Lecture 7

VHDL (Part-2)

Concurrent and Sequential Statements, Loops

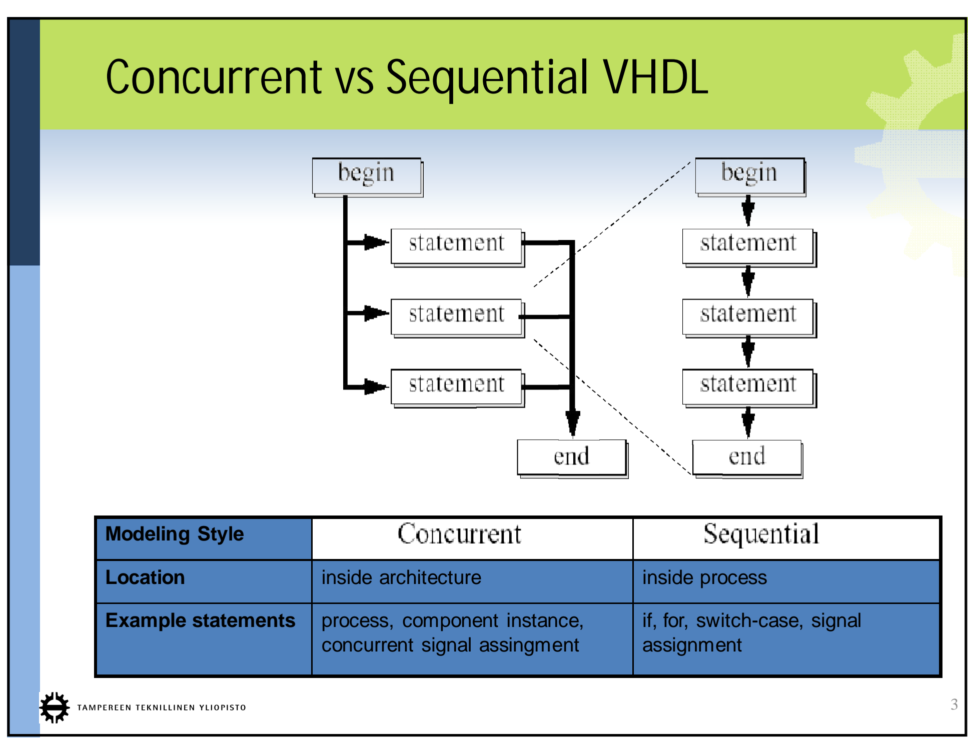
TAMPEREEN TEKNILLINEN YLIOPISTO

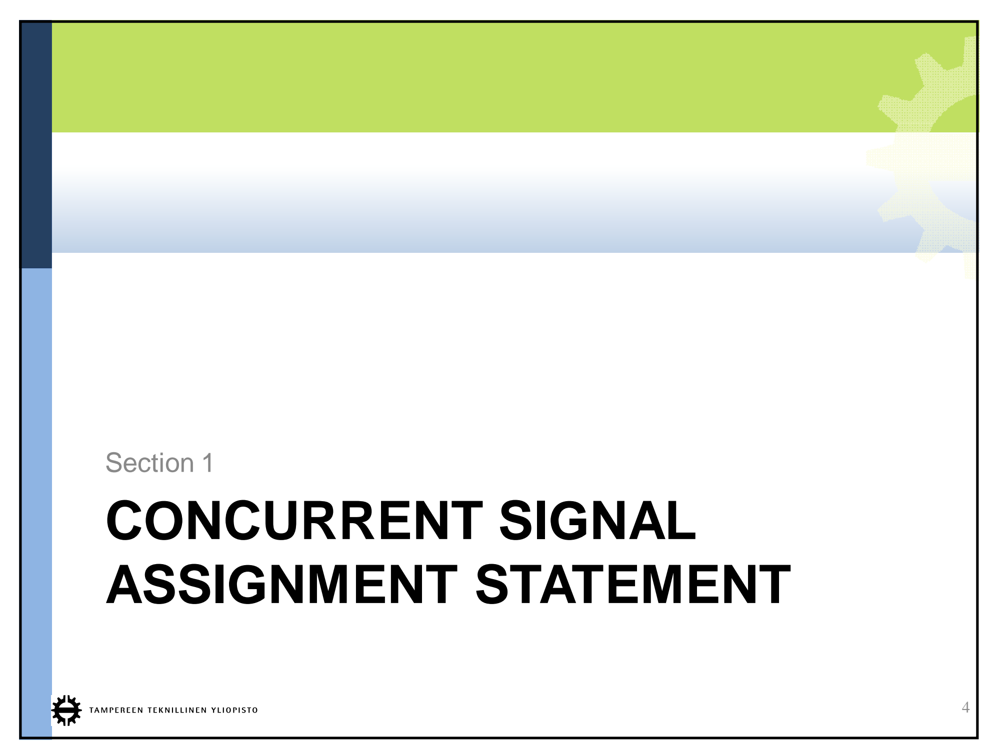
Concurrent vs Sequential Statements

Concurrent statements
- Simple signal assignment statement
- Conditional signal assignment statement
- Selected signal assignment statement

