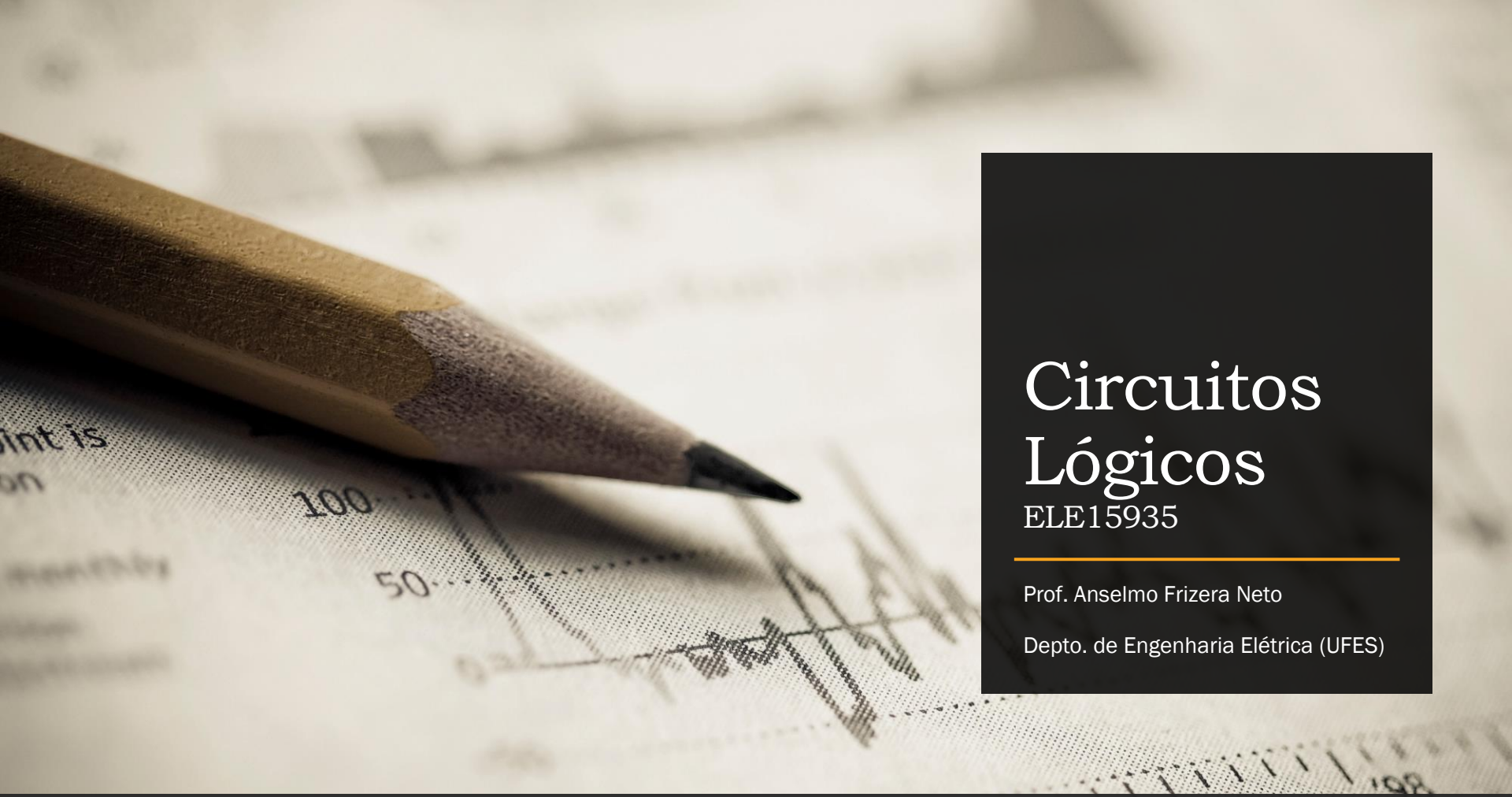


Circuitos Lógicos

ELE15935

Prof. Anselmo Frizera Neto

Depto. de Engenharia Elétrica (UFES)

