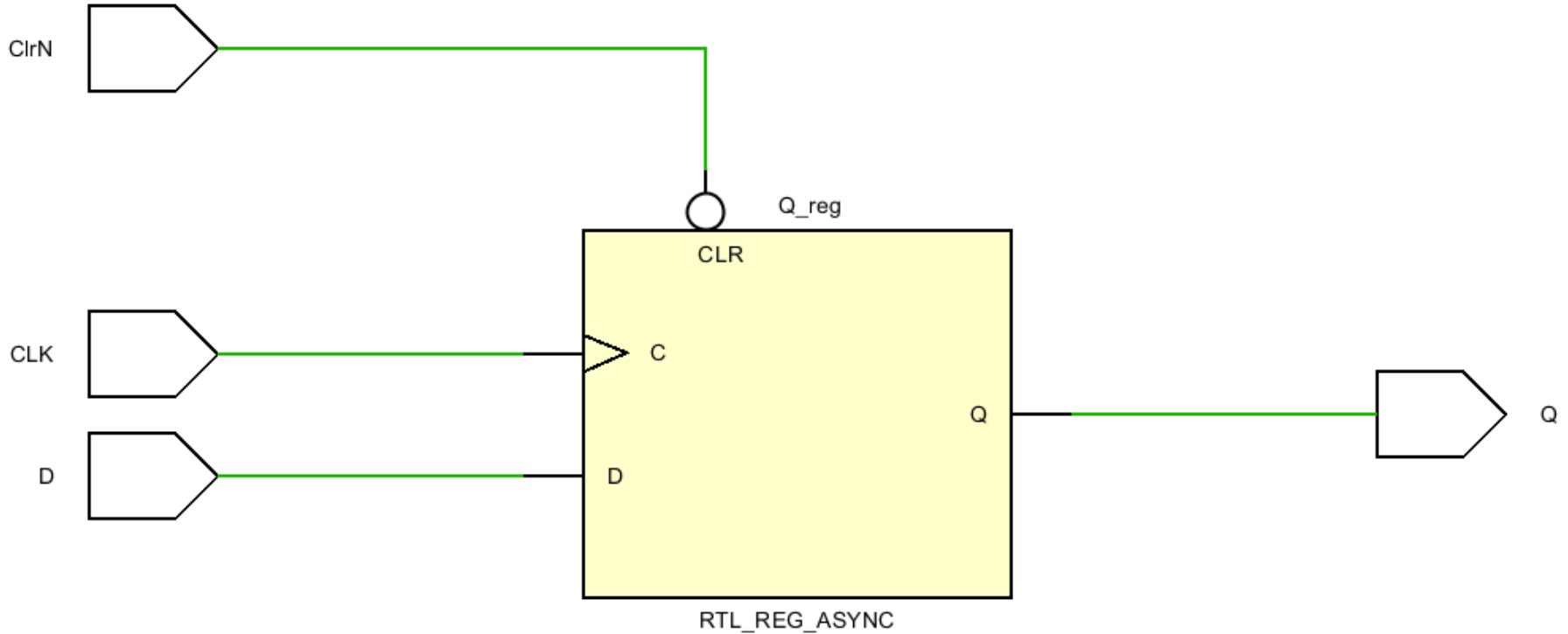
- Clear signal is active-high or active-low?
- How many if statements?



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

Schematic

- Asynchronous Clear - Clear at anytime



Full VHDL

- Double
“end if”

```
34 entity DFF_AC is
35     -- Port ( );
36     port (
37         CLK, D, ClrN: in std_logic;
38         Q: out std_logic
39     );
40 end DFF_AC;
41
42 architecture Behavioral of DFF_AC is
43 begin
44     process (CLK, ClrN)
45     begin
46         if CLRn = '0' then Q <= '0';
47         else if CLK'event and CLK = '1'
48             then Q <= D;
49         end if;
50     end if;
51     end process;
52 end Behavioral;
```

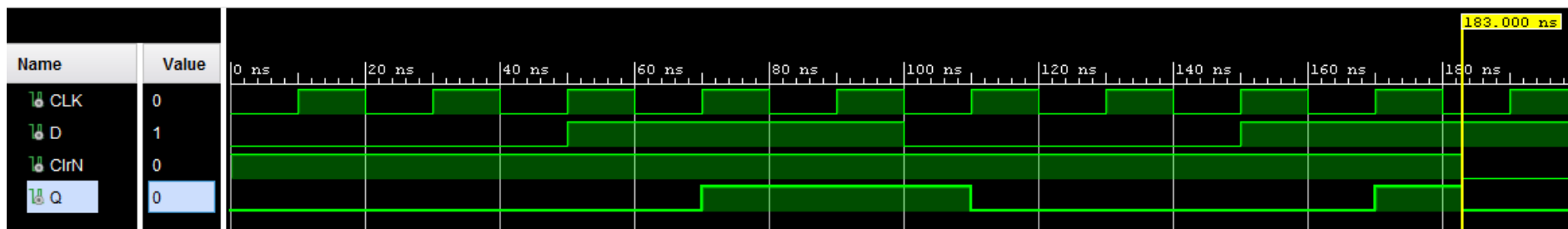
TestBench Example

- Always remember:
 - You are the designer, you can create any testbench waveform simulation.

```
38  architecture Behavioral of DFF_AC_tb is
39      component DFF_AC
40          port (
41              CLK, D, ClrN: in std_logic;
42              Q: out std_logic
43          );
44      end component;
45      signal CLK, D, ClrN, Q: std_logic := '0';
46  begin
47      uut: DFF_AC port map (CLK, D, ClrN, Q);
48      clk <= not clk after 10 ns;
49      process
50      begin
51          ClrN <= '1';
52          wait for 50 ns;
53          D <= '1';
54          wait for 50 ns;
55          D <= '0';
56          wait for 50 ns;
57          D <= '1';
58          wait for 33 ns;
59          ClrN <= '0';
60          wait;
61      end process;
62  end Behavioral;
```

Waveform

- Checking the 4 important points
 - D = '1' at 50 ns, Q = '1' at 70 ns
 - D = '0' at 100 ns, Q = '0' at 110 ns
 - D = '1' at 150 ns, Q = '1' at 170 ns
 - ClrN = '0' at 183 ns, Q = '0' at 183 ns (asynchronously)



In-class Exercise

- Please modify the previous example, and model a D-FF with synchronous clear signal

```
34 entity DFF_AC is
35   -- Port ( );
36   port (
37       CLK, D, ClrN: in std_logic;
38       Q: out std_logic
39   );
40 end DFF_AC;
41
42 architecture Behavioral of DFF_AC is
43 begin
44     process (CLK, ClrN)
45     begin
46         if CLRn = '0' then Q <= '0';
47         else if CLK'event and CLK = '1'
48             then Q <= D;
49         end if;
50     end if;
51 end process;
52 end Behavioral;
```

