ECE 545
Lecture 5

Data Flow Modeling in VHDL
Required reading

• P. Chu, *RTL Hardware Design using VHDL*

*Chapter 4, Concurrent Signal Assignment Statements of VHDL*
Types of VHDL Description

VHDL Descriptions

- dataflow
  - Concurrent statements
- structural
  - Components and interconnects
- behavioral (sequential)
  - Sequential statements
    - Registers
    - State machines
    - Instruction decoders

Subset most suitable for synthesis
Synthesizable VHDL

Dataflow VHDL Description → VHDL code synthesizable

Dataflow VHDL Description × VHDL code synthesizable
Register Transfer Level (RTL) Design Description

Today’s Topic

Combinational Logic

Registers

Combinational Logic
Data-Flow VHDL

Concurrent Statements

• concurrent signal assignment
  \[ \leftarrow \]

• conditional concurrent signal assignment
  \[ \text{when-else} \]

• selected concurrent signal assignment
  \[ \text{with-select-when} \]
Concurrent signal assignment

\[
\text{target\_signal} \leftarrow \text{expression};
\]
Conditional concurrent signal assignment

When - Else

target_signal <= value1 when condition1 else value2 when condition2 else . . . valueN-1 when conditionN-1 else valueN;
Selected concurrent signal assignment

With –Select-When

```
with choice_expression select
    target_signal <= expression1 when choices_1,
                     expression2 when choices_2,
                     ...,
                     expressionN when choices_N;
```
Data-Flow VHDL

Concurrent Statements

• *simple* concurrent signal assignment
  
  \[ \leftrightarrow \]

• *conditional* concurrent signal assignment
  
  \[ \text{when}-\text{else} \]

• *selected* concurrent signal assignment
  
  \[ \text{with}-\text{select}-\text{when} \]
Wires and Buses
Signals

SIGNAL a : STD_LOGIC;

SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);
Merging wires and buses

```vhdl
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);
d <= a & b & c;
```
Splitting buses

SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

a <= d(9 downto 6);
b <= d(5 downto 1);
c <= d(0);
Data-flow VHDL: Example
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY fulladd IS
    PORT ( x : IN STD_LOGIC ;
             y : IN STD_LOGIC ;
             cin : IN STD_LOGIC ;
             s : OUT STD_LOGIC ;
             cout : OUT STD_LOGIC ) ;
END fulladd ;
ARCHITECTURE dataflow OF fulladd IS
BEGIN
    s       <=   x XOR y XOR cin ;
    cout   <=  (x AND y) OR (cin AND x) OR (cin AND y) ;
END dataflow ;
Logic Operators

• Logic operators

\[
\text{and} \quad \text{or} \quad \text{nand} \quad \text{nor} \quad \text{xor} \quad \text{not} \quad \text{xnor}
\]

• Logic operators precedence

Highest

\[
\text{not}
\]

\[
\text{and} \quad \text{or} \quad \text{nand} \quad \text{nor} \quad \text{xor} \quad \text{xnor}
\]

Lowest

only in VHDL-93 or later
No Implied Precedence

Wanted: \( y = ab + cd \)

**Incorrect**

\( y \leq a \text{ and } b \text{ or } c \text{ and } d \);
equivalent to
\( y \leq ((a \text{ and } b) \text{ or } c) \text{ and } d \);
equivalent to
\( y = (ab + c)d \)

**Correct**

\( y \leq (a \text{ and } b) \text{ or } (c \text{ and } d) \);
arith_result <= a + b + c - 1;
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
Data-Flow VHDL

Concurrent Statements

- **simple** concurrent signal assignment $(\leftrightarrow)$
- **conditional** concurrent signal assignment (when-else)
- **selected** concurrent signal assignment (with-select-when)
- **generate scheme for equations** (for-generate)
Conditional concurrent signal assignment

When - Else

```vhdl
target_signal <= value1 when condition1 else value2 when condition2 else . . . valueN-1 when conditionN-1 else valueN;
```
Most often implied structure

**When - Else**

```plaintext
target_signal <= value1 when condition1 else
value2 when condition2 else
...
valueN-1 when conditionN-1 else
valueN;
```
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;

![Diagram](image)
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;
• E.g.,

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

![Diagram](image-url)
E.g.,

```vhls
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;
```

![Diagram of logic circuit](image)
Signed and Unsigned Types

**Behave exactly like**

`STD_LOGIC_VECTOR`

plus, they determine whether a given vector should be treated as a signed or unsigned number.