Lab. 3. Roteamento com Instrução de Atribuição Concorrente

Tópicos da aula de hoje:

- With … select
- When
- Process
- If
- Case

Bibliografia:

- P. Chu, “FPGA Prototyping by VHDL Examples”, Capítulo 3

Overview

Many designs require to select a result from different alternatives based on a given condition

It can be done by routing in hardware

Concurrent VHDL statement

- Conditional signal assignment: use priority routing network (somewhat like if-then-else)
- Selected signal assignment: use multiplexing routing network (somewhat like case)

Conditional Signal Assignment

Overall effect somewhat like if-then-else

Simplified syntax:

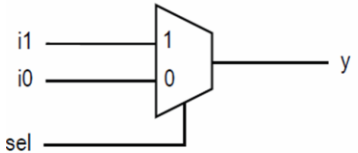
```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2 when boolean_expr_2 else
                 . . .
               value_expr_n;
```

Evaluation in ascending order

Boolean expressions and value expressions

- Implemented by hardware circuits
- Evaluated in parallel

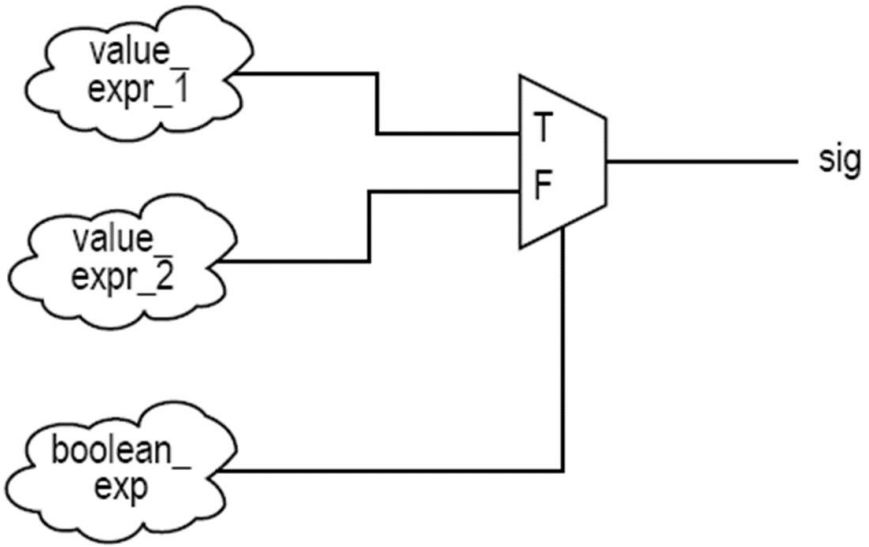
“Priority-routing network”



sel	y
0 (false)	i0
1 (true)	i1

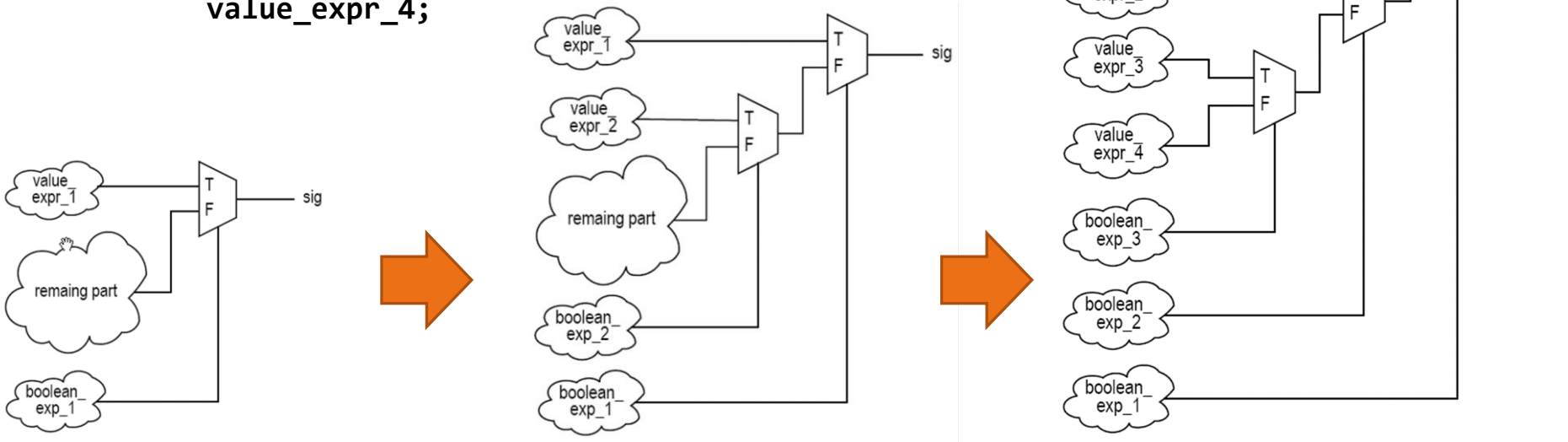
Conceptual implementation

```
sig <= value_expr_1 when boolean_expr else
       value_expr_2;
```