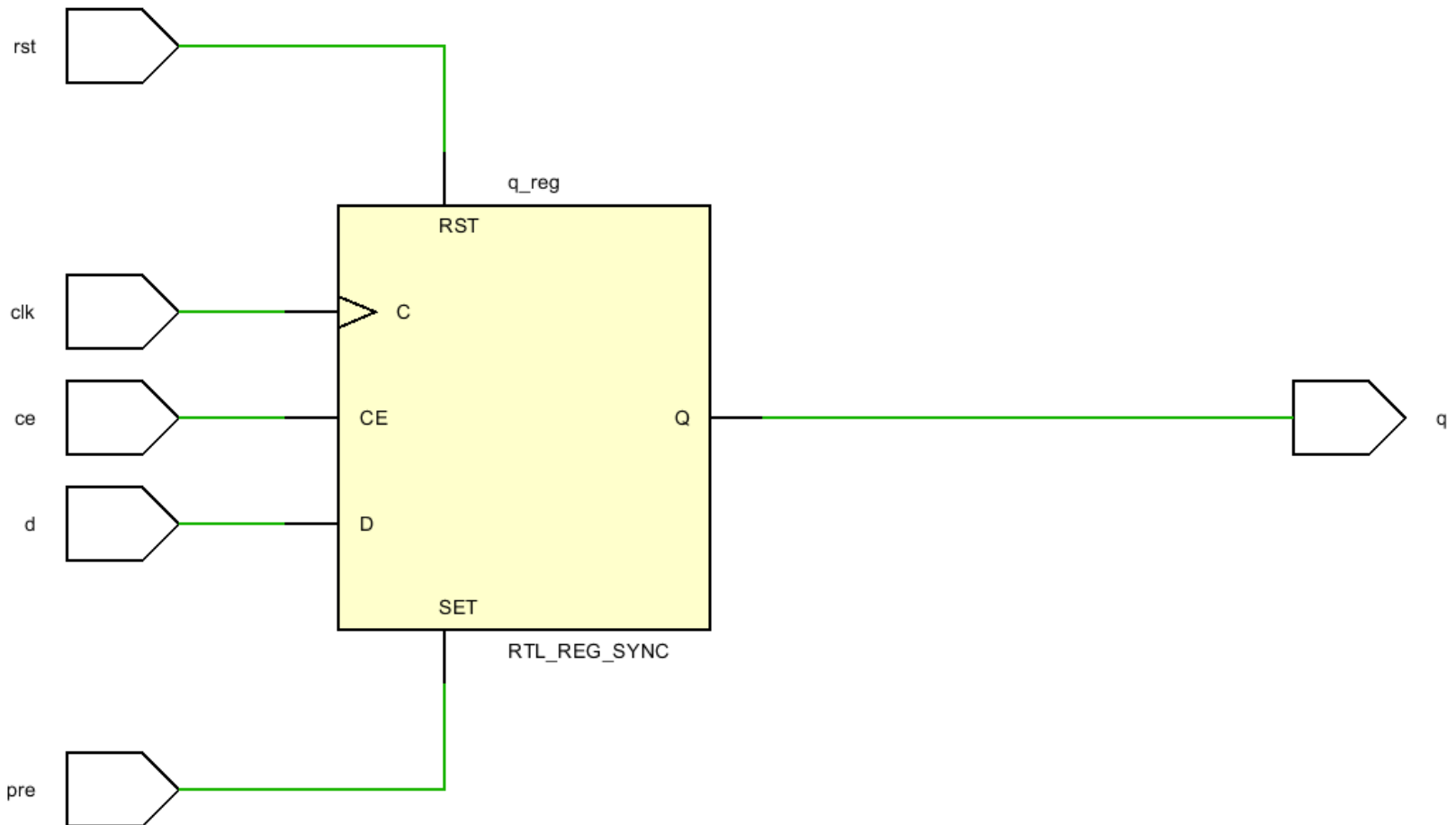
DFF with Preset & Clock Enable

- Using “rising_edge(clk)”
- Using multiple elsif statements
- How do you develop the testbench model for simulating this design?

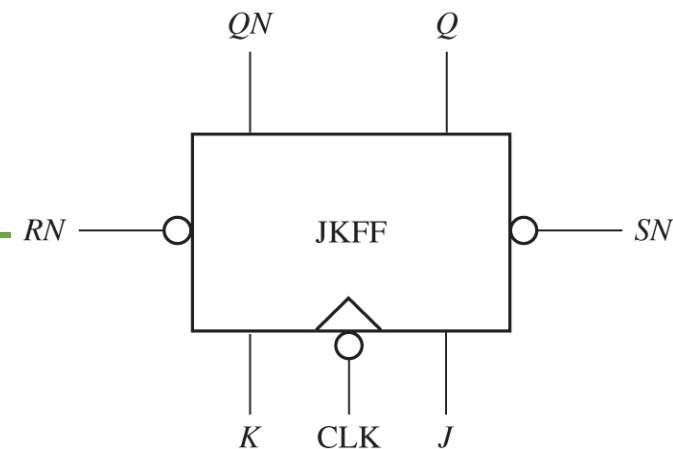
```
2 | library IEEE;
3 | use IEEE.STD_LOGIC_1164.ALL;
4 |
5 | entity DFF_Preset is
6 |     port (
7 |         clk : in std_logic;
8 |         rst : in std_logic;
9 |         pre : in std_logic;
10 |        ce  : in std_logic;
11 |        d   : in std_logic;
12 |        q   : out std_logic
13 |    );
14 | end DFF_Preset;
15 |
16 | architecture Behavioral of DFF_Preset is
17 | begin
18 |     process (clk) is
19 |     begin
20 |         if rising_edge(clk) then
21 |             if (rst='1') then
22 |                 q <= '0';
23 |             elsif (pre='1') then
24 |                 q <= '1';
25 |             elsif (ce='1') then
26 |                 if (d='1') then
27 |                     q <= '1';
28 |                 elsif (d='0') then
29 |                     q <= '0';
30 |                 end if;
31 |             end if;
32 |         end if;
33 |     end process;
34 | end architecture Behavioral;
```

Schematic

- How many input signals?



J-K Flip-Flop



- SN and RN Signals

entity JKFF is

```
    port(SN, RN, J, K, CLK: in bit; -- inputs
          Q, QN: out bit);
```

end JKFF;

architecture JKFF1 of JKFF is

```
    signal Qint: bit;          -- Qint can be used as input or output
```

begin

```
    Q <= Qint;                -- output Q and QN to port
```

```
    QN <= not Qint;           -- combinational output
```

```
                                -- outside process
```

```
    process(SN, RN, CLK)
```

```
    begin
```

```
        if RN = '0' then Qint <= '0' after 8 ns; -- RN = '0' will clear the FF
```

```
        elsif SN = '0' then Qint <= '1' after 8 ns; -- SN='0' will set the FF
```

```
        elsif CLK'event and CLK = '0' then          -- falling edge of CLK
```

```
            Qint <= (J and not Qint) or (not K and Qint) after 10 ns;
```

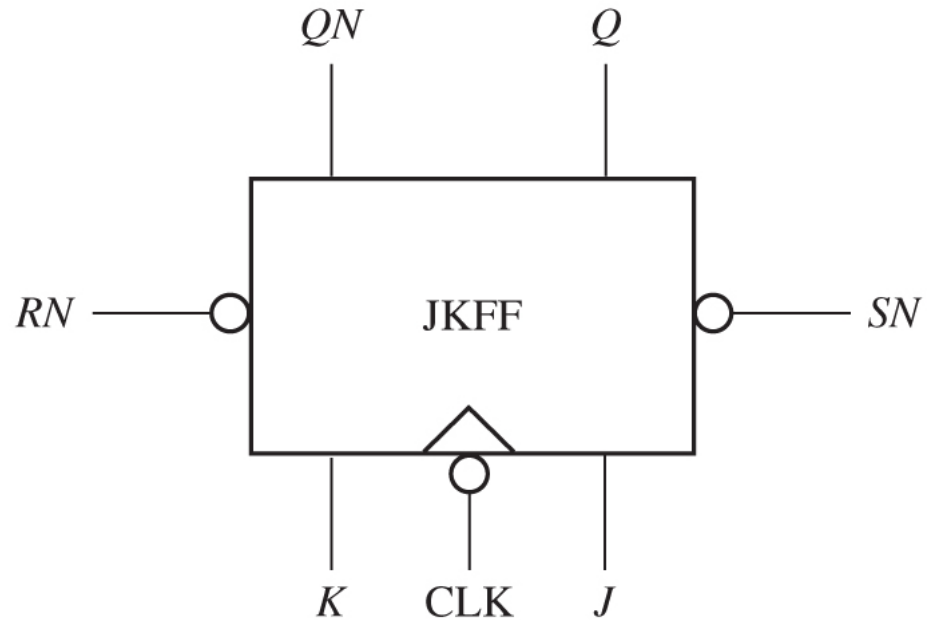
```
        end if;
```

```
    end process;
```

end JKFF1;

In-Class Exercise

- Please work on the testbench for this JK-FF design
- You have 5-mins to complete this task.



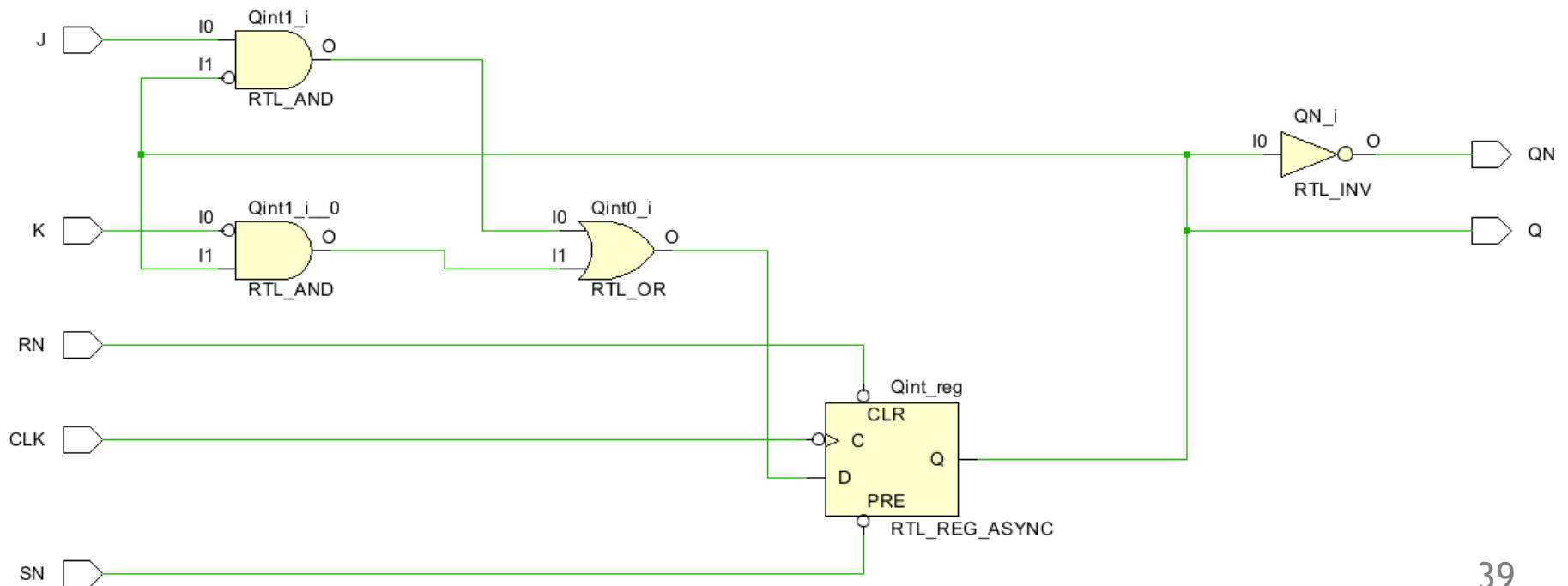
Full VHDL

- Check the RN and SN signals
- Memory Qint signal for holding the value

```
5 entity JKFF is
6     port (
7         SN, RN, J, K, CLK: in bit;
8         Q, QN: out bit
9     );
10 end JKFF;
11
12 architecture Behavioral of JKFF is
13     signal Qint: bit;
14     begin
15         Q <= Qint;
16         QN <= not Qint;
17
18         process(SN, RN, CLK)
19         begin
20             if RN = '0' then
21                 Qint <= '0';
22             elsif SN = '0' then
23                 Qint <= '1';
24             elsif CLK'event and CLK = '0' then
25                 Qint <= (J and not Qint) or (not K and Qint);
26             end if;
27         end process;
28     end Behavioral;
```

