Concurrent Signal Assignment Statements
Outline

1. Combinational versus sequential circuit
2. Simple signal assignment statement
3. Conditional signal assignment statement
4. Selected signal assignment statement
5. Conditional vs. selected signal assignment
1. Combinational vs. sequential circuit

- **Combinational circuit:**
  - No internal state
  - Output is a function of inputs only
  - No latches/FFs or closed feedback loop

- **Sequential circuit:**
  - With internal state
  - Output is a function of inputs and internal state

- **Sequential circuit to be discussed later**
2. Simple signal assignment statement
• Simple signal assignment is a special case of conditional signal assignment

• Syntax:
  signal_name <= projected_waveform;

• E.g.,
  y <= a + b + 1 after 10 ns;

• Timing info ignored in synthesis and δ-delay is used:
  signal_name <= value_expression
• E.g.,
  status <= '1';
  even <= (p1 and p2) or (p3 and p4);
  arith_out <= a + b + c - 1;
• Implementation of last statement
Signal assignment statement with a closed feedback loop

- a signal appears in both sides of a concurrent assignment statement
- E.g.,
  \[ q <= ((\text{not } q) \text{ and } (\text{not } en)) \text{ or } (d \text{ and } en); \]
- Syntactically correct
- Form a closed feedback loop
- Should be avoided
3. Conditional signal assignment statement

- Syntax
- Examples
- Conceptual implementation
- Detailed implementation examples
Syntax

• Simplified syntax:

```plaintext
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_n
```
E.g., 4-to-1 mux

• Function table:

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>a</td>
</tr>
<tr>
<td>0 1</td>
<td>b</td>
</tr>
<tr>
<td>1 0</td>
<td>c</td>
</tr>
<tr>
<td>1 1</td>
<td>d</td>
</tr>
</tbody>
</table>
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;
E.g., 2-to-2\(^2\) binary decoder

- Function table:

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0001</td>
</tr>
<tr>
<td>0 1</td>
<td>0010</td>
</tr>
<tr>
<td>1 0</td>
<td>0100</td>
</tr>
<tr>
<td>1 1</td>
<td>1000</td>
</tr>
</tbody>
</table>
library ieee;
use ieee.std_logic_1164.all;
entity decoder4 is
  port(
    s: in std_logic_vector(1 downto 0);
    x: out std_logic_vector(3 downto 0)
  );
end decoder4;
architecture cond_arch of decoder4 is
begin
  x <= "0001" when (s="00") else "0010" when (s="01") else "0100" when (s="10") else "1000";
end cond_arch;
E.g., 4-to-2 priority encoder

• Function table:

<table>
<thead>
<tr>
<th>input r</th>
<th>output code</th>
<th>active</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0 1-</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1-</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>00</td>
<td>0</td>
</tr>
</tbody>
</table>
library ieee;
use ieee.std_logic_1164.all;

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;
E.g., simple ALU

- Function table:

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctrl</td>
<td>result</td>
</tr>
<tr>
<td>0</td>
<td>src0 + 1</td>
</tr>
<tr>
<td>100</td>
<td>src0 + src1</td>
</tr>
<tr>
<td>101</td>
<td>src0 - src1</td>
</tr>
<tr>
<td>110</td>
<td>src0 and src1</td>
</tr>
<tr>
<td>111</td>
<td>src0 or src1</td>
</tr>
</tbody>
</table>
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity simple_alu is
  port(
    ctrl: in std_logic_vector(2 downto 0);
    src0, src1: in std_logic_vector(7 downto 0);
    result: out std_logic_vector(7 downto 0)
  );
end simple_alu;

architecture cond_arch of simple_alu is
  signal sum, diff, inc: std_logic_vector(7 downto 0);
begin
  inc <= std_logic_vector(signed(src0)+1);
  sum <= std_logic_vector(signed(src0)+signed(src1));
  diff <= std_logic_vector(signed(src0)-signed(src1));
  result <= inc when ctrl(2)='0' else
    sum when ctrl(1 downto 0)="00" else
    diff when ctrl(1 downto 0)="01" else
    src0 and src1 when ctrl(1 downto 0)="10" else
    src0 or src1;
end cond_arch;
Conceptual implementation