Sequential Statements
- VHDL process
- Sequential signal assignment statement
- Variable assignment statement
- If statement
- Case statement
- Simple for loop statement

2


Concurrent vs Sequential VHDL
begin
statement
statement
statement
end
begin
statement
statement
statement
end
Modeling Style
Concurrent
Sequential
Location
inside architecture
inside process
Example statements
process, component instance, concurrent signal assingment
if, for, switch-case, signal assignment
TAMPEREEN TEKNILLINEN YLIOPISTO
3


Section 1
CONCURRENT SIGNAL ASSIGNMENT STATEMENT
TAMPEREEN TEKNILLINEN YLIOPISTO
4

Architecture body

Simplified syntax

```
architecture arch_name of entity_name is
   declarations;
begin
   concurrent statement;
   concurrent statement;
   concurrent statement;
   . . .
end arch_name;
```

TAMPEREEN TEKNILLINEN YLIOPISTO

5

Simple Signal Assignment

Syntax:

signal_name <= projected_waveform;

– E.g.,

```
y <= a + b + 1 after 10 ns;
```

TAMPEREEN TEKNILLINEN YLIOPISTO

6

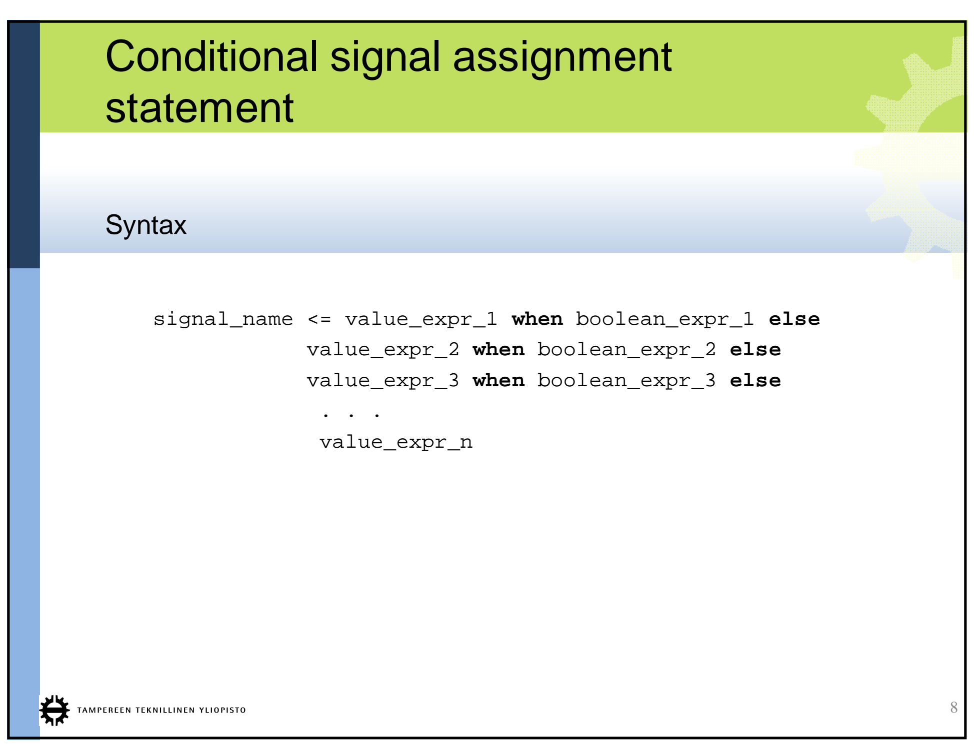
Examples

E.g.,
- status <= '1';
- even <= (p1 and p2) or (p3 and p4);
- arith_out <= a + b + c - 1;

