ECE 545
Lecture 7
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:

- Testbenches
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

• **simple** concurrent signal assignment ($\leftarrow$)

• **conditional** concurrent signal assignment (when-else)

• **selected** concurrent signal assignment (with-select-when)
Simple concurrent signal assignment

\[
<=
\]

target_signal <= 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

```vhdl
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
  \((\leftarrow)\)

• **conditional** concurrent signal assignment
  \((\text{when-else})\)

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

SIGNAL a : STD_LOGIC;

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

\[
d = a \parallel b \parallel c
\]

```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 <=
```
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;
```

\[ d = a \| b \| c \]
Splitting 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);

a <=
b <=
c <=
```

\[
\begin{align*}
a &= d \\
b &= d \\
c &= d
\end{align*}
\]
Splitting buses

\[ a = d_{9..6} \]
\[ b = d_{5..1} \]
\[ c = d_0 \]

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);

\begin{align*}
  a & \leftarrow d(9 \text{ downto } 6); \\
  b & \leftarrow d(5 \text{ downto } 1); \\
  c & \leftarrow d(0);
\end{align*}
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 ;
Data-flow VHDL: Example (2)

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 ;

 equivalent to

ARCHITECTURE dataflow OF fulladd IS
BEGIN
    cout <= (x AND y) OR (cin AND x) OR (cin AND y) ;
    s <= x XOR y XOR cin ;
END dataflow ;
Logic Operators

- Logic operators
  
  and  or  nand  nor  xor  not  xnor

- Logic operators precedence
  
  Highest
  
  not
  
  and  or  nand  nor  xor  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)
Conditional concurrent signal assignment

When - Else

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;
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 of a comparator circuit with inputs a, b, c, x, y, and output r.]
\[
\text{signal } a, b, r: \text{ unsigned}(7 \text{ downto 0}); \\
\text{signal } x, y: \text{ unsigned}(3 \text{ downto 0}); \\
\]

\[
\begin{align*}
 r &\leq a+b \text{ when } x+y>1 \text{ else } \\
 &a-b-1 \text{ when } x>y \text{ and } y!=0 \text{ else } \\
 &a+1;
\end{align*}
\]
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
  
  \[
  = \quad /= \quad < \quad <= \quad > \quad >=
  \]

- Logic and relational operators precedence

Highest

\[
\text{not} \\
= \quad /= \quad < \quad <= \quad > \quad >=
\]

and \quad or \quad nand \quad nor \quad xor \quad xnor

Lowest
Priority of logic and relational operators

\[ \text{compare} \quad a = bc \]

**Incorrect**

... when \( a = b \text{ and } c \) else ... 
equivalent to

... when \((a = b) \text{ and } c\) else ...

**Correct**

... when \( a = (b \text{ and } c) \) else ...
Data-Flow VHDL

Concurrent Statements

• **simple** concurrent signal assignment
  \(\leftarrow\)

• **conditional** concurrent signal assignment
  (when-else)

• **selected** concurrent signal assignment
  (with-select-when)
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;
```
Most Often Implied Structure

*With –Select-When*

```plaintext
with choice_expression select
  target_signal <= expression1 when choices_1,
                  expression2 when choices_2,
                  ...  
                  expressionN when choices_N;
```

Diagram:
- `choice_expression` → `choices_1` → `expression1`
- `choice_expression` → `choices_2` → `expression2`
- `choice_expression` → `choices_N` → `expressionN`
- `choices_1`, `choices_2`, `choices_N` are connected to `target_signal`
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
  begin
    x <= a when "00",
         b when "01",
         c when "10",
         d when others;
  end sel_arch;
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>
Can “11” be used to replace others?

```plaintext
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
  begin
    result <= inc when "000" | "001" | "010" | "011",
              sum when "100",
              diff when "101",
              src0 and src1 when "110",
              src0 or src1 when others;
  end;

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 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>
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;
```

```
<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:
    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](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;
```

```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

\[
\text{sig <= value_expr_0 when bool_exp_0 else value_expr_1 when bool_exp_1 else value_expr_2 when bool_exp_2 else value_expr_n;}
\]

\[
\text{sel(2) <= '1' when bool_exp_0 else '0';}
\text{sel(1) <= '1' when bool_exp_1 else '0';}
\text{sel(0) <= '1' when bool_exp_2 else '0';}
\]

\[
\text{with sel select}
\text{sig <= value_expr_0 when "100" | "101" | "110" | "111",}
\text{value_expr_1 when "010" | "011",}
\text{value_expr_2 when "001",}
\text{value_expr_n when others;}
\]
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.,

```
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

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

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

\[
\begin{align*}
x & \leftarrow c \ \text{when} \ (s="10") \ \text{else} \\
b & \ \text{when} \ (s="01") \ \text{else} \\
a & \ \text{when} \ (s="00") \ \text{else} \\
d & ; \\
\end{align*}
\]
"when-else" should be used when:

1) there is only one condition (and thus, only one else), as in the 2-to-1 MUX

2) conditions are independent of each other (e.g., they test values of different signals)

3) conditions reflect priority (as in priority encoder); one with the highest priority need to be tested first.
"with-select-when" should be used when there is
1) more than one condition
2) conditions are closely related to each other
   (e.g., represent different ranges of values of the
   same signal)
3) all conditions have the same priority (as in the
   4-to-1 MUX).