Schematic

- How do you simulate this design?
- Do you know if it is correct by looking at the schematic diagram



Processes Using Wait Statements

- Instead of using a sensitivity list, one can use wait statements.
- A process cannot have both a sensitivity list and wait statements.
- A process with wait statements may have the form:

```
process  
begin  
    sequential-statements  
    wait-statement  
    sequential-statements  
    wait-statement  
    . . .  
end process;
```

Processes Using Wait Statements

- Three forms of wait statements:
 1. **wait on** sensitivity-list;
 2. **wait for** time-expression;
 3. **wait until** Boolean-expression;
- The order of execution is not the same as the order in which signals are updated.
- If a signal is updated at the same time, the last value overrides.

“wait on” Example

- Process(A, B, C, D)
- Begin
 - $C \leq A$ and B ;
 - $E \leq C$ or D ;
- End process;

- Process
- Begin
 - $C \leq A$ and B ;
 - $E \leq C$ or D ;
 - Wait on A, B, C, D;
- End process;

“wait for” Example

- You use it all the time in the testbench design

```
38 architecture Behavioral of DFF_AC_tb is
39     component DFF_AC
40     port (
41         CLK, D, ClrN: in std_logic;
42         Q: out std_logic
43     );
44     end component;
45     signal CLK, D, ClrN, Q: std_logic := '0';
46 begin
47     uut: DFF_AC port map (CLK, D, ClrN, Q);
48     clk <= not clk after 10 ns;
49     process
50     begin
51         ClrN <= '1';
52         wait for 50 ns;
53         D <= '1';
54         wait for 50 ns;
55         D <= '0';
56         wait for 50 ns;
57         D <= '1';
58         wait for 33 ns;
59         ClrN <= '0';
60         wait;
61     end process;
62 end Behavioral;
```

“wait until” Example

- Process
- Begin
 - wait until clk'event and clk = '1';
 - A <= E;
 - B <= F;
 - C <= G;
- End process;
- The process waits for a rising clock edge.

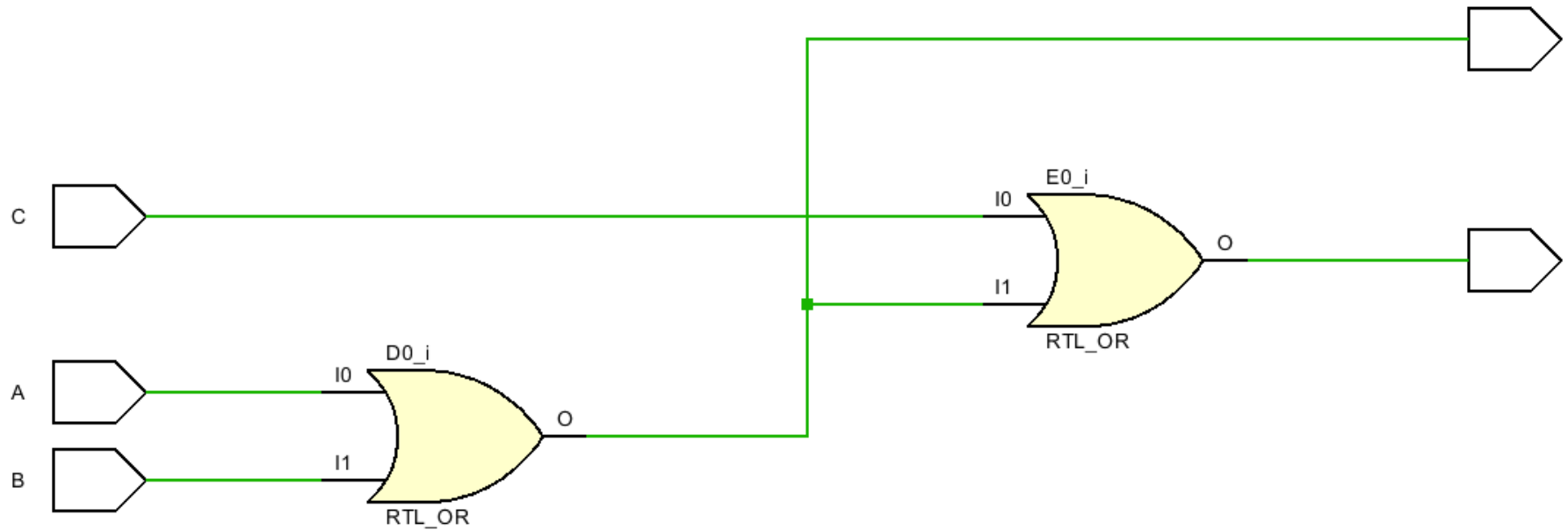
Example: NO Gates

- What will be generated?

```
entity nogates is  
  port(A, B, C: in bit;  
        D: buffer bit;  
        E: out bit);  
end nogates;  
  
architecture behave of nogates is  
begin  
  process(A, B, C)  
  begin  
    D <= A or B after 5 ns;  -- statement 1  
    E <= C or D after 5 ns;  -- statement 2  
  end process;  
end behave;
```

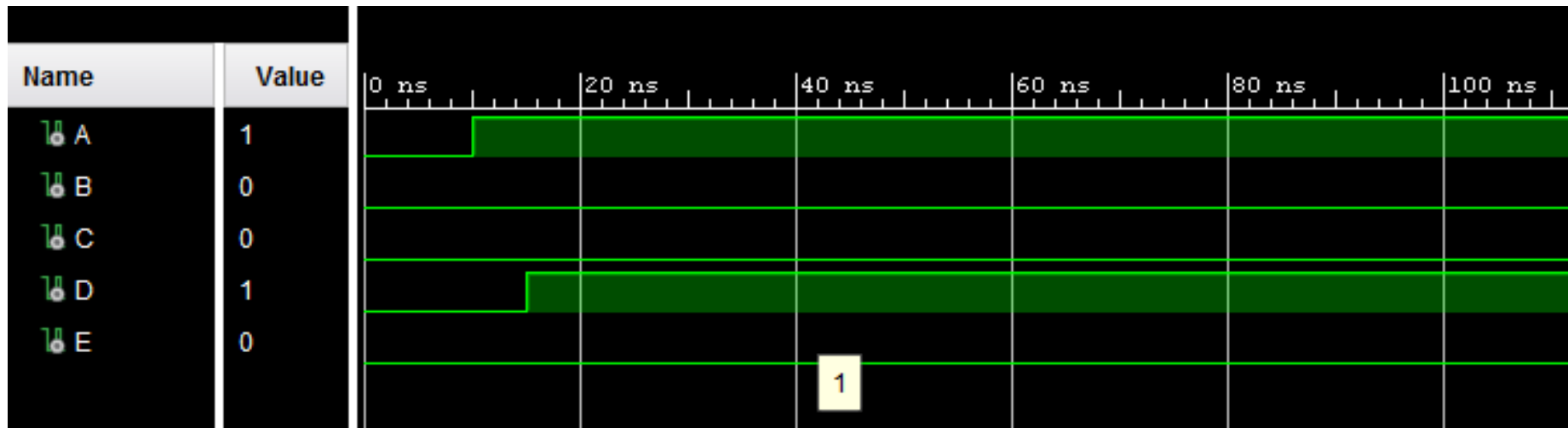
Schematic Diagram

- Looks pretty good!



Incorrect Waveform

- E should be high, when A has been set high.



