Lecture 6 (Chapter 2)

Introduction to VHDL: Processes

Khaza Anuarul Hoque
ECE 4250/7250

VHDL Processes

A VHDL process has the following basic form:

```
process(sensitivity-list)
begin
   sequential-statements
end process;
```

VHDL Processes

When the concurrent statements

```
C <= A and B;  -- concurrent
E <= C or D;   -- statements
```

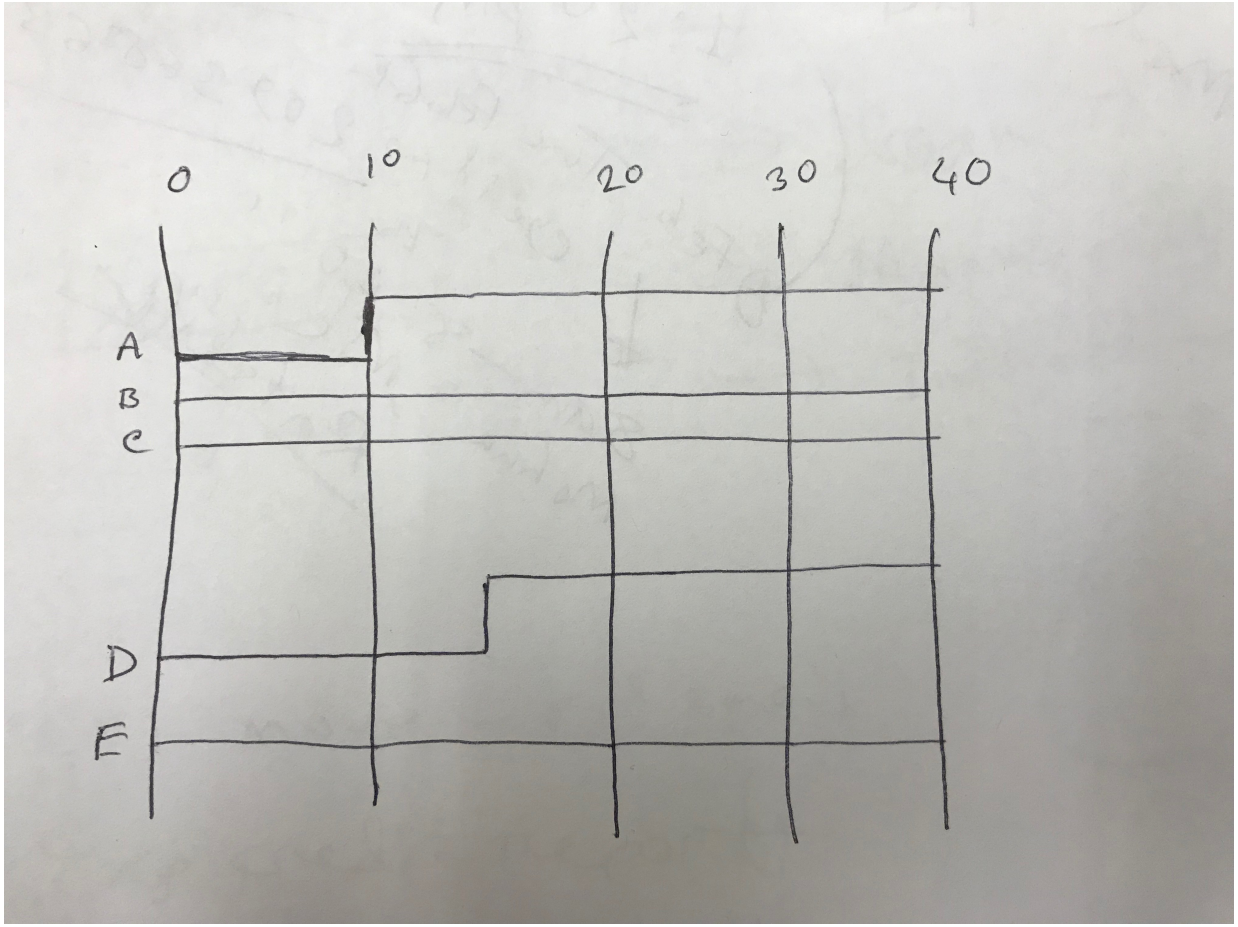
```
process(A, B, C, D)
begin
  C <= A and B;   -- sequential
  E <= C or D;    -- statements
end process;
```

VHDL code with a processes

```
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;
```

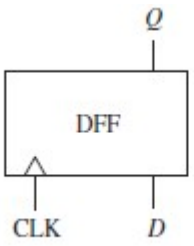
VHDL code with a processes



Modeling Flip-Flops Using VHDL Processes

- 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
- Example:

