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:
  
  ```
  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>s</td>
<td>x</td>
</tr>
<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>

E.g., 2-to-2^2 binary decoder

- Function table:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>x</td>
</tr>
<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

- Function table:

<table>
<thead>
<tr>
<th>r</th>
<th>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</td>
<td>00</td>
<td>0</td>
</tr>
</tbody>
</table>

E.g., simple ALU

- Function table:

<table>
<thead>
<tr>
<th>ctrl</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 --</td>
<td>src0 + 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>src0 + src1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>src0 - src1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>src0 and src1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>src0 or src1</td>
</tr>
</tbody>
</table>

Conceptual implementation

- Syntax:

```vhdl
signal_name <= value_expr_1 when boolean_expr_1
              value_expr_2 when boolean_expr_2
              value_expr_3 when boolean_expr_3
              ... 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 when boolean_expr_2 else value_expr_3 when boolean_expr_3 else value_expr_4;
```

Detailed implementation examples

- 2-to-1 mux

```
signal a,b,y: std_logic;
y <= '0' when a#b else '1';
```

- E.g.,

```
 0 1
1 1
```

Input | Output
-----|-----
00 | 1
01 | 0
10 | 0
11 | 1
4. Selected signal assignment statement

- Syntax
- Examples
- Conceptual implementation
- Detailed implementation examples

Syntax

- Simplified syntax:
  ```
  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
    x <= a when "00",
        b when "01",
        c when "10",
        others;
end sel_arch;
```

• Can "11" be used to replace others?

```vhdl
architecture sel_arch of mux4 is begin
    with s select
    x <= a when "00",
        b when "01",
        c when "10",
        d when "11";
end sel_arch;
```

E.g., 2-to-2² binary decoder

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

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

```vhdl
architecture sel_arch of prio_encoder42 is begin
    with x select
    code <= "11" when "1000"|"1001"|"1010"|"1011",
        "1100"|"1101"|"1110"|"1111",
        "01" when "0001"|"0010"|"0101"|"0111",
        "00" when others;
active <= r(3) or r(2) or r(1) or r(0);
end sel_arch;
```

E.g., simple ALU

```vhdl
architecture sel_arch of simple_alu is 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
    result <= inc when "0001"|"0010"|"0111",
            sum when "100",
            src0 and src1 when "110",
            src0 or src1 when others;
end sel_arch;
```

• Can we use '-'?

```vhdl
architecture sel_arch of simple_alu is begin
    with a select
    x <= "11" when "1---",
        "10" when "01--",
        "01" when "001-",
        "00" when others;
end sel_arch;
```

E.g., simple ALU

```vhdl
architecture sel_arch of simple_alu is 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
    result <= inc when "0001"|"0010"|"0111",
            sum when "100",
            src0 and src1 when "110",
            src0 or src1 when others;
end sel_arch;
```
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 and b;
    with tmp select
    y <= '0' when "00",
        '1' when "01",
        '1' when "10",
        '1' when others; -- "11"
end a;
```

Conceptual implementation

- Achieved by a multiplexing circuit
- Abstract (k+1)-to-1 multiplexer
  - sel is with a data type of (k+1) values: c0, c1, c2, ..., ck

```vhdl
-- select_expression is with a data type of 5 values: c0, c1, c2, c3, c4

with select_expression select
  sig <= value_expr_0 when c0,
       value_expr_1 when c1,
       value_expr_n when others;
```

Detailed implementation examples

- 4-to-1 mux
- E.g.,
  ```vhdl
  signal a, b, r: unsigned(7 downto 0);
  signal n: std_logic_vector(1 downto 0);
  ```
  with n select
    r <= a+1 when "11",
    a-b-1 when "10",
    a+b when others;

- ```vhdl
  ```
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 sel=0,
       value_expr_1 when sel=3 or sel=5,
       value_expr_2 when sel=6,
       value_expr_n when others;
```

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;
```

```vhdl
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
      pc_reg;
    ```

  - May "over-specify" for a functional table based circuit.
    - E.g., mux
    ```vhdl
    x <= a when (s="00") else
       b when (s="01") else
       c when (s="10") else
       d;
    ```