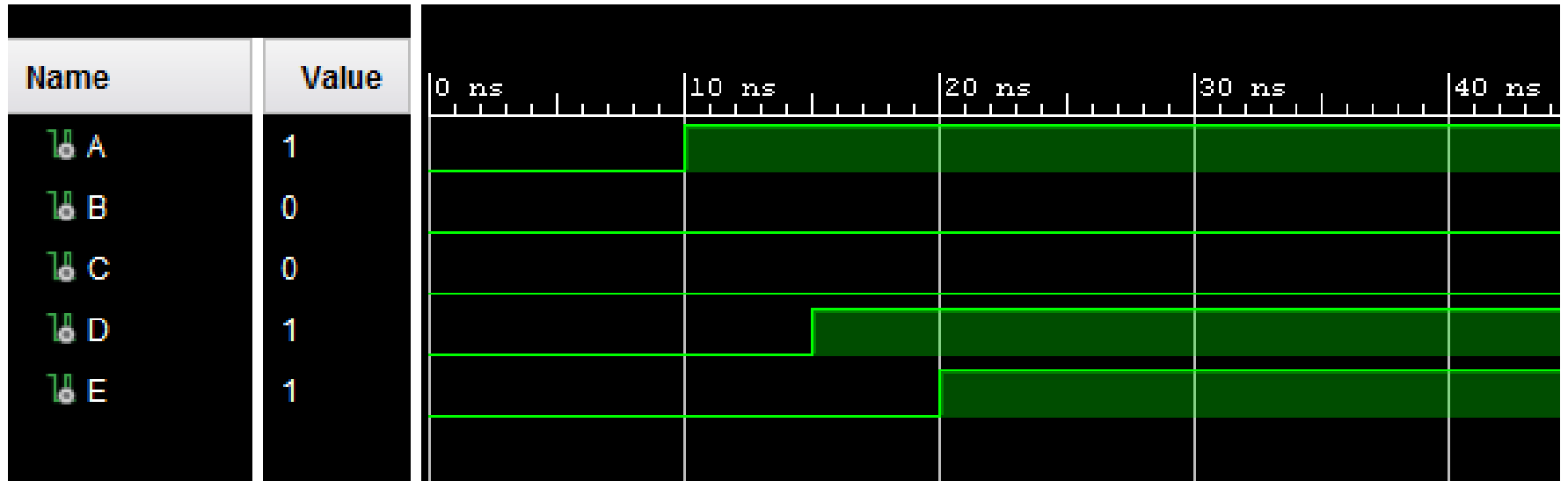
Why Sensitivity List matters?

- (A, B, C) vs. (A, B, C, D)

```
34 entity nogates is
35 -- Port ( );
36 port (
37     A, B, C: in bit;
38     D: buffer bit;
39     E: out bit
40 );
41 end nogates;
42
43 architecture Behavioral of nogates is
44
45 begin
46     process (A, B, C, D)
47     begin
48         D <= A or B after 5 ns;
49         E <= C or D after 5 ns;
50     end process;
51
52 end Behavioral;
```

Correct Waveform

- E will be changed after D changed.



Multiple Processes

- Very often we use multiple process in one architecture block
- P1 generates signal A value, P2 generates signal B value

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
  signal A,B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns; end if;
  end process P2;
end test1;
```

Simple Synthesis Examples

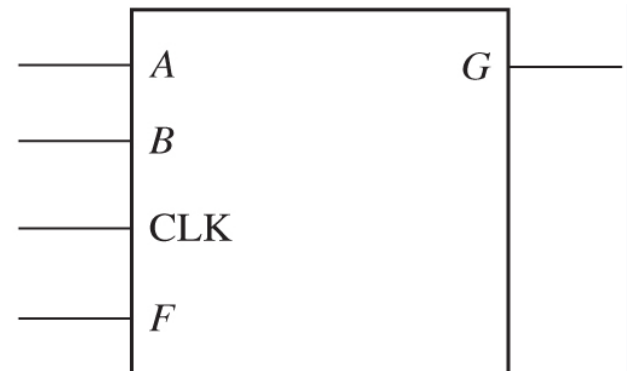
- In order for code to synthesize correctly, certain conventions must be followed.
- Even if VHDL code gives the correct result when simulated, it may not result in hardware that works correctly when synthesized.

Simple Synthesis Examples

- Considering the code below, what hardware will you get if you synthesized this code?
- Do you get Latches / Flip Flops?

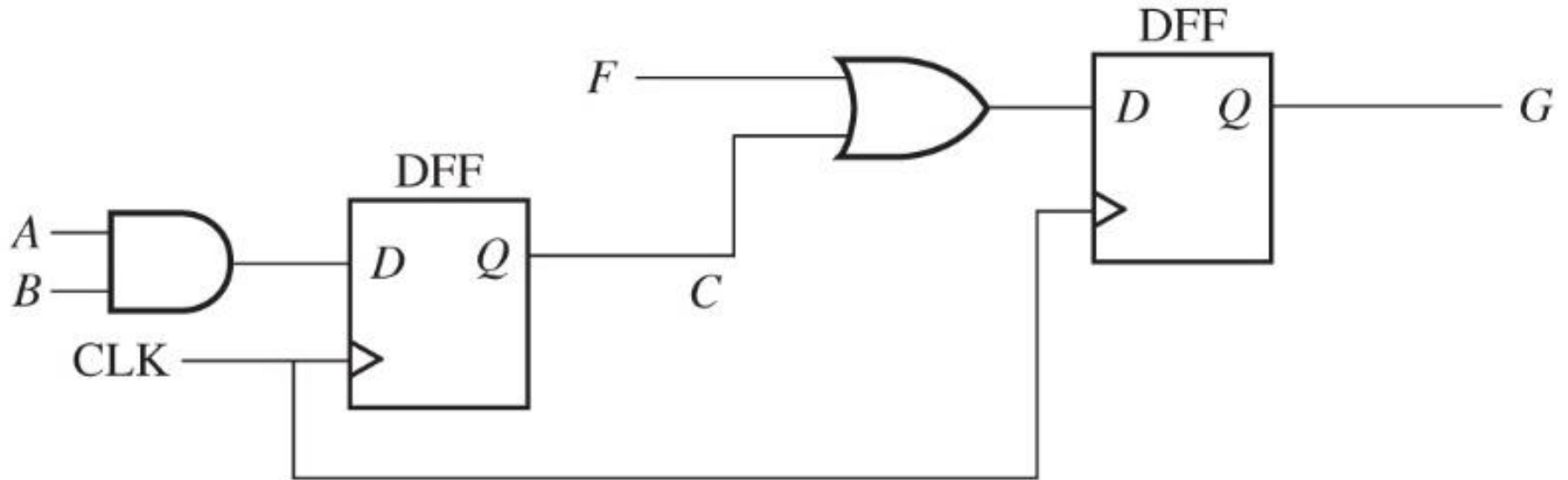
```
entity Q3 is
  port(A,B,F, CLK: in bit;
        G: out bit);
end Q3;

architecture circuit of Q3 is
  signal C: bit;
begin
  process(CLK)
  begin
    if (CLK = '1' and CLK'event) then
      C <= A and B; -- statement 1
      G <= C or F; -- statement 2
    end if;
  end process;
end circuit;
```



Simple Synthesis Examples

- Answer:



Outline

- Introduction
- VHDL Process
- VHDL MUX
- VHDL Register & Counter
- VHDL Synthesis
- More VHDL Examples
- Summary

VHDL Models for Multiplexers

- A **multiplexer** is a combinational circuit and can be modeled using:
 - A conditional or selective signal assignment statement
 - A case statement within a process
- A conditional signal assignment statement has the form:

```
signal_name <= expression1 when condition1  
    else expression2 when condition2  
    [else expressionN];
```

VHDL Models for Multiplexers

- A selected signal assignment statement has the form:

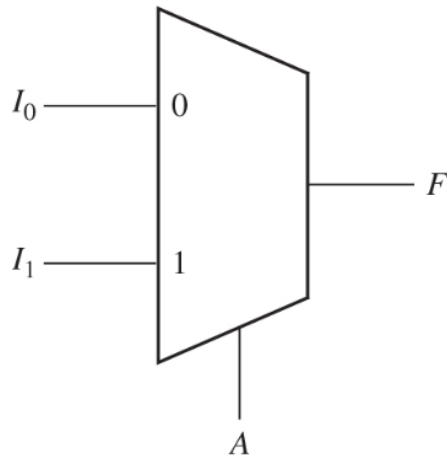
```
with expression_s select
    signal_s <= expression1 [after delay-time] when choice1,
               expression2 [after delay-time] when choice2,
               ...
               [expression_n [after delay-time] when others];
```

- A case statement has the form:

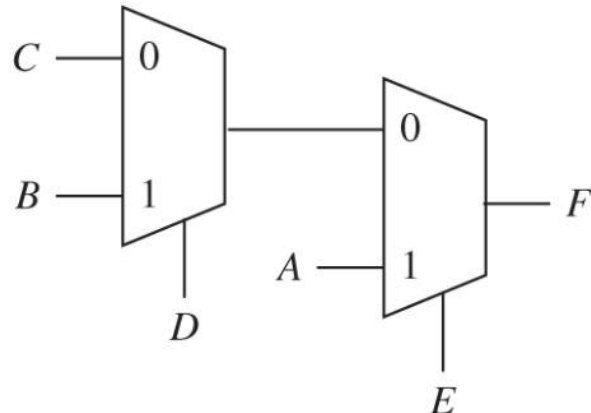
```
case expression is
    when choice1 => sequential statements1
    when choice2 => sequential statements2
    . . .
    [when others => sequential statements]
end case;
```

VHDL Models for Multiplexers

- Using concurrent statements



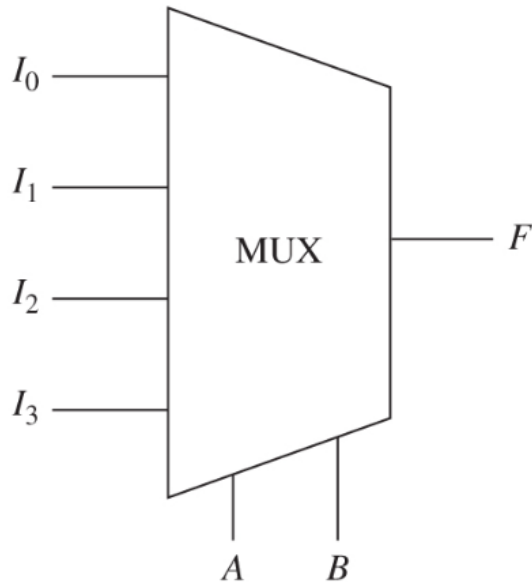
```
-- conditional signal assignment statement  
F <= I0 when A = '0' else I1;
```