Implementation of last statement

a
b
+
c
+
1
-
arith_result

TAMPEREEN TEKNILLINEN YLIOPISTO 7



E.g., 4-to-1 mux

input s	output x
0 0	a
0 1	b
1 0	c
1 1	d

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4 is
   port(
      a,b,c,d: in std_logic_vector(7 downto 0);
      s: in std_logic_vector(1 downto 0);
      x: out std_logic_vector(7 downto 0)
   );
end mux4 ;
architecture cond_arch of mux4 is
begin
   x <= a when (s="00") else
        b when (s="01") else
        c when (s="10") else
        d;
end cond_arch;
```

TAMPEREEN TEKNILLINEN YLIOPISTO

9

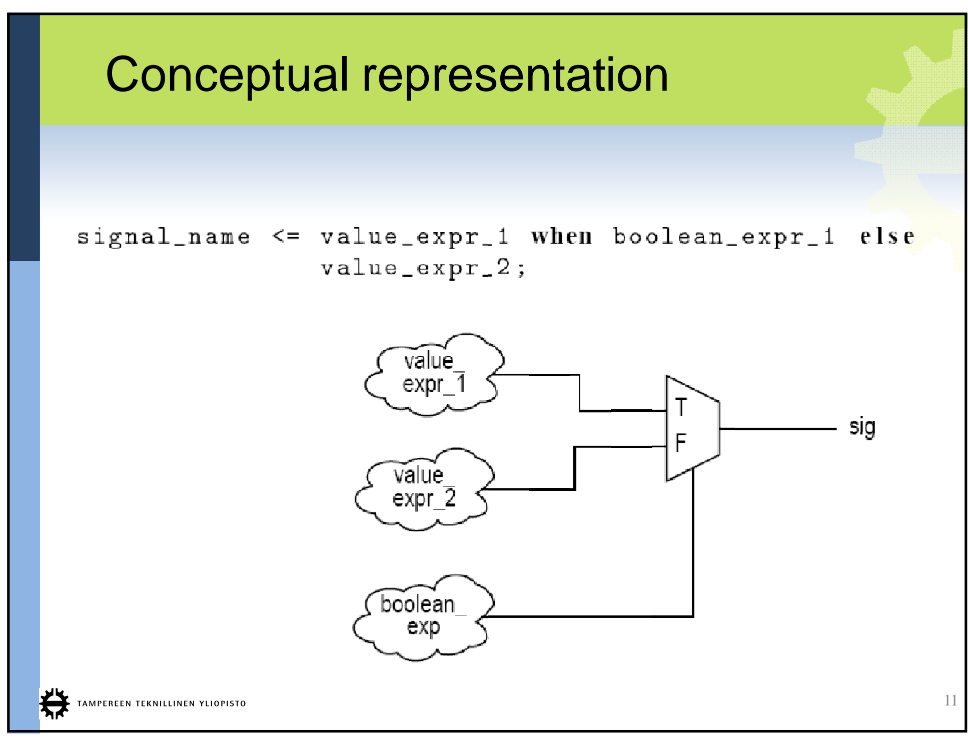
E.g., 4-to-2 priority encoder

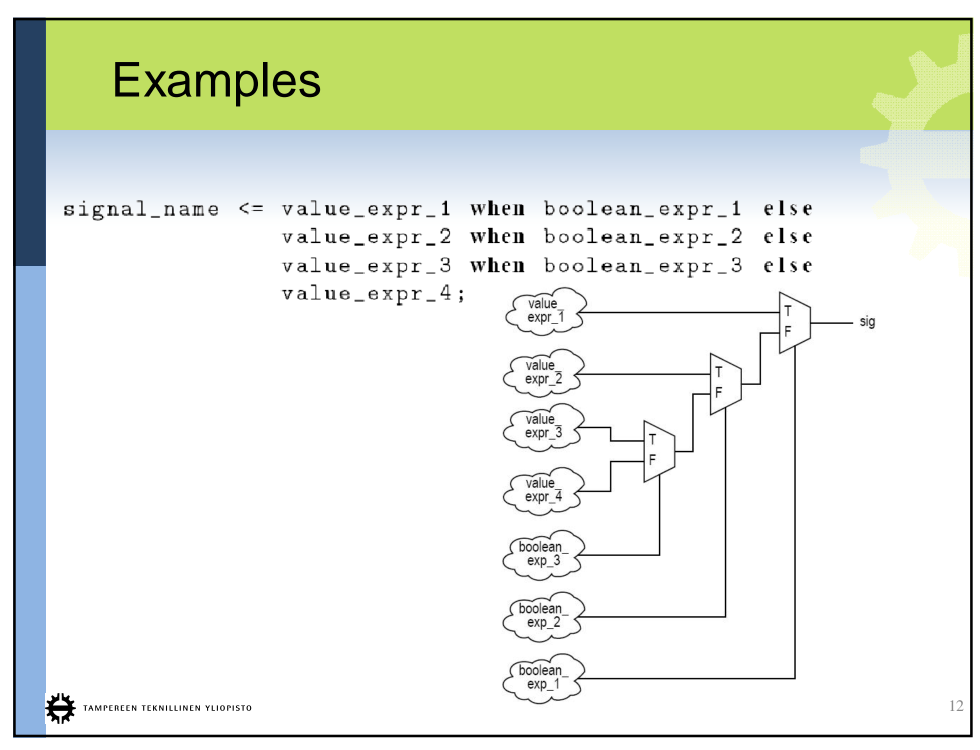
input r	output code	active
1 – – –	11	1
0 1 – –	10	1
0 0 1 –	01	1
0 0 0 1	00	1
0 0 0 0	00	0

