ECE 545 Lecture 5
Dataflow Modeling in VHDL

Required reading
- P. Chu, RTL Hardware Design using VHDL
  Chapter 4, Concurrent Signal Assignment Statements of VHDL
  Chapter 6.3, Realization of VHDL Data Types

Types of VHDL Description

- VHDL Descriptions
  - dataflow
  - structural
  - behavioral (sequential)

Components and interconnects
- Concurrent statements
- Sequential statements
- Testbenches
- 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
- **simple** concurrent signal assignment (→)
- **conditional** concurrent signal assignment (when-else)
- **selected** concurrent signal assignment (with-select-when)
Simple concurrent signal assignment

```vhdl
<=
  target_signal <= expression;
```

Conditional concurrent signal assignment

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

Selected concurrent signal assignment

```vhdl
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
  (<=)
- conditional concurrent signal assignment
  (when-else)
- selected concurrent signal assignment
  (with-select-when)

Signals

```vhdl
SIGNAL a : STD_LOGIC;
SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);
```

Wires and Buses
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;

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;

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;

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 ;

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

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

• Logic operators precedence
  - in VHDL-93 and later
  - Highest
  - Lowest

No Implied Precedence

Wanted: \( y = ab + cd \)

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

Correct
\( y <= (a \text{ and } b) \text{ or } (c \text{ and } d) ; \)

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 \( (\Rightarrow) \)
• conditional concurrent signal assignment \( (\text{when-else}) \)
• selected concurrent signal assignment \( (\text{with-select-when}) \)
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**

```
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;

```
• E.g.,

```vhdl
signal a, b, r: unsigned(3 downto 0);
signal x, y: unsigned(3 downto 0);
... r <= a+b when x>y else a+1 when x>y and y=0 else a+1;
```

### 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
  ```

### Data-Flow VHDL

**Concurrent Statements**

- **simple concurrent signal assignment**  
  (<=)

- **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**

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

Allowed formats of *choices_k*

- WHEN value
  - WHEN value_1 | value_2 | ... | value N
  - WHEN OTHERS

Allowed formats of *choice_k - example*

```vhdl
WITH sel SELECT
    y <= a WHEN "000",
        c WHEN "001" | "111",
        d WHEN OTHERS;
```

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;
```

```plaintext
<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?

```
with s select
  x <= a when "00",
       b when "01",
       c when "10",
       d when "11";
```

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

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

E.g., simple ALU

```
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));
  result <= inc when "0000" or "0011" or "0100",
            sum when "100",
            diff when "011",
            src0 or src1 when "110",
            src0 or src1 when others;
end sel_arch;
```

E.g., Truth table

```
library ieee;
use ieee.std_logic_1164.all;
entity truth_table is
  port(
    a,b: in std_logic;
    y: out std_logic);
end entity truth_table;
architecture a of truth_table is
begin
  tmp <= a & b;
  with tmp select
    y <= '0' when '00',
         '1' when '01',
         '1' when '10',
         '1' when others; -- '1'
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
with select_expression select
  sig <= value_expr_0 when c0,
         value_expr_1 when c1,
         value_expr_n when others;
```
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 or (sel=c3) or (sel=c5) else
         value_expr_2 when sel=c2, or (sel=c6) else
         value_expr_n when others;
```

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. e.g.,

```plaintext
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`

```plaintext
x <= a when (a="00") else
  b when (a="01") else
  c when (a="10") else
da;
```

when-else vs. with-select-when (1)

"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.

when-else vs. with-select-when (2)

"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).

