Chapter 4
Concurrent Signal Assignment
Statements

Dr. Curt Nelson
Engr433 – Digital Design

Outline

Combinational versus sequential circuits;
Simple signal assignment statement;
Conditional signal assignment statement;
Selected signal assignment statement;
Conditional vs. selected signal assignment.
Combinational vs. Sequential Circuits

- Combinational circuit
  - No internal state;
  - Output is a function of inputs only;
  - No latches/FFs or closed feedback loop.
- Sequential circuit
  - Contains internal state;
  - Output is a function of inputs and internal state.
- Sequential circuit will be discussed in Chapter 5.

Simple Signal Assignment Statement

- Simple signal assignment is a special case of conditional signal assignment.
- Syntax
  
  signal_name <= value_expression;

- Example
  
  y <= a + b + 1 after 10 ns;

- Timing info is ignored in synthesis and δ-delay (tiny delay) is used instead.
Simple Signal Assignment Statement

- Example
  
  ```
  status <= '1';
  even <= (p1 and p2) or (p3 and p4);
  arith_result <= a + b + c - 1;
  ```

- Implementation of example

![Diagram of arithmetic expression]

Signal Assignment Statement With Closed Feedback Loop

- A signal appears in both sides of a concurrent assignment statement.
- Example
  
  ```
  q <= ((not q) and (not en)) or (d and en);
  ```

- Syntactically correct.
- Forms a closed feedback loop.
- Should be avoided.
Conditional Signal Assignment Statement

- 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 ... 
  ```

Example: 4-to-1 Mux

```vhdl
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;
```

<table>
<thead>
<tr>
<th>input</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>
Example: 2-to-2\(^2\) Binary Decoder

```vhdl
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;
```

Example: 4-to-2 Priority Encoder

```vhdl
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;
```
Example: Simple ALU

```vhdl
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
begin
  inc <= std_logic_vector(unsigned(src0)+1);
  sum <= std_logic_vector(unsigned(src0)+unsigned(src1));
  diff <= std_logic_vector(unsigned(src0)-unsigned(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;
  ```

- **Evaluated in descending order.**
- **Implicit priority**
  - Top value expression has a “higher priority”
2-to-1 Multiplexer

- 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;
```

2-to-1 “Abstract” Mux
4-to-1 “Abstract” Mux

```vhdl
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;
```

Implementation Example #1

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

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a=b</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Implementation Example #2

```vhdl
signal a, b, c, x, y, r: std_logic;

r <= a when x = y else b when x > y else c;
```

Implementation Example #2 Schematic
Implementation Example #3

```vhdl
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;
```

Selected Signal Assignment Statement

- **Simplified syntax**
  ```vhdl
  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`
Example: 4-to-1 Mux

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

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>x</td>
</tr>
<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>

Example: 4-to-1 Mux

- Can “11” be used to replace others?

  with s select
  x <= a when "00",
    b when "01",
    c when "10",
    d when "11";
Example: 2-to-\(2^2\) Binary Decoder

\begin{verbatim}
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;
\end{verbatim}

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0001</td>
</tr>
<tr>
<td>01</td>
<td>0010</td>
</tr>
<tr>
<td>10</td>
<td>0100</td>
</tr>
<tr>
<td>11</td>
<td>1000</td>
</tr>
</tbody>
</table>

Example: 4-to-2 Priority Encoder

\begin{verbatim}
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;
end sel_arch;
\end{verbatim}

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>r(3)</td>
<td>r(2)</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>111</td>
<td>101</td>
</tr>
<tr>
<td>1111</td>
<td>1010</td>
</tr>
<tr>
<td>0000</td>
<td>000</td>
</tr>
</tbody>
</table>
Example: 4-to-1 Mux

- Can we use ‘-’?

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

- Note implied priority.

---

Example: 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 "000"|"001"|"010"|"011",
              sum when "100",
              diff when "101",
              src0 and src1 when "110",
              src0 or src1 when others; -- "111"
end sel_arch;
```

---

<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>
Example: 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
  y <= '0' when "00",
      '1' when "01",
      '1' when "10",
      '1' when others; -- "11"
end a;
```

Truth Table - 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
Conceptual Implementation

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

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

Detailed Implementation Example

```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 for Conceptual Implementation](image1)

![Diagram for Detailed Implementation Example](image2)
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_expr_0 else
      value_expr_1 when bool_expr_1 else
      value_expr_2 when bool_expr_2 else
      value_expr_n;
```

```vhdl
sel(2) <= '1' when bool_expr_0 else '0';
sel(1) <= '1' when bool_expr_1 else '0';
sel(0) <= '1' when bool_expr_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 like this

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