```
entity prio_encoder42 is
   port(
      r: in std_logic_vector(3 downto 0);
      code: out std_logic_vector(1 downto 0);
      active: out std_logic
       );
end prio_encoder42;
architecture cond_arch of prio_encoder42 is
begin
   code <= "11" when (r(3)='1') else
           "10" when (r(2)='1') else
           "01" when (r(1)='1') else
           "00";
   active <= r(3) or r(2) or r(1) or r(0);
end cond_arch ;
```

TAMPEREEN TEKNILLINEN YLIOPISTO

10


Conceptual representation
signal_name <= value_expr_1 when boolean_expr_1 else
value_expr_2;
value_
expr_1
value_
expr_2
boolean_
exp
T
F
sig
TAMPEREEN TEKNILLINEN YLIOPISTO
11

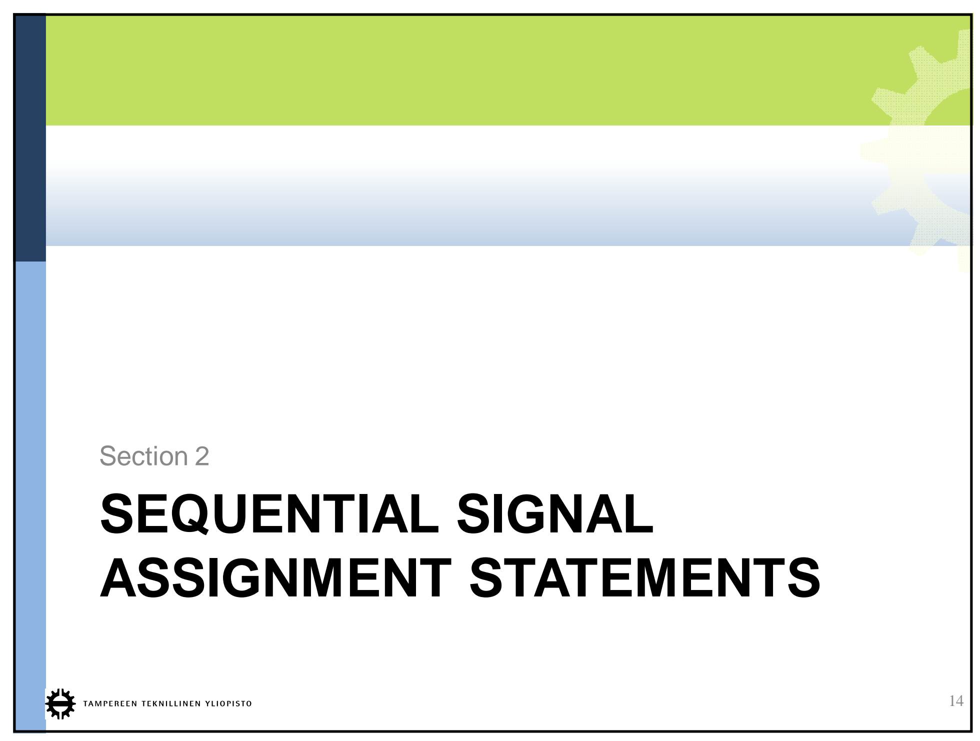

Examples
signal_name <= value_expr_1 when boolean_expr_1 else
value_expr_2 when boolean_expr_2 else
value_expr_3 when boolean_expr_3 else
value_expr_4;
value_
expr_1
value_
expr_2
value_
expr_3
value_
expr_4
boolean_
exp_3
boolean_
exp_2
boolean_
exp_1
T
F
sig
TAMPEREEN TEKNILLINEN YLIOPISTO
12

ALU

E.g..