```
F <= A when E = '1'  
      else B when D = '1'  
      else C;
```

4-to-1 MUX Example

- `Sel <= A & B;`



```
sel <= A&B;  
--selected signal assignment statement  
with sel select  
    F <= I0 when "00",  
        I1 when "01",  
        I2 when "10",  
        I3 when "11",
```

Outline

- Introduction
- VHDL Process
- VHDL MUX
- VHDL Register & Counter
- VHDL Synthesis
- More VHDL Examples
- Summary

Modeling Registers and Counters

- When several flip-flops change state on the same clock edge, statements representing these flip-flops can be placed in the same clocked processes.

- Example:

```
process (CLK)
begin
  if CLK'event and CLK = '1' then
    Q1 <= Q3 after 5 ns;
    Q2 <= Q1 after 5 ns;
    Q3 <= Q2 after 5 ns;
  end if;
end process;
```

- These flip-flops all change state following the rising edge of the clock, and the three statements in the always statement execute in sequence with no delay.

Modeling Registers and Counters

- If we were to omit the delay in the preceding example and replace the sequential statements with:

```
Q1 <= Q3;  Q2 <= Q1;  Q3 <= Q2;
```

the operation would be essentially the same. The three statements execute in sequence in zero time, and then the Q's values change after a delta delay.

Modeling Registers and Counters

- VHDL code for a simple synchronous counter:

```
signal Q: unsigned (3 downto 0);  
-----  
process (CLK)  
begin  
    if CLK'event and CLK = '1' then  
        if ClrN = '0' then Q <= "0000";  
        elsif En = '1' then Q <= Q + 1;  
        end if;  
    end if;  
end process;
```

- Cleared on the rising edge of the clock when ClrN = '0' and incremented when ClrN = En = '1'.

Modeling Registers and Counters

- Avoiding unwanted latches:

- VHDL signals retain their current values until they are changed. This can result in the creation of unwanted latches when the code is synthesized.

- For example:

```
if X = '1' then B <= 1; end if;
```

creates latches to hold B when X changes to '0'.

- Always include an **else** clause to avoid unwanted latches.

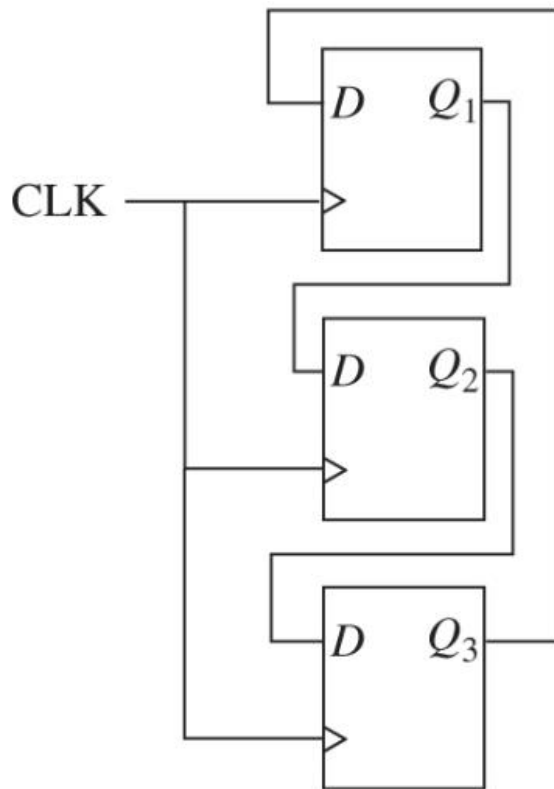
- For example:

```
if X = '1' then B <= 1 else B <= 0; end if;
```

creates a MUX to switch B from 1 to 0.

Cyclic Shift Register

- $Q1 \rightarrow Q2 \rightarrow Q3$



```
process(CLK)
begin
    if CLK'event and CLK = '1' then
        Q1 <= Q3 after 5 ns;
        Q2 <= Q1 after 5 ns;
        Q3 <= Q2 after 5 ns;
    end if;
end process;
```

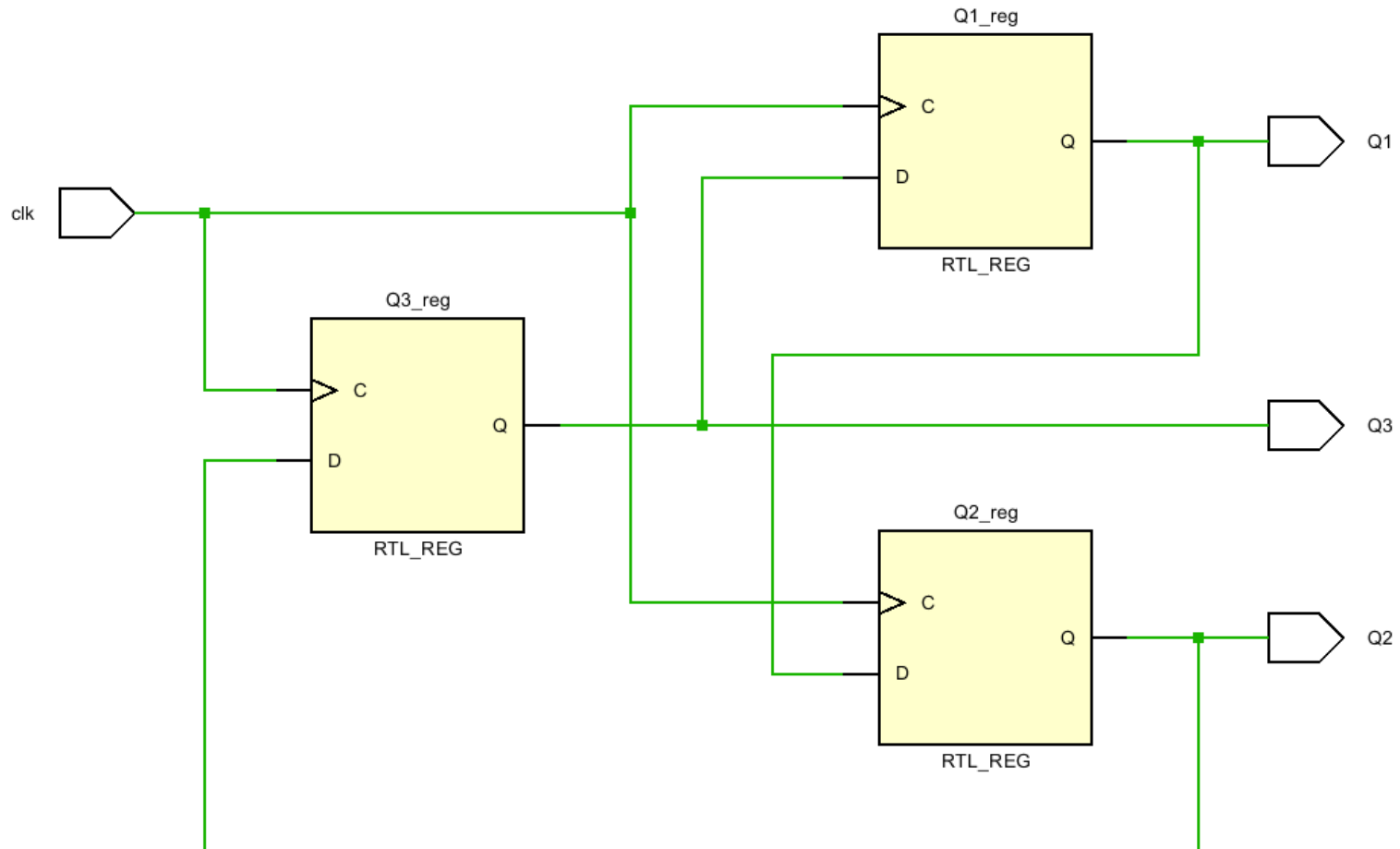
VHDL Code

- Use “Buffer” or “out”?