• Syntax:
  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_n;

• Evaluation in ascending order
• Achieved by “priority-routing network”
• Top value expression has a “higher priority”
2-to-1 “abstract” mux

- sel has a data type of boolean
- If sel is true, the input from “T” port is connected to output.
- If sel is false, the input from “F” port is connected to output.
signal_name <= value_expr_1 when boolean_expr_1 else value_expr_2;
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;

RTL Hardware Design  Chapter 4
Detailed implementation examples

• 2-to-1 mux

![2-to-1 mux diagram]
- E.g.,

```vhdl
signal a, b, y: std_logic;

y <= '0' when a = b else
     '1';
```

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
</tr>
</tbody>
</table>
• E.g.,

```vhdl
signal a, b, c, x, y, r: std_logic;
...
    r <= a when x = y else
    b when x > y else
    c;
```

```text
= >
```

```text
TF
```

```text
TF
```
E.g.,

```verilog
signal a, b, r: unsigned(7 downto 0);
signal x, y: unsigned(3 downto 0);

r <= a+b when x+y>1 else
   a-b-1 when x>y and y!=0 else
   a+1;
```
4. Selected signal assignment statement

- Syntax
- Examples
- Conceptual implementation
- Detailed implementation examples
Syntax

• Simplified syntax:

```pascal
with select_expression select
    signal_name <=
        value_expr_1 when choice_1,
        value_expr_2 when choice_2,
        value_expr_3 when choice_3,
        . . .
        value_expr_n when choice_n;
```
• 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
E.g., 4-to-1 mux

```vhdl
architecture sel_arch of mux4 is
begin
    with s select
    begin
        x <= a when "00",
            b when "01",
            c when "10",
            d when others;
    end sel_arch;
end sel_arch;
```

<table>
<thead>
<tr>
<th>s</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>a</td>
</tr>
<tr>
<td>01</td>
<td>b</td>
</tr>
<tr>
<td>10</td>
<td>c</td>
</tr>
<tr>
<td>11</td>
<td>d</td>
</tr>
</tbody>
</table>
• Can “11” be used to replace others?

```vhdl
with s select
  x <= a when "00",
    b when "01",
    c when "10",
    d when "11";
```
E.g., $2$-to-$2^2$ binary decoder

```vhdl
architecture sel_arch of decoder4 is
begin
    with sel select
    begin
        x <= "0001" when "00",
             "0010" when "01",
             "0100" when "10",
             "1000" when others;
    end sel_arch;
end;
```

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0001</td>
</tr>
<tr>
<td>0 1</td>
<td>0010</td>
</tr>
<tr>
<td>1 0</td>
<td>0100</td>
</tr>
<tr>
<td>1 1</td>
<td>1000</td>
</tr>
</tbody>
</table>
E.g., 4-to-2 priority encoder

```vhdl
architecture sel_arch of prio_encoder42 is
begin
  with r select
  code <= "11" when "1000"|"1001"|"1010"|"1011"|
                     "1100"|"1101"|"1110"|"1111",
       "10" when "0100"|"0101"|"0110"|"0111",
       "01" when "0010"|"0011",
       "00" when others;
  active <= r(3) or r(2) or r(1) or r(0);
end sel_arch;
```

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>code</td>
</tr>
<tr>
<td>r</td>
<td></td>
</tr>
<tr>
<td>1 -- -</td>
<td>11</td>
</tr>
<tr>
<td>0 1 --</td>
<td>10</td>
</tr>
<tr>
<td>0 0 1-</td>
<td>01</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>00</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>00</td>
</tr>
</tbody>
</table>
Can we use ‘-’?

```vhdl
with a select
    x <= "11" when "1---", "10" when "01--", "01" when "001-", "00" when others;
```
E.g., simple ALU

architecture sel_arch of simple_alu is

  signal sum, diff, inc: std_logic_vector(7 downto 0);

begin

  inc <= std_logic_vector(signed(src0)+1);
  sum <= std_logic_vector(signed(src0)+signed(src1));
  diff <= std_logic_vector(signed(src0)-signed(src1));

  with ctrl select
"when "000" | "001" | "010" | "011"," when "100",
"  when "101",
"  when "110",
"  when others;

end sel_arch;