```
signal a,b,r: unsigned(7 downto 0);
signal s: std_logic_vector(1 downto 0);
   . . .
   with s select
      r <= a+1    when "11",
           a-b-1  when "10",
           a+b    when others;
```

TAMPEREEN TEKNILLINEN YLIOPISTO 13



Sequential vs. Concurrent

```
architecture x of entity is
    declaration
    begin
    concurrent statement;
    concurrent statement;
    process()
      begin
      sequential statement;
      sequential statement;
      end
    concurrent statement;
    process()
      begin
      sequential statement;
      sequential statement;
      end
end x;
```

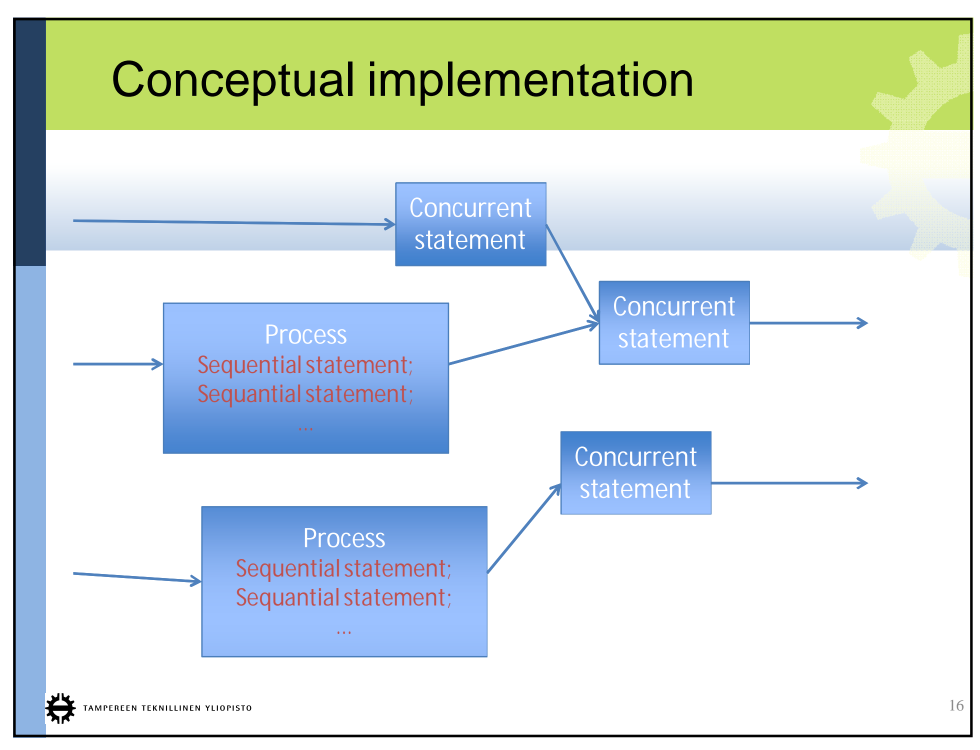
concurrent

sequential!

TAMPEREEN TEKNILLINEN YLIOPISTO

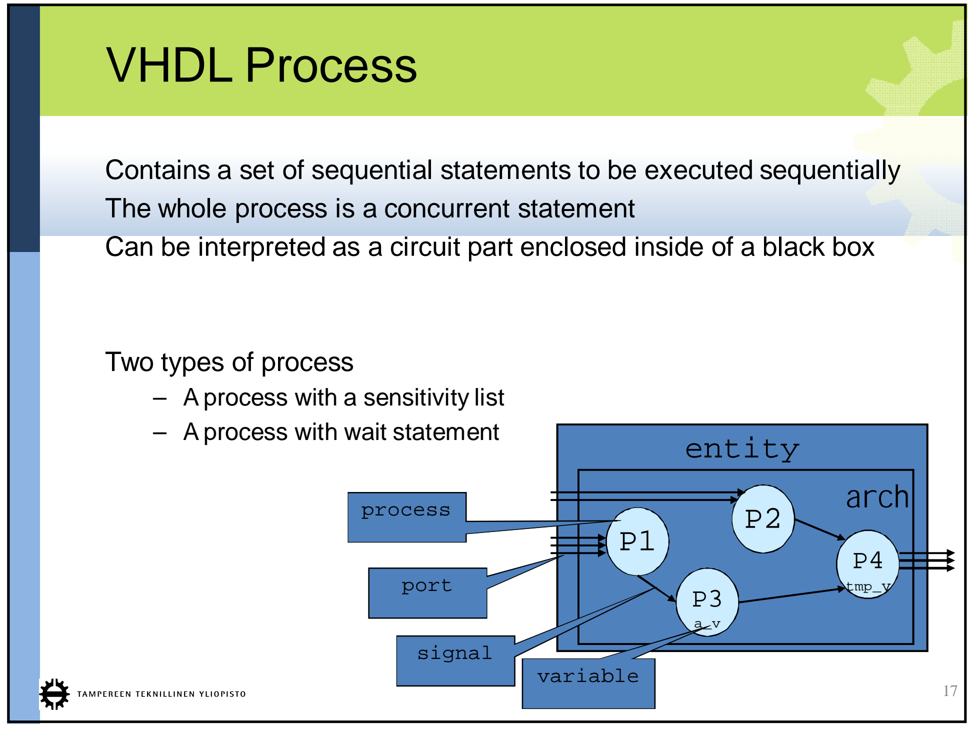
15

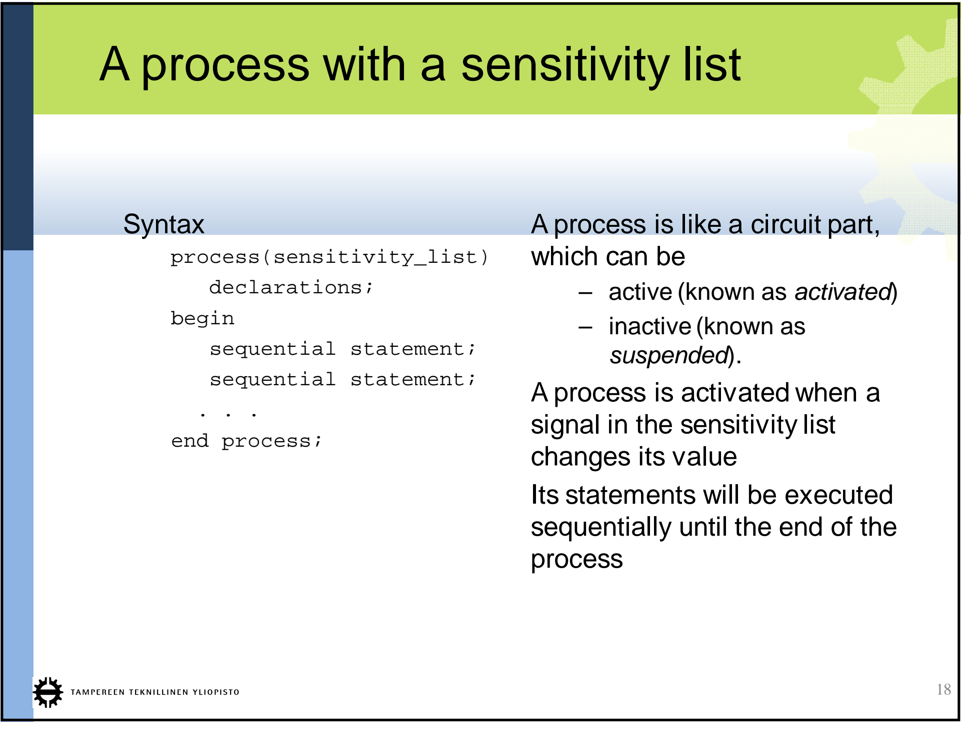
Conceptual implementation



TAMPEREEN TEKNILLINEN YLIOPISTO

16


VHDL Process
Contains a set of sequential statements to be executed sequentially
The whole process is a concurrent statement
Can be interpreted as a circuit part enclosed inside of a black box
Two types of process
– A process with a sensitivity list
– A process with wait statement
entity
arch
process
port
signal
variable
P1
P2
P3
P4
TAMPEREEN TEKNILLINEN YLIOPISTO
17


A process with a sensitivity list
Syntax
process(sensitivity_list)
declarations;
begin
sequential statement;
sequential statement;
. . .
end process;
A process is like a circuit part, which can be
– active (known as activated)
– inactive (known as suspended).
A process is activated when a signal in the sensitivity list changes its value
Its statements will be executed sequentially until the end of the process
TAMPEREEN TEKNILLINEN YLIOPISTO
18

Example

E.g, 3-input and circuit