```
5 entity ShiftRegister is
6     port(
7         clk: in std_logic;
8         Q1, Q2, Q3: buffer std_logic
9     );
10 end ShiftRegister;
11
12 architecture Behavioral of ShiftRegister is
13
14 begin
15     process(clk)
16     begin
17         if clk'event and clk = '1' then
18             Q1 <= Q3 ;
19             Q2 <= Q1 ;
20             Q3 <= Q2 ;
21         end if;
22     end process;
23 end Behavioral;
```

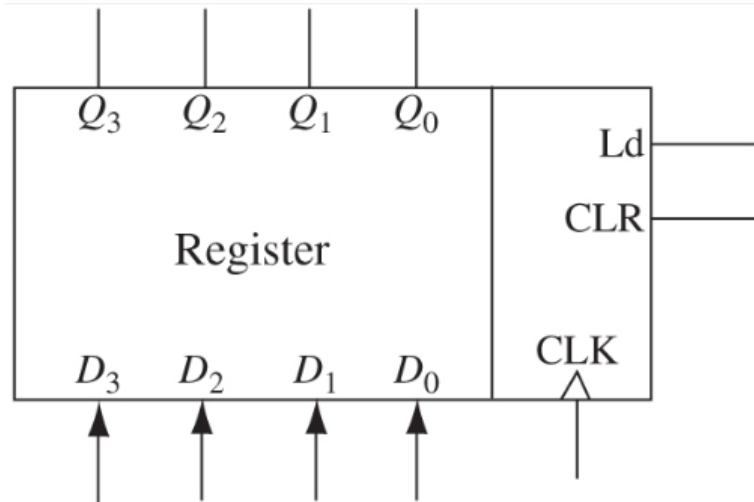
Schematic

- Check the connections



Register with Synchronous Clear and Load

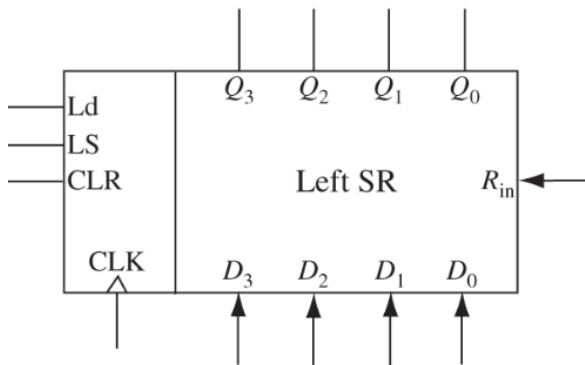
- Load and clear on the rising edge of the clock



```
process(CLK)
begin
    if CLK'event and CLK = '1' then
        if CLR = '1' then Q <= "0000";
        elsif Ld = '1' then Q <= D;
        end if;
    end if;
end process;
```

Left Shift Register with Clear and Load

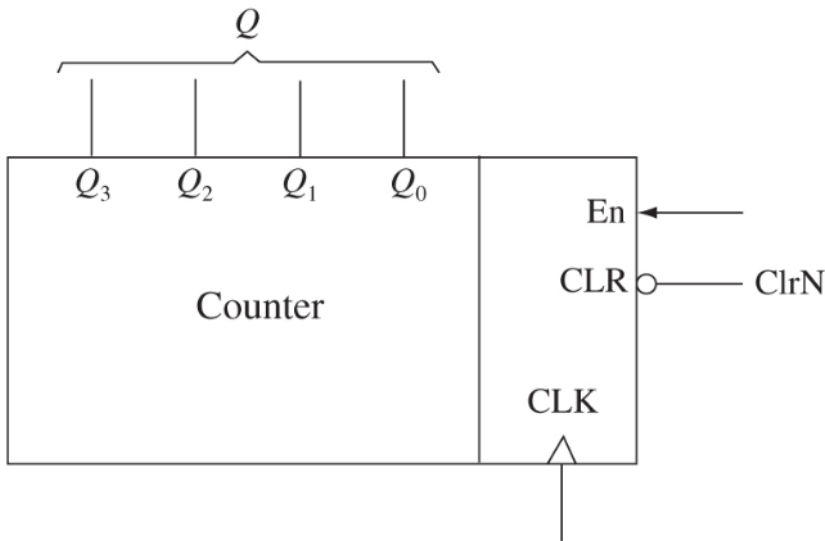
- What is the major difference as compared to the previous example?



```
process(CLK)
begin
  if CLK'event and CLK = '1' then
    if CLR = '1' then Q <= "0000";
    elsif Ld = '1' then Q <= D;
    elsif LS = '1' then Q <= Q(2 downto 0) & Rin;
    end if;
  end if;
end process;
```

Synchronous Counter

- process (CLK)



```
signal Q: unsigned (3 downto 0);  
-----  
process (CLK)  
begin  
    if CLK'event and CLK = '1' then  
        if ClrN = '0' then Q <= "0000";  
        elsif En = '1' then Q <= Q + 1;  
        end if;  
    end if;  
end process;
```

Mealy State Machine

- The output Z and next state are computed before the active edge of the clock
- State change at the rising edge of the clock

```
entity Code_Converter is
    port(X, CLK: in bit;
         Z: out bit);
end Code_Converter;

architecture Behavioral of Code_Converter is
    signal State, Nextstate: integer range 0 to 6;
begin
    process(State, X) -- Combinational Circuit
    begin
        case State is
            when 0 =>
                if X = '0' then Z <= '1'; Nextstate <= 1;
                else Z <= '0'; Nextstate <= 2; end if;
            when 1 =>
                if X = '0' then Z <= '1'; Nextstate <= 3;
                else Z <= '0'; Nextstate <= 4; end if;
            when 2 =>
                if X = '0' then Z <= '0'; Nextstate <= 4;
                else Z <= '1'; Nextstate <= 4; end if;
            when 3 =>
                if X = '0' then Z <= '0'; Nextstate <= 5;
                else Z <= '1'; Nextstate <= 5; end if;
            when 4 =>
                if X = '0' then Z <= '1'; Nextstate <= 5;
                else Z <= '0'; Nextstate <= 6; end if;
            when 5 =>
                if X = '0' then Z <= '0'; Nextstate <= 0;
                else Z <= '1'; Nextstate <= 0; end if;
            when 6 =>
                if X = '0' then Z <= '1'; Nextstate <= 0;
                else Z <= '0'; Nextstate <= 0; end if;
            when others => null; -- should not occur
        end case;
    end process;