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>src0 + 1</td>
</tr>
<tr>
<td>001</td>
<td>src0 + src1</td>
</tr>
<tr>
<td>010</td>
<td>src0 - src1</td>
</tr>
<tr>
<td>011</td>
<td>src0 and src1</td>
</tr>
<tr>
<td>100</td>
<td>src0 or src1</td>
</tr>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
</tr>
</tbody>
</table>
E.g., Truth table

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity truth_table is
    port(
        a, b: in std_logic;
        y: out std_logic
    );
end truth_table;
architecture a of truth_table is
    signal tmp: std_logic_vector(1 downto 0);
begin
    tmp <= a & b;
    with tmp select
    begin
        y <= '0' when "00",
            '1' when "01",
            '1' when "10",
            '1' when others; -- "11"
    end a:
```

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>
Conceptual implementation

- Achieved by a multiplexing circuit
- Abstract \((k+1)\)-to-1 multiplexer
  - sel is with a data type of \((k+1)\) values: \(c_0, c_1, c_2, \ldots, c_k\)
– select_expression is with a data type of 5 values: c0, c1, c2, c3, c4

\[
\begin{align*}
\text{with } \text{select_expression} & \text{ select} \\
\text{sig } \leq \text{value_expr}_0 & \text{ when } \text{c0}, \\
\text{value_expr}_1 & \text{ when } \text{c1}, \\
\text{value_expr}_n & \text{ when } \text{others};
\end{align*}
\]
Detailed implementation examples

• 4-to-1 mux

![4-to-1 multiplexer circuit diagram]
• E.g.,

```vhdl
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;
```

![Diagram of a hardware design](image)
3. Conditional vs. selected signal assignment

• Conversion between conditional vs. selected signal assignment
• Comparison
From selected assignment to conditional assignment

```vhdl
with sel select
  sig <= value_expr_0 when c0,
       value_expr_1 when c1|c3|c5,
       value_expr_2 when c2|c4,
       value_expr_n when others;
```

```vhdl
sig <=
  value_expr_0 when (sel=c0) else
  value_expr_1 when (sel=c1) or (sel=c3) or (sel=c5) else
  value_expr_2 when (sel=c2) or (sel=c4) else
  value_expr_n;
```
From conditional assignment to selected assignment

```vhdl
sig <= value_expr_0 when bool_exp_0 else
     value_expr_1 when bool_exp_1 else
     value_expr_2 when bool_exp_2 else
     value_expr_n;

sel(2) <= '1' when bool_exp_0 else '0';
sel(1) <= '1' when bool_exp_1 else '0';
sel(0) <= '1' when bool_exp_2 else '0';
with sel select
  sig <= value_expr_0 when "100" | "101" | "110" | "111",
       value_expr_1 when "010" | "011",
       value_expr_2 when "001",
       value_expr_n when others;
```
Comparison

• Selected signal assignment:
  – good match for a circuit described by a functional table
  – E.g., binary decoder, multiplexer
  – Less effective when an input pattern is given a preferential treatment
• Conditional signal assignment:
  – good match for a circuit that needs to give preferential treatment for certain conditions or to prioritize the operations
  – E.g., priority encoder
  – Can handle complicated conditions. e.g.,

```vhdl
pc_next <=
  pc_reg + offset when (state=jump and a=b) else
pc_reg + 1 when (state=skip and flag='1') else
...```


– May “over-specify” for a functional table based circuit.

– E.g., mux

\[
x <= \begin{cases} 
a & \text{when } (s = "00") \\
b & \text{when } (s = "01") \\
c & \text{when } (s = "10") \\
d & \text{else}
\end{cases}
\]

\[
x <= \begin{cases} 
c & \text{when } (s = "10") \\
a & \text{when } (s = "00") \\
b & \text{when } (s = "01") \\
d & \text{else}
\end{cases}
\]

\[
x <= \begin{cases} 
c & \text{when } (s = "10") \\
b & \text{when } (s = "01") \\
a & \text{when } (s = "00") \\
d & \text{else}
\end{cases}
\]