```
signal a,b,c,y: std_logic;
process(a,b,c)
begin
   y <= a and b and c;
end process;
```

How to interpret this:

```
process(a)
begin
   y <= a and b and c;
end process;
```

For a combinational circuit, all input should be included in the sensitivity list

Sequential Signal Assignment Statement

Syntax

```
signal_name <= value_expression;
```

Syntax is identical to the simple concurrent signal assignment

Caution:

- Inside a process, a signal can be assigned multiple times, but only the last assignment takes effect

Examples

E.g.,

```
process(a,b,c,d)
Begin y <= a or c;
      y <= a and b;
      y <= c and d;
 end process;
```

It is same as

```
process(a,b,c,d)
begin
   y <= c and d;
end process;
```


IF statement

Syntax

```
if boolean_expr_1 then
   sequential_statements;
elsif boolean_expr_2 then
   sequential_statements;
elsif boolean_expr_3 then
   sequential_statements;
. . .
else
   sequential_statements;
end if;
```

E.g., 4-to-1 mux

input s	output x
0 0	a
0 1	b
1 0	c
1 1	d

```
architecture if_arch of mux4 is
begin
   process(a,b,c,d,s)
   begin
      if (s="00") then
         x <= a;
      elsif (s="01")then
         x <= b;
      elsif (s="10")then
         x <= c;
      else
         x <= d;
      end if;
   end process;
end if_arch;
```


Comparison to conditional signal assignment

Two statements are the same if there is only one output signal in if statement

If statement is more flexible

Sequential statements can be used in then, elsif and else branches:

- Multiple statements
- Nested if statements

```
sig <= value_expr_1 when boolean_expr_1 else
       value_expr_2 when boolean_expr_2 else
       value_expr_3 when boolean_expr_3 else
          . . .
       value_expr_n;
```

It can be written as

```
process(...)
   if boolean_expr_1 then
      sig <= value_expr_1;
   elsif boolean_expr_2 then
      sig <= value_expr_2;
   elsif boolean_expr_3 then
      sig <= value_expr_3;
   . . .
   else
      sig <= value_expr_n;
   end if;
end process
```

Incomplete branch and incomplete signal assignment

According to VHDL definition:

- Only the “then” branch is required; “elsif” and “else” branches are optional
- When a signal is unassigned due to omission, it keeps the “previous value” (implying “memory”)

Incomplete branch

E.g.,

```
process(a,b)
begin
   if (a=b) then
      eq <= '1';
   end if ;
end process;
```

It implies

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

Incomplete branch (cont'd)

Fix

```
process(a,b)
begin
    if (a=b) then
        eq <= '1';
    else
        eq <= '0';
    end if ;
end process
```


Incomplete signal assignment

E.g.,

```
process(a,b)
begin
    if (a>b) then
        gt <= '1';
    elsif (a=b) then
        eq <= '1';
    else
        lt <= '1';
    end if;
end process;
```

Fixes

```
process(a,b)
begin
   if (a>b) then
      gt <= '1';
      eq <= '0';
      lt <= '0';
   elsif (a=b) then
      gt <= '0';
      eq <= '1';
      lt <= '0';
   else
      gt <= '0';
      eq <= '0';
      lt <= '1';
   end if;
end process;
```

```
process(a,b)
begin
   gt <= '0';
   eq <= '0';
   lt <= '0';
   if (a>b) then
      gt <= '1';
   elsif (a=b) then
      eq <= '1';
   else
      lt <= '1';
   end if;
end process;
```


Conceptual implementation

Same as conditional signal assignment statement if the if statement consists of

- One output signal
- One sequential signal assignment in each branch

Multiple sequential statements can be constructed recursively

```
if boolean_expr then
   sig_a <= value_expr_a_1;
   sig_b <= value_expr_b_1
else
   sig_a <= value_expr_a_2;
   sig_b <= value_expr_b_2;
end if;
```

Case statement

Syntax

```
case case_expression is
   when choice_1 =>
      sequential statements;
   when choice_2 =>
      sequential statements;
   . . .
   when choice_n =>
      sequential statements;
end case;
```


E.g., 4-to-1 mux

input s	output x
0 0	a
0 1	b
1 0	c
1 1	d

```
architecture case_arch of mux4 is
begin
   process(a,b,c,d,s)
   begin
      case s is
         when "00" =>
            x <= a;
         when "01" =>
            x <= b;
         when "10" =>
            x <= c;
         when others =>
            x <= d;
      end case;
   end process;
end case_arch;
```

Comparison to selected signal assignment

Two statements are the same if there is only one output signal in case statement

Case statement is more flexible

Sequential statements can be used in choice branches

```
with sel_exp select
   sig <= value_expr_1 when choice_1,
          value_expr_2 when choice_2,
          value_expr_3 when choice_3,
          . . .
          value_expr_n when choice_n;
```

It can be rewritten as:

```
case sel_exp is
   when choice_1 =>
      sig <= value_expr_1;
   when choice_2 =>
      sig <= value_expr_2;
   when choice_3 =>
      sig <= value_expr_3;
   . . .

   when choice_n =>
      sig <= value_expr_n;
end case;
```


Incomplete signal assignment

According to VHDL definition:

- When a signal is unassigned, it keeps the “previous value” (implying “memory”)