FIGURE 2-16: VHDL Code for a Simple D Flip-Flop

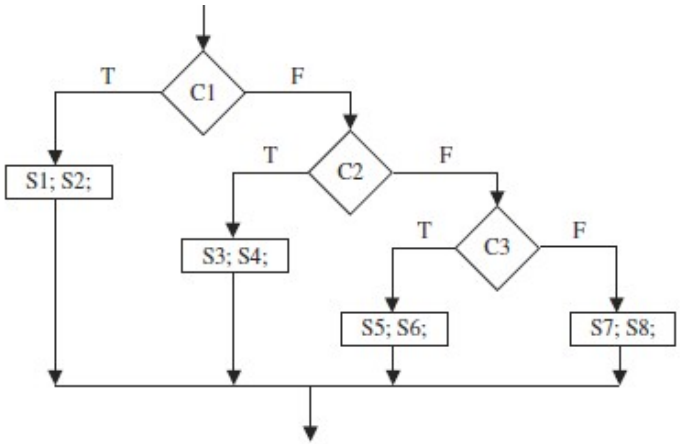



```
process(CLK)
begin
  if CLK'event and CLK = '1'  -- rising edge of CLK
    then Q <= D;
  end if;
end process;
```

Modeling Flip-Flops Using VHDL Processes (continued)

- Flowchart depiction using nested Ifs and Else Ifs:

FIGURE 2-19:
Equivalent Representations of a Flow Chart Using Nested Ifs and Elsifs




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

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 (continued)

- Three forms of wait statements:
  1. `wait on sensitivity-list;`
  2. `wait for time-expression;`
  3. `wait until Boolean-expression;`

Processes Using Wait Statements (continued)

```
process(A, B, C, D)
begin
  C <= A and B after 5 ns;
  E <= C or D after 5 ns;
end process;
```

```
process
begin
  C <= A and B after 5 ns;
  E <= C or D after 5 ns;
  wait on A, B, C, D;
end process;
```

Processes Using Wait Statements (continued)

The order of execution is not the same as the order in which signals are updated.

```
process
begin
  wait until clk'event and clk = '1';
  A <= E after 10 ns;   -- (1)
  B <= F after 5 ns;    -- (2)
  C <= G;  -- (3)
  D <= H after 5 ns;    -- (4)
end process;
```

Processes Using Wait Statements (continued)

If a signal is updated at the same time, the last value overrides.

```
process(CLK)
begin
  if CLK'event and CLK = '0' then
    Q <= A; Q <= B; Q <= C;
  end if;
end process;
```

Processes Using Wait Statements (continued)

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

architecture exam of gates is
begin
  process
  begin
    D <= A or B after 2 ns;
    E <= not C and A;
  end process;
end exam;
```

Variables, Signals, and Constants

- **Variables**:
  - Declaration of the form:
    ```
    variable list_of_variable_names: type_name [ := initial_value];
    ```
  - Must be declared inside a process.
  - Are local to the process.
  - Updated instantly using:
    ```
    variable_name := expression;
    ```
- **Signals**:
  - Declaration of the form:
    ```
    signal list_of_signal_names: type_name [ := initial_value];
    ```
  - Must be declared outside a process.
  - Can be used anywhere within the architecture if declared at the start.
  - Updated after a delay using:
    ```
    signal_name <= expression [after delay];
    ```

Variables, Signals, and Constants (continued)

- **Constants**:
  - Common declaration form:

```
constant constant_name: type_name := constant_value;
```

  - Constants declared at the start of an architecture can be used anywhere in that architecture.
  - Constants declared within a process are local to that process.

Variables, Signals, and Constants (continued)

When to Use a Signal versus a Variable: If whatever you are modeling actually corresponds to some physical signal in your circuit, you should use a signal. If whatever you are modeling is simply a temporary value that you are using for convenience of programming, a variable will be sufficient. Values represented using variables will not appear on any physical wire in the implied circuit. If you would like them to appear, you should use signals.

```
variable_name <= expression [after delay];
```

Variables, Signals, and Constants (continued)

FIGURE 2-60: Process Using Variables and Corresponding Simulation Output

```
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

Variables, Signals, and Constants (continued)

FIGURE 2-61: Process Using Signals and Corresponding Simulation Output

```
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

Variables, Signals, and Constants (continued)

FIGURE 2-62: Process Using Variables and Corresponding Simulation Output

```
entity dummy is
end dummy;

architecture var of dummy is
signal trigger, sum: integer:=0;
begin
  process(trigger)
  variable var1: integer:=1;
  variable var2: integer:=2;
  variable var3: integer:=3;
  begin
    var1 := var2 + var3;
    var2 := var1;
    var3 := var2;
    sum <= var1 + var2 + var3;
  end process;
end var;
```

Simulation Output of 2-62

ns	delta	trigger	Var1	Var2	Var3	Sum
0	+0	0	1	2	3	0
0	+1	0	5	5	5	15
10	+0	1	10	10	10	15
10	+1	1	10	10	10	30

Variables, Signals, and Constants (continued)

FIGURE 2-63: Process Using Signals and Corresponding Simulation Output

```
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(trigger)
  begin
    sig1 <= sig2 + sig3;
    sig2 <= sig1;
    sig3 <= sig2;
    sum <= sig1 + sig2 + sig3;
  end process;
end sig;
```

Simulation Output of 2-63

ns	delta	trigger	Sig1	Sig2	Sig3	Sum
0	+0	0	1	2	3	0
0	+1	0	5	1	2	6
10	+0	1	5	1	2	6
10	+1	1	3	5	1	8