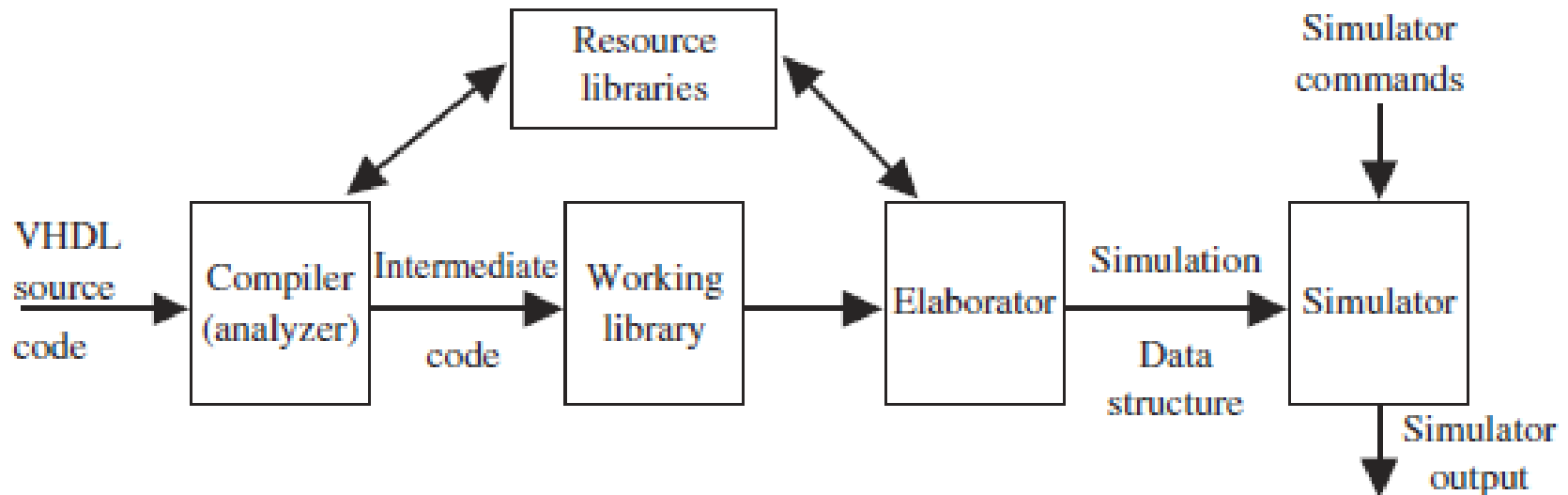
    process(CLK) -- State Register
    begin
        if CLK'EVENT and CLK = '1' then -- rising edge of clock
            State <= Nextstate;
        end if;
    end process;
end Behavioral;
```

Outline

- Introduction
- VHDL Process
- VHDL MUX
- VHDL Register & Counter
- VHDL Synthesis
- More VHDL Examples
- Summary

Compilation, Simulation, and Synthesis of VHDL Code

- Three phases: analysis (compilation), elaboration, and simulation



Compilation, Simulation, and Synthesis of VHDL Code

- **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

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



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

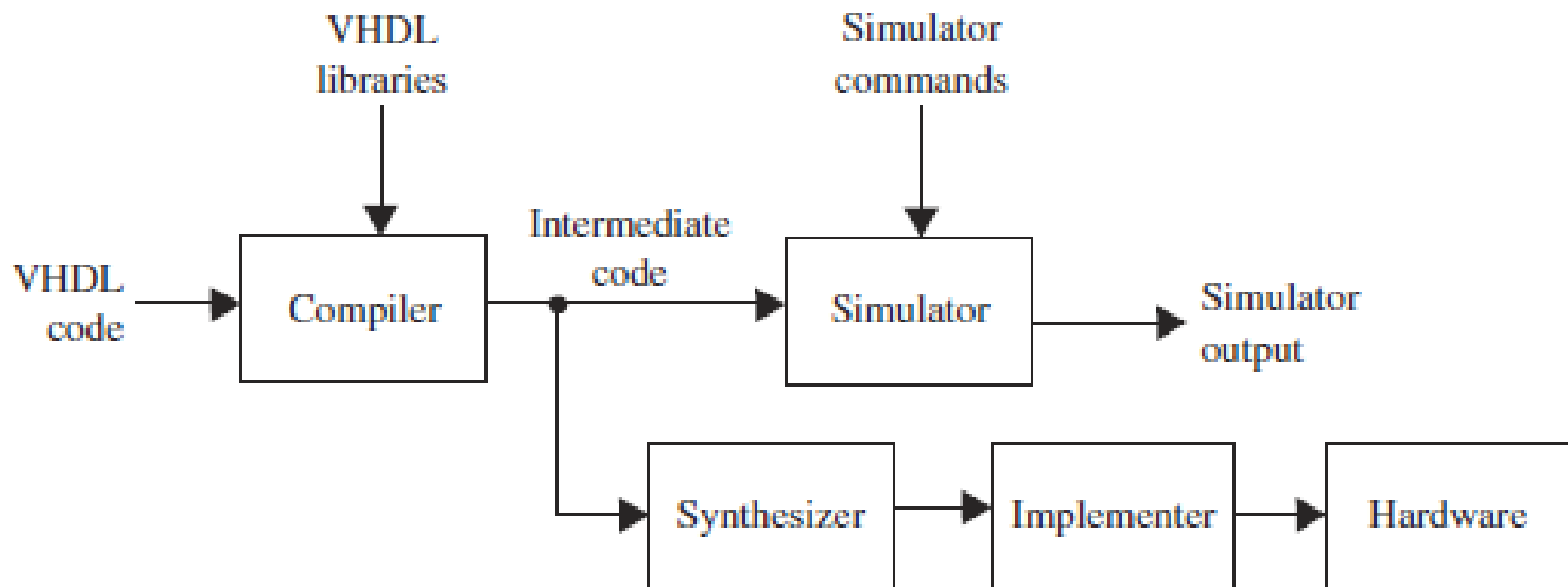
ns	delta	A	B	C	D
0	+0	0	1	0	1
3	+0	1	1	0	1
3	+1	1	0	0	1
3	+2	1	0	1	1
8	+0	1	0	1	0

Compilation, Simulation, and Synthesis of VHDL Code

- Simulation with multiple processes:
 - Processes will execute concurrently.
 - Statements inside processes occur sequentially.
 - Processes take no time to execute unless they have wait statements.
 - Each time an event occurs, any processes that have been waiting on the event are executed in zero time, and any resulting signal changes are queued up to occur at some future time.
 - One of the most important uses of VHDL is to synthesize or automatically create hardware from a VHDL description.

Compilation, Simulation, and Synthesis of VHDL Code

- Simulation and synthesis process:



Tips for Debugging VHDL Code

- Make sure that your test sequences are appropriately designed so you do not interpret a working circuit as incorrect.
- Do not put statements sequentially inside a process if they need to be executed at the same time.
- If there are multiple statements that should take effect during execution of a case in a case statement, they need to be independent.
- If the same constant number is used in multiple places, use a constant symbol so the constant is typed only once.
- Check ranges of all signals. They should be at least equal to the highest value computed in the program. Check the warning signals in addition to fixing errors.

Outline

- Introduction
- VHDL Process
- VHDL MUX
- VHDL Register & Counter
- VHDL Synthesis
- More VHDL Examples
- Summary

VHDL Libraries

- Extend the functionality of VHDL.
- Define types, functions, components, and overloaded operators.
- Need a library statement and a use statement.

```
library IEEE;  
use IEEE.numeric_bit.all;
```
- The **numeric_bit** package uses **bit_vectors** to represent unsigned and signed binary numbers.

Example - Unsigned Vectors

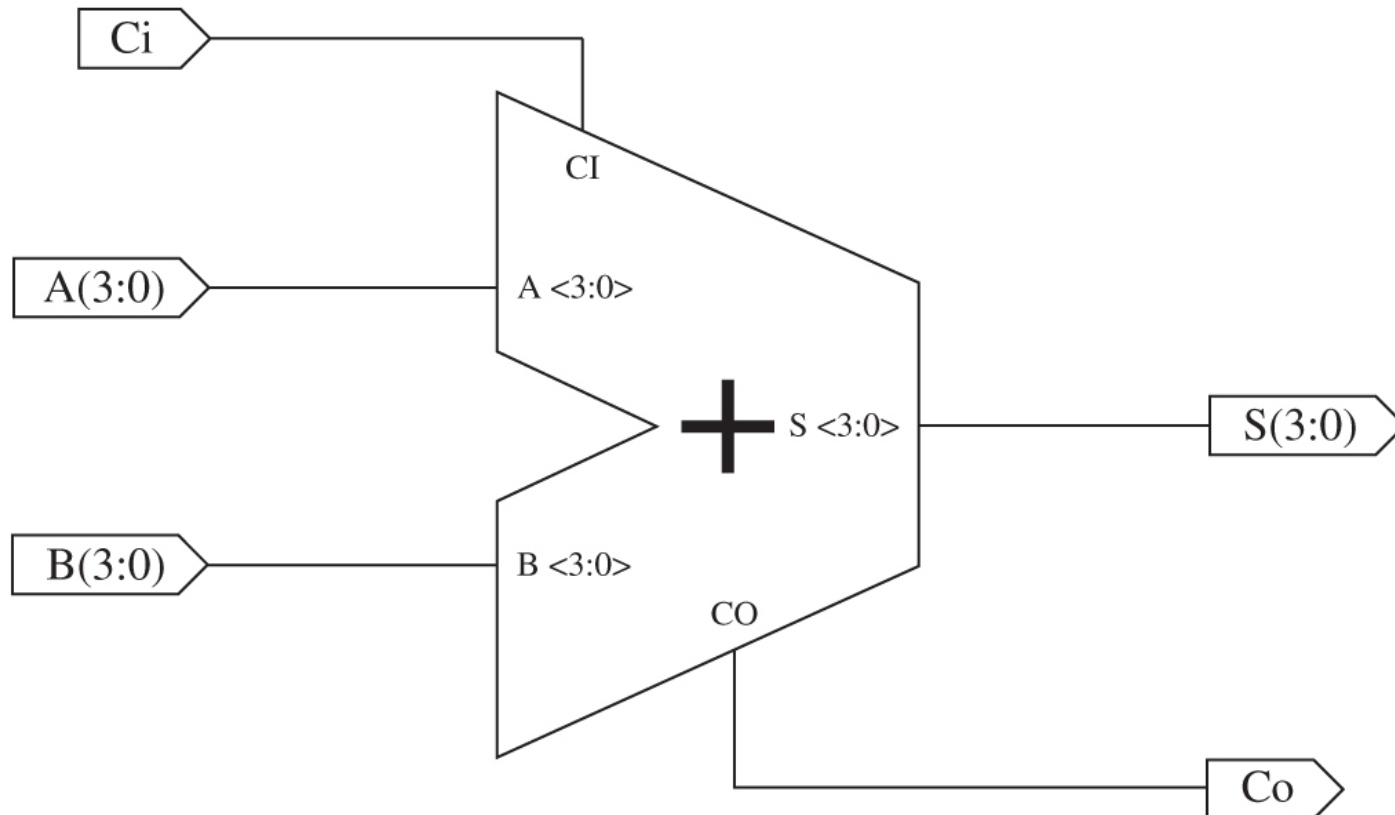
- Check the unsigned operations

```
library IEEE;
use IEEE.numeric_bit.all;
entity Adder4 is
    port(A, B: in unsigned(3 downto 0); Ci: in bit;    -- Inputs
          S: out unsigned(3 downto 0); Co: out bit);    -- Outputs
end Adder4;

architecture overload of Adder4 is
    signal Sum5: unsigned(4 downto 0);
begin
    Sum5 <= '0' & A + B + unsigned'(0=>Ci);    -- adder
    S <= Sum5(3 downto 0);
    Co <= Sum5(4);
end overload;
```

Schematic

- Synthesized output for the VHDL Code



Example - std_logic_unsigned

- 4-bit Adder example

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity Adder4 is
    port(A, B: in std_logic_vector(3 downto 0); Ci: in std_logic; --Inputs
          S: out std_logic_vector(3 downto 0); Co: out std_logic); --Outputs
end Adder4;

architecture overload of Adder4 is
    signal Sum5: std_logic_vector(4 downto 0);
begin
    Sum5 <= '0' & A + B + Ci; --adder
    S <= Sum5(3 downto 0);
    Co <= Sum5(4);
end overload;
```