```
process(a)
   case a is
      when "100"|"101"|"110"|"111" =>
         high <= '1';
      when "010"|"011" =>
         middle <= '1';
      when others =>
         low <='1';
   end case;
end process;
```

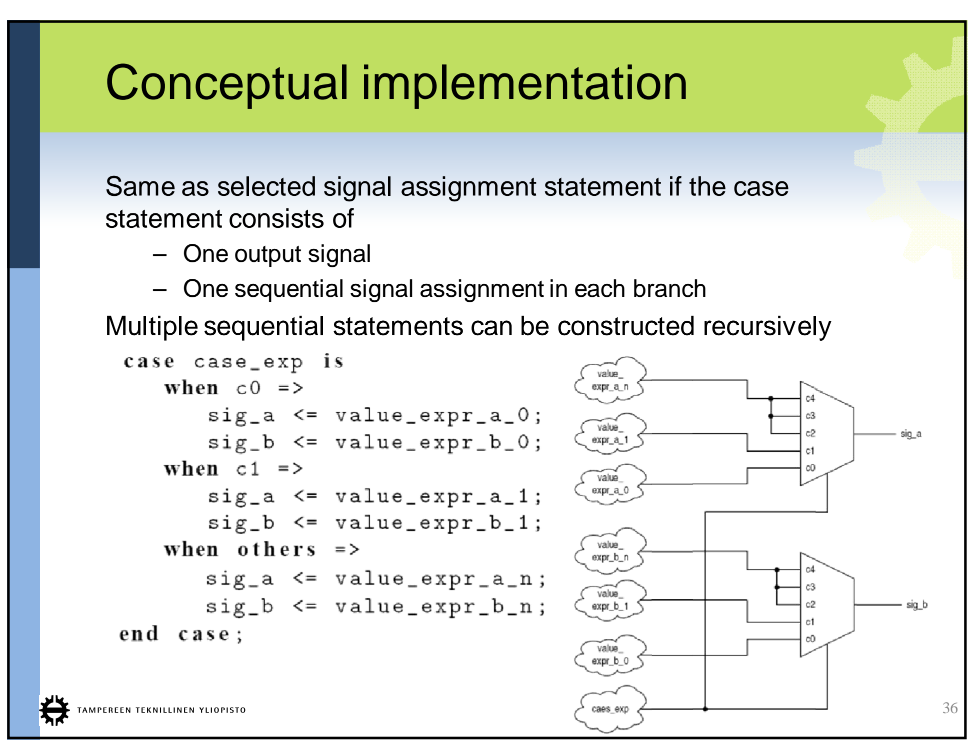
Fixes

```
process(a)
   case a is
      when "100"|"101"|"110"|"111" =>
         high <= '1';
         middle <= '0';
         low <= '0';
      when "010"|"011" =>
         high <= '0';
         middle <= '1';
         low <= '0';
      when others =>
         high <= '0';
         middle <= '0';
         low <= '1';
      end case;
   end process;
```

```
process(a)
   high <= '0';
   middle <= '0';
   low <= '0';
   case a is
      when "100"|"101"|"110"|"111" =>
         high <= '1';
      when "010"|"011" =>
         middle <= '1';
      when others =>
         low <='1';
   end case;
end process;
```

TAMPEREEN TEKNILLINEN YLIOPISTO

35



Simple for loop statement

VHDL provides a variety of loop constructs

Only a restricted form of loop can be synthesized

Syntax of simple for loop:

```
for index in loop_range loop
    sequential statements;
end loop;
```

loop_range must be static

Index assumes value of loop_range from left to right

Loops: example

Hardware:

```
add_i: for i in 0 to max_c-1 generate
    b(i) <= a(i) + i;
end generate add_i;
```

Generates <max_c> parallel computation units

E.g., bit-wide xor

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

entity wide_xor is
   port(
      a, b: in std_logic_vector(3 downto 0);
      y: out std_logic_vector(3 downto 0)
   );
end wide_xor;

architecture demo_arch of wide_xor  is
   constant WIDTH: integer := 4;
begin
   process(a, b)
   begin
      for i in (WIDTH-1) downto 0 loop
         y(i) <= a(i) xor b(i);
      end loop;
   end process;
end demo_arch;
```


Conceptual implementation

"Unroll" the loop
For loop should be treated as "shorthand" for repetitive statements
E.g., bit-wise xor

```
y(3) <= a(3) xor b(3);
y(2) <= a(2) xor b(2);
y(1) <= a(1) xor b(1);
y(0) <= a(0) xor b(0);
```

E.g., reduced-xor

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

entity reduced_xor_demo is
   port(
      a: in std_logic_vector(3 downto 0);
      y: out std_logic
   );
end reduced_xor_demo;

architecture demo_arch of reduced_xor_demo is
   constant WIDTH: integer := 4;
   signal tmp: std_logic_vector(WIDTH-1 downto 0);
begin
   process(a,tmp)
   begin
      tmp(0) <= a(0);    -- boundary bit
      for i in 1 to (WIDTH-1) loop
         tmp(i) <= a(i) xor tmp(i-1);
      end loop;
   end process;
   y <= tmp(WIDTH-1);
end demo_arch;
```


Summary

Concurrent vs. sequential statement

Concurrent
- simple assignment
- selected assignment
- conditional assignment

Sequential
- Processes
- If statement
- Case statement
- Loop statement