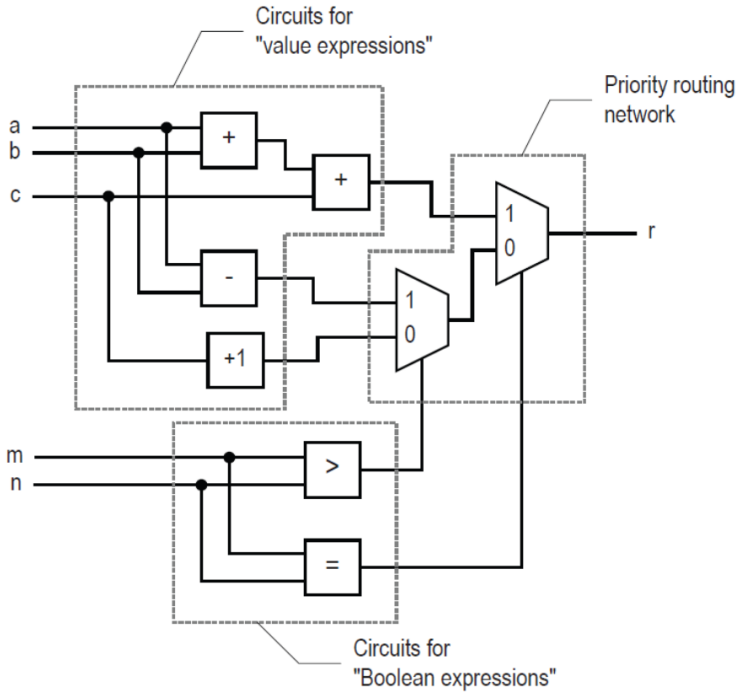
Conceptual implementation (cont.)

```
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;
```



Conceptual implementation

```
r <=  a + b + c when m = n else
              a - b      when m > n else
              c + 1;
```



Example: 4-request priority encoder

input r	output pcode
1 – – –	100
0 1 – –	011
0 0 1 –	010
0 0 0 1	001
0 0 0 0	000

```
library ieee;
use ieee.std_logic_1164.all;
entity prio_encoder is
   port(
      r     : in  std_logic_vector(4 downto 1);
      pcode : out std_logic_vector(2 downto 0)
   );
end prio_encoder;

architecture cond_arch of prio_encoder is
begin
   pcode <= "100" when (r(4) = '1') else
            "011" when (r(3) = '1') else
            "010" when (r(2) = '1') else
            "001" when (r(1) = '1') else
            "000";
end cond_arch;
```

Example: 2-to-4 decoder with enable

input			output
en	a(1)	a(0)	y
0	–	–	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000