Signals vs. Variables

- This one uses variable:

```
entity dummy is  
end dummy;
```

```
architecture var of dummy is  
signal trigger, sum: integer:=0;  
begin
```

```
  process
```

```
    variable var1: integer:=1;
```

```
    variable var2: integer:=2;
```

```
    variable var3: integer:=3;
```

```
  begin
```

```
    wait on trigger;
```

```
    var1 := var2 + var3;
```

```
    var2 := var1;
```

```
    var3 := var2;
```

```
    sum <= var1 + var2 + var3;
```

```
  end process;
```

```
end var;
```

Simulation Output of 2-60

ns	delta	trigger	Var1	Var2	Var3	Sum
0	+0	0	1	2	3	0
0	+1	0	1	2	3	0
10	+0	1	5	5	5	0
10	+1	1	5	5	5	15

Signals vs. Variables

- This one uses signals:

```
entity dummy is
end dummy;
architecture sig of dummy is
signal trigger, sum: integer:=0;
signal sig1: integer:=1;
signal sig2: integer:=2;
signal sig3: integer:=3;
begin
  process
  begin
    wait on trigger;
    sig1 <= sig2 + sig3;
    sig2 <= sig1;
    sig3 <= sig2;
    sum <= sig1 + sig2 + sig3;
  end process;
end sig;
```

Simulation Output of 2-61

ns	delta	trigger	Sig1	Sig2	Sig3	Sum
0	+0	0	1	2	3	0
0	+1	0	1	2	3	0
10	+0	1	1	2	3	0
10	+1	1	5	1	2	6

Arrays

- VHDL arrays can be used to create memory arrays and specify the values to be stored in these arrays.

- Must declare an array type and array object. Ex:

```
type array_type_name is array index_range of element_type;  
signal array_name: array_type_name [ := initial_values];
```

- **Matrices:**

- Multidimensional arrays, for example:

```
type matrix4x3 is array (1 to 4, 1 to 3) of integer;  
variable matrixA: matrix4x3 := ((1, 2, 3), (4, 5, 6), (7, 8, 9),  
                                (10, 11, 12));
```

- **Unconstrained array:**

- Dimensions are left undefined when an array is declared.
- Range must be specified when the array object is declared.

Arrays Example

- A new data type OutTable is defined to be an array of 16 bits:

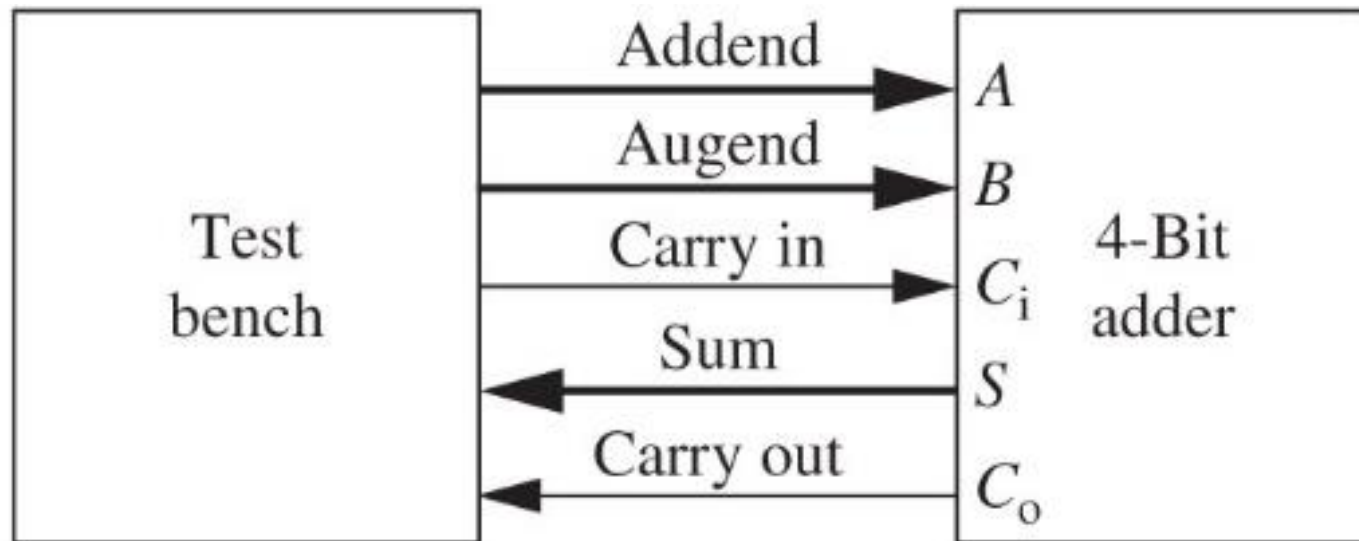
```
library IEEE;
use IEEE.numeric_bit.all;

entity parity_gen is
  port(X: in unsigned(3 downto 0);
       Y: out unsigned(4 downto 0));
end parity_gen;

architecture Table of parity_gen is
  type OutTable is array(0 to 15) of bit;
  signal ParityBit: bit;
  constant OT: OutTable := ('1','0','0','1','0','1','1','0',
                             '0','1','1','0','1','0','0','1');
begin
  ParityBit <= OT(to_integer(X));
  Y <= X & ParityBit;
end Table;
```

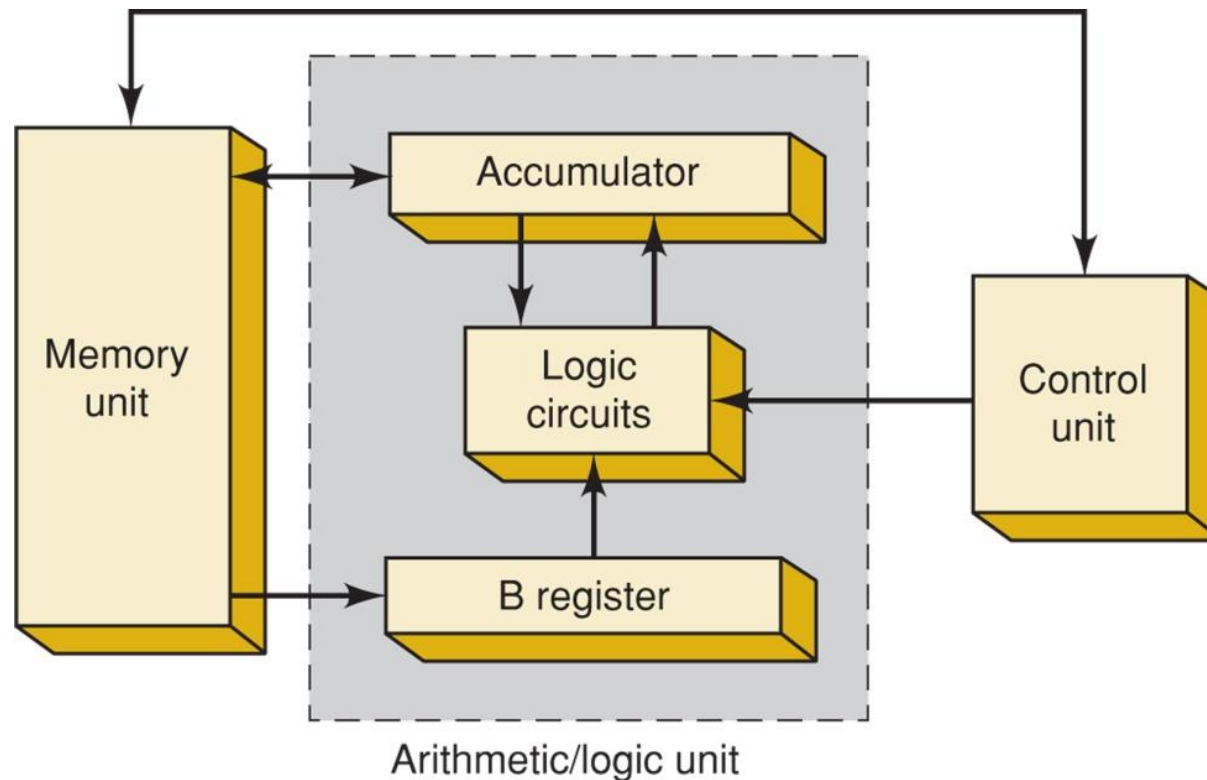
Test Bench

- Test bench
 - Code that provides input combinations to test a model.
- Test bench for testing a 4-bit binary adder:



Arithmetic Circuits

- An arithmetic/logic unit (ALU) accepts data stored in memory, and executes arithmetic and logic operations as instructed by the control unit.



Summary

- VHDL can be used to model combinational logic and sequential machines.
- Some of the VHDL Codes are unable to compile.
- It is important to learn how to use Process statement properly
- Be careful when we use the sensitivity list.
- Compilation, Simulation, Synthesis Steps.
- Arrays in VHDL can be used while modeling the repetition.