Use of ‘-’

• In conventional logic design
  – ‘-’ as input value: shorthand to make table compact
  – E.g.,

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>req</td>
<td>code</td>
</tr>
<tr>
<td>1 0 0 10</td>
<td></td>
</tr>
<tr>
<td>1 0 1 10</td>
<td></td>
</tr>
<tr>
<td>1 1 0 10</td>
<td></td>
</tr>
<tr>
<td>1 1 1 10</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>req</td>
<td>code</td>
</tr>
<tr>
<td>1 - 10</td>
<td></td>
</tr>
<tr>
<td>0 1 01</td>
<td></td>
</tr>
<tr>
<td>0 0 00</td>
<td></td>
</tr>
</tbody>
</table>

– ‘-’ as output value: help simplification
– E.g.,
  – ‘-’ assigned to `t: a + b`
  – ‘-’ assigned to `0: a'b + ab'`
Use ‘-‘ in VHDL

- As input value (against our intuition):
  - Wrong:
    \[
    y \leftarrow \text{"10" when } \text{req}=	ext{"1-" else } \text{"01" when } \text{req}=	ext{"01-" else } \text{"00" when } \text{req}=	ext{"001" else } \text{"00"};
    \]

- Fix #1:
  \[
  y \leftarrow \text{"10" when } \text{req(2)}=	ext{"1" else } \text{"01" when } \text{req(2 downto 1)}=	ext{"01" else } \text{"00" when } \text{req(2 downto 0)}=	ext{"001" else } \text{"00"};
  \]

- Fix #2:
  \[
  \text{use ieee.numeric_std.all;}
  \]
  \[
  y \leftarrow \text{"10" when } \text{std_match(req,\text{"1-"}) else } \text{"01" when } \text{std_match(req,\text{"01-"}) else } \text{"00" when } \text{std_match(req,\text{"001"}) else } \text{"00"};
  \]

Wrong:

\[
\text{with req select}
\]
\[
y \leftarrow \text{"10" when } \text{"1-"}, \text{"01" when } \text{"01-"}, \text{"00" when } \text{"001"}, \text{"00" when } \text{others};
\]

Fix:

\[
\text{with req select}
\]
\[
y \leftarrow \text{"10" when } \text{"100"|"101"|"110"|"111"}, \text{"00" when } \text{"010"|"011"}, \text{"00" when } \text{others};
\]

‘-‘ as an output value in VHDL

- May work with some software

\[
\text{sel } \leftarrow a \& b;
\]
\[
\text{with sel select}
\]
\[
y \leftarrow \text{"0" when } \text{"00"}, \text{"1" when } \text{"01"}, \text{"1" when } \text{"10"}, \text{"-" when } \text{others};
\]

Use and synthesis of ‘Z’

- Tri-state buffer:
  - Output with “high-impedance”
  - Not a value in Boolean algebra
  - Need special output circuitry (tri-state buffer)

\[
\text{OE}
\]
\[
\text{A}_{\text{IN}} \rightarrow y
\]
\[
\begin{array}{c c}
\text{OE} & y \\
0 & Z \\
1 & \text{A}_{\text{IN}}
\end{array}
\]

- Major application:
  - Bi-directional I/O pins
  - Tri-state bus

- VHDL description:
  \[
y \leftarrow \text{\text{'Z' when oe='1' else a}_{\text{IN}};}
\]

- ‘Z’ cannot be used as input or manipulated
  \[
f \leftarrow \text{\text{'Z' \text{and a;}}}
\]

\[
y \leftarrow \text{\text{data}_{\text{A}} \text{when in}_{\text{BUS}}=\text{'Z' else}}
\]
\[
\text{data}_{\text{B}};
\]
• Separate tri-state buffer from regular code:
  – Less clear:
    ```
    with sel select
    y <= 'Z' when "00",
    '1' when "01"|"11",
    '0' when others;
    ```
  – Better:
    ```
    with sel select
    tmp <= '1' when "01"|"11",
    '0' when others;
    y <= 'Z' when sel="00" else tmp;
    ```

Bi-directional i/o pins

```
entity bi_deno is
  port(bi: inout std_logic;
  ...
begin
  sig_out <= output_expression;
  ... <= expression_with_sig_in;
  ... bi <= sig_out when dir='1' else 'Z';
  sig_in <= bi;
  ...
```
• Problem with tri-state bus
  – Difficult to optimize, verify and test
  – Somewhat difficult to design: "parking", "fighting"

• Alternative to tri-state bus: mux

```vhdl
with src_select select
  data_bus <= i0 when "00",
             i1 when "01",
             i2 when "10",
             i3 when others; -- "11"
```