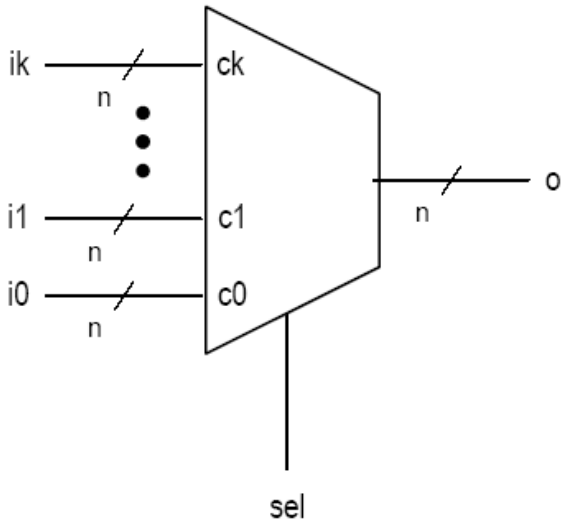
```
library ieee;
use ieee.std_logic_1164.all;
entity decoder_2_4 is
   port(
      a  : in  std_logic_vector(1 downto 0);
      en : in  std_logic;
      y  : out std_logic_vector(3 downto 0)
   );
end decoder_2_4;

architecture cond_arch of decoder_2_4 is
begin
   y <= "0000" when (en = '0') else
        "0001" when (a = "00") else
        "0010" when (a = "01") else
        "0100" when (a = "10") else
        "1000";    -- a="11"
end cond_arch;
```

Selected signal assignment

Overall effect somewhat like case

Simplified syntax:

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



select_expression

- Discrete type or 1-D array
- With finite possible values

choice_i

- A value of the data type

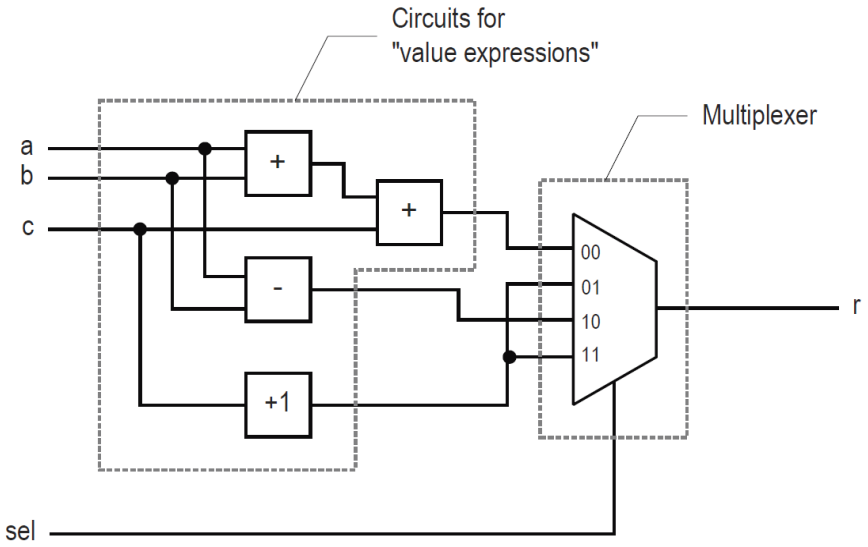
Choices must be

- mutually exclusive, all inclusive, others can be used as last choice_i

Conceptual implementation

E.g., select_expression w/ 5 choices:

```
signal sel: std_logic_vector(1 downto 0);
. . .
with sel select
   r <= a + b + c when "00",
        a - b     when "10",
        c + 1     when others;
```



Example: 4-request priority encoder

input r	output pcode
1 – – –	100
0 1 – –	011
0 0 1 –	010
0 0 0 1	001
0 0 0 0	000

```
architecture sel_arch of prio_encoder is
begin
   with r select
      pcode <= "100" when "1000"|"1001"|"1010"|"1011"|
                          "1100"|"1101"|"1110"|"1111",
               "011" when "0100"|"0101"|"0110"|"0111",
               "010" when "0010"|"0011",
               "001" when "0001",
               "000" when others;    -- r="0000"
end sel_arch;
```

Caveat

Can “1 – – –” be used to replace “1000” | “1001” | “1010” | … | “1111”?
→ No. ‘–’ interpreted as a specific symbolic value and should be avoided

```
architecture sel_arch of prio_encoder is
begin
   with r select
      pcode <= "100" when "1000"|"1001"|"1010"|"1011"|
                          "1100"|"1101"|"1110"|"1111",
               "011" when "0100"|"0101"|"0110"|"0111",
               "010" when "0010"|"0011",
               "001" when "0001",
               "000" when others;    -- r="0000"
end sel_arch;
```