Require

```use ieee.numeric_std.all;```
Operators

• Relational operators

=    /=    <    <=    >    >=

• Logic and relational operators precedence

Highest

not
=    /=    <    <=    >    >=

and    or    nand    nor    xor    xnor

Lowest
Priority of logic and relational operators

compare a = bc

Incorrect
... when a = b and c else ...
equivalent to
... when (a = b) and c else ...

Correct
... when a = (b and c) else ...
Data-Flow VHDL

**Concurrent Statements**

- **simple** concurrent signal assignment
  
- **conditional** concurrent signal assignment
  
- **selected** concurrent signal assignment
  
- **generate scheme for equations**
Selected concurrent signal assignment

**With –Select-When**

```vhdl
with choice_expression select
    target_signal <= expression1 when choices_1,
                     expression2 when choices_2,
                     . . .
                     expressionN when choices_N;
```
Most Often Implied Structure

\textbf{With –Select-When}

\begin{verbatim}
with choice_expression select
    target_signal <= expression1 when choices_1,
                 expression2 when choices_2,
                 \ldots
                 expressionN when choices_N;
\end{verbatim}
Allowed formats of $choices_k$

WHEN value

WHEN value_1 | value_2 | ... | value N

WHEN OTHERS
Allowed formats of \textit{choice}_k - example

\begin{verbatim}
WITH sel SELECT
    y <= a WHEN "000",
    c WHEN "001" | "111",
    d WHEN OTHERS;
\end{verbatim}
Syntax

• 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
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",
          d when others;
end sel_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>
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
    x <= "0001" when "00",
        "0010" when "01",
        "0100" when "10",
        "1000" 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>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., simple ALU

```vhdl
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
        result <= inc when "000" | "001" | "010" | "011",
                  sum when "100",
                  diff when "101",
                  src0 and src1 when "110",
                  src0 or src1 when others;
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></td>
<td></td>
</tr>
<tr>
<td>000</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>
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
    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
- select_expression 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;
```
• 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 circuit with inputs a, b, and s, and output r. The diagram includes adders, subtractors, and logic gates to compute the output based on the inputs and the select signal s.](image-url)
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;

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

\[
\begin{align*}
\text{sig} & \leftarrow \text{value_expr}_0 \quad \text{when} \quad \text{bool_exp}_0 \quad \text{else} \\
& \hspace{1cm} \text{value_expr}_1 \quad \text{when} \quad \text{bool_exp}_1 \quad \text{else} \\
& \hspace{1cm} \text{value_expr}_2 \quad \text{when} \quad \text{bool_exp}_2 \quad \text{else} \\
& \hspace{1cm} \text{value_expr}_n;
\end{align*}
\]

\[
\begin{align*}
\text{sel}(2) & \leftarrow '1' \quad \text{when} \quad \text{bool_exp}_0 \quad \text{else} \quad '0'; \\
\text{sel}(1) & \leftarrow '1' \quad \text{when} \quad \text{bool_exp}_1 \quad \text{else} \quad '0'; \\
\text{sel}(0) & \leftarrow '1' \quad \text{when} \quad \text{bool_exp}_2 \quad \text{else} \quad '0'; \\
\text{with} \quad \text{sel} \quad \text{select} \\
& \hspace{1cm} \text{sig} \leftarrow \text{value_expr}_0 \quad \text{when} \quad "100" | "101" | "110" | "111", \\
& \hspace{1cm} \text{value_expr}_1 \quad \text{when} \quad "010" | "011", \\
& \hspace{1cm} \text{value_expr}_2 \quad \text{when} \quad "001", \\
& \hspace{1cm} \text{value_expr}_n \quad \text{when} \quad \text{others};
\end{align*}
\]
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.,

```verilog
  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

\[
x \leftarrow a \text{ when } (s="00") \text{ else b when } (s="01") \text{ else c when } (s="10") \text{ else d;}
\]

\[
x \leftarrow c \text{ when } (s="10") \text{ else a when } (s="00") \text{ else b when } (s="01") \text{ else d;}
\]

\[
x \leftarrow c \text{ when } (s="10") \text{ else b when } (s="01") \text{ else a when } (s="00") \text{ else d;}
\]