Can “0000” be used in place of others?
→ No

E.g., 2-to-4 decoder with enable

	input		output
en	a(1)	a(0)	y
0	–	–	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000

```
architecture sel_arch of decoder_2_4 is
   signal s: std_logic_vector(2 downto 0);
begin
   s <= en & a;
   with s select
      y <= "0000" when "000"|"001"|"010"|"011",
           "0001" when "100",
           "0010" when "101",
           "0100" when "110",
           "1000" when others;    -- s="111"
end sel_arch;
```

Modeling with Process

Overview

A process:

- contains a set of sequential statements to be executed sequentially
- is a concurrent statement
- can be interpreted as a circuit part enclosed inside of a black box
- may or may not be able to be mapped to physical hardware

Two types of process

- A process with a sensitivity list
- A process with wait statement

```
process(sensitivity_list)
begin
    sequential statement;
    sequential statement;
   . . .
end process;
```

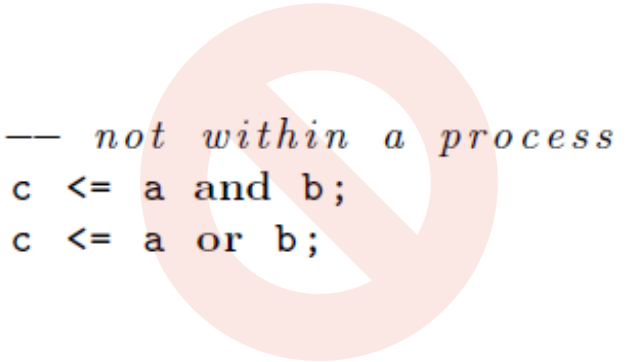
A process is activated when a signal in the sensitivity list changes its value
Its statements will be executed sequentially until the end of the process

Example

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

=

```
process(a,b)
begin
    c <= a or b;
end process;
```



```
-- not within a process
c <= a and b;
c <= a or b;
```

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;
```

```
r <= a + b + c when m = n else
     a - b     when m > 0 else
     c + 1;
```

```
process(a,b,c,m,n)
begin
   if m = n then
      r <= a + b + c;
   elsif m > 0 then
      r <= a - b;
   else
     r <= c + 1;
   end if;
end;
```

If statement

Multiple sequential statements can be included in a branch of an if statements; e.g.,

- Multiple signal assignments
- Nested if statements

They may infer complex priority routing structure

Example: 4-request priority encoder

input r	output pcode
1 – – –	100
0 1 – –	011
0 0 1 –	010
0 0 0 1	001
0 0 0 0	000

```
architecture if_arch of prio_encoder is
begin
   process(r)
   begin
      if (r(4) = '1') then
         pcode <= "100";
      elsif (r(3) = '1') then
         pcode <= "011";
      elsif (r(2) = '1') then
         pcode <= "010";
      elsif (r(1) = '1') then
         pcode <= "001";
      else
         pcode <= "000";
      end if;
   end process;
end if_arch;
```

Example: 2-to-4 decoder with enable

	input		output
en	a(1)	a(0)	y
0	–	–	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000

```
architecture if_arch of decoder_2_4 is begin
   process(en,a)
   begin
      if (en='0') then
         y <= "0000";
      elsif (a="00") then
         y <= "0001";
      elsif (a="01")then
         y <= "0010";
      elsif (a="10")then
         y <= "0100";
      else
         y <= "1000";
      end if;
   end process;
end if_arch;
```

Case statement

Syntax:

```
case sel is
   when choice_1 =>
      sequential statements;
   when choice_2 =>
      sequential statements;
   . . .
   when others =>
      sequential statements;
end case;
```

```
with sel select
   r <= a + b + c   when "00",
        a - b       when "10",
        c + 1       when others;
```

```
process(a,b,c,sel)
begin
   case sel is
      when "00" =>
         r <= a + b + c;
      when "10" =>
         r <= a - b;
      when others =>
         r <= c + 1;
   end case;
end;
```

Case statement (cont.)

Multiple sequential statements can be included in a choice branch of case statements; e.g.,

- Multiple signal assignments
- If statement
- Another case statement

They may infer complex routing structure

Example: 4-request priority encoder

input r	output pcode
1 – – –	100
0 1 – –	011
0 0 1 –	010
0 0 0 1	001
0 0 0 0	000

```
architecture case_arch of prio_encoder is
begin
   process(r)
   begin
      case r is
         when "1000"|"1001"|"1010"|"1011"|
              "1100"|"1101"|"1110"|"1111" =>
            pcode <= "100";
         when "0100"|"0101"|"0110"|"0111" =>
            pcode <= "011";
         when "0010"|"0011" =>
            pcode <= "010";
         when "0001" =>
            pcode <= "001";
         when others =>
            pcode <= "000";
      end case;
   end process;
end case_arch;
```

Example: 2-to-4 decoder with enable

input			output
en	a(1)	a(0)	y
0	–	–	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000

```
architecture case_arch of decoder_2_4 is
   signal s: std_logic_vector(2 downto 0);
begin
   s <= en & a;
   process(s)
   begin
      case s is
         when "000"|"001"|"010"|"011" =>
            y <= "0000";
         when "100" =>
            y <= "0001";
         when "101" =>
            y <= "0010";
         when "110" =>
            y <= "0100";
         when others =>
            y <= "1000";
      end case;
   end process;
end case_arch;
```

Comparison to concurrent statement

Simple if statement infers a priority-routing network
(like conditional signal assignment)

Simple case statement infers a multiplexing network
(like selected signal assignment)

But if/case statements are more flexible and versatile

Use of if/case statements

- Think them as routing strucutre
- Careless use may lead to overly complicated or even unsynthesizable routing structure
- Disciplined use can make code more descriptive and infer “sharing”

Example: “sharing” of a boolean expression circuit

Two comparators are inferred

```
large <= a when a > b else b;
small <= b when a > b else a;
```

One comparator is inferred and shared

```
process(a,b)
begin
   if a > b then
      large <= a;
      small <= b;
   else
      large <= b;
      small <= a;
   end if;
end;
```

Example: max circuit (find max of a, b, and c)

Using process (easy to understand)

```
process(a,b,c)
begin
   if (a > b) then
      if (a > c) then
         max <= a;
      else
         max <= c;
      end if;
   else
      if (b > c) then
         max <= b;
      else
         max <= c;
      end if;
   end if;
end process;
```

Using concurrent statements (less clear)

```
max <= a when ((a > b) and (a > c)) else
       c when (a > b) else
       b when (b > c) else
       c;
```

Using concurrent statements (better)

```
signal ac_max, bc_max: std_logic;
. . .
ac_max <= a when (a > c) else c;
bc_max <= b when (b > c) else c;
max    <= ac_max when (a > b) else bc_max;
```

Unintended memory

A signal keeps its “previous value” if it is not assigned in a process

This infers an unintended “memory” component (in form of a “loop” or a latch during synthesis)

It is a serious design error since a combinational circuit should not contain an internal state (“memory”)

Avoid unintended memory:

- Include all input signals in the sensitivity list
- Include the else branch in an if statement
- Assign a value to a every signal in every branch of an if or case statement

An easy way to do is to assign a default value to every signal in the beginning of the process

Unintended memory

```
process(a)              -- b missing from sensitivity list
begin
   if (a > b) then      -- eq not assigned in this branch
      gt <= '1';
   elsif (a = b) then   -- gt not assigned in this branch
      eq <= '1';
   end if;              -- else branch is omitted
end process;
```

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

Unintended memory

```
process(a)              -- b missing from sensitivity list
begin
   if (a > b) then      -- eq not assigned in this branch
      gt <= '1';
   elsif (a = b) then   -- gt not assigned in this branch
      eq <= '1';
   end if;              -- else branch is omitted
end process;
```

```
process(a,b)
begin
   gt <= '0';           -- assign default value
   eq <= '0';
   if (a > b) then
      gt <= '1';
   elsif (a = b) then
      eq <= '1';
   end if;
end process;
```