ECE 545 Lecture 5

Data Flow Modeling of Combinational Logic

Required reading

- P. Chu, RTL Hardware Design using VHDL

  Chapter 4, Concurrent Signal Assignment
  Statements of VHDL

VHDL Design Styles

- dataflow
- structural
- behavioral (sequential)

Concurrent statements
Components and interconnects
Sequential statements
- Registers
- State machines
- Instruction decoders

Subset most suitable for synthesis

Synthesizable VHDL

Dataflow VHDL Design Style ➔ VHDL code synthesizable

Dataflow VHDL Design Style ❌ 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 (when-else)
- selected concurrent signal assignment (with-select-when)
- generate scheme for equations (for-generate)

Data-flow VHDL: Example

\begin{verbatim}
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 ;
\end{verbatim}

Logic Operators

- Logic operators
  - and, or, nand, nor, xor, not, xnor

- Logic operators precedence
  - Highest: not
  - Lowest: and, or, nand, nor, xor, xnor
  - 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
\[
y = (ab + cd);
\]

**Correct**
\[
y \leq (a \text{ and } b) \text{ or } (c \text{ and } d);
\]

E.g.,
\[
\begin{align*}
\text{status} & \leq '1'; \\
\text{even} & \leq (p1 \text{ and } p2) \text{ or } (p3 \text{ and } p4); \\
\text{arith\_out} & \leq a + b + c - 1;
\end{align*}
\]

Implementation of last statement

Data-flow VHDL

**Major instructions**

Concurrent statements
- concurrent signal assignment \((\leftarrow)\)
- conditional concurrent signal assignment
  \((\text{when-else})\)
- selected concurrent signal assignment
  \((\text{with-select-when})\)
- generate scheme for equations
  \((\text{for-generate})\)

**Conditional concurrent signal assignment**

*When - Else*

*When - Else*

**Most often implied structure**

*When - Else*

\[
\begin{align*}
\text{target\_signal} & \leftarrow \text{value}1 \quad \text{when} \quad \text{condition}1 \quad \text{else} \\
& \quad \text{value}2 \quad \text{when} \quad \text{condition}2 \quad \text{else} \\
& \quad \ldots \\
& \quad \text{value}N-1 \quad \text{when} \quad \text{condition}N-1 \quad \text{else} \\
& \quad \text{value}N;
\end{align*}
\]

When - Else

\[
\begin{align*}
\text{target\_signal} & \leftarrow \text{value}1 \quad \text{when} \quad \text{condition}1 \quad \text{else} \\
& \quad \text{value}2 \quad \text{when} \quad \text{condition}2 \quad \text{else} \\
& \quad \ldots \\
& \quad \text{value}N-1 \quad \text{when} \quad \text{condition}N-1 \quad \text{else} \\
& \quad \text{value}N;
\end{align*}
\]

**Most often implied structure**

*When - Else*

\[
\begin{align*}
\text{target\_signal} & \leftarrow \text{value}1 \quad \text{when} \quad \text{condition}1 \quad \text{else} \\
& \quad \text{value}2 \quad \text{when} \quad \text{condition}2 \quad \text{else} \\
& \quad \ldots \\
& \quad \text{value}N-1 \quad \text{when} \quad \text{condition}N-1 \quad \text{else} \\
& \quad \text{value}N;
\end{align*}
\]
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;
```

- E.g.,
  ```
  signal a, b, y: std_logic;
  y <= '0' when a = b else '1';
  ```

- E.g.,
  ```
  signal a, b, c, x, y, r: std_logic;
  r <= a when x = y else b when z = y else c;
  ```
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

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 …

VHDL operators

Operator Class | Operator
---|---

Highest precedence

Miscellaneous: **, ABS, NOT

Multiplying: *, /, MOD, REM

Sign: +, -

Adding: +, -

Shifting: SLL, SRL, SRA

Relational: =, <, <=, >, >=

Lowest precedence

Logical: AND, OR, NAND, NOR, XOR, XNOR
Data-flow VHDL

Major instructions

Concurrent statements

• concurrent signal assignment  (=)
• conditional concurrent signal assignment
  (when-else)
• selected concurrent signal assignment
  (with-select-when)
• generate scheme for equations
  (for-generate)

Selected concurrent signal assignment

With –Select-When

\[
\text{with choice_expression select}
\]
\[
\text{target_signal <= expression1 when choices_1,}
\]
\[
\text{expression2 when choices_2,}
\]
\[
\vdots
\]
\[
\text{expressionN when choices_N;}
\]

Most Often Implied Structure

With –Select-When

\[
\text{with choice_expression select}
\]
\[
\text{target_signal <= expression1 when choices_1,}
\]
\[
\text{expression2 when choices_2,}
\]
\[
\vdots
\]
\[
\text{expressionN when choices_N;}
\]

Allowed formats of choices_k

WHEN value

WHEN value_1 | value_2 | ... | value N

WHEN OTHERS

Allowed formats of choice_k - example

WITH sel SELECT
\[
y <= a \text{ WHEN } "000",
\]
\[
c \text{ WHEN } "001" | "111",
\]
\[
d \text{ WHEN OTHERS;}
\]

Syntax

• Simplified syntax:

\[
\text{with select_expression select}
\]
\[
\text{signal_name <=}
\]
\[
\text{value_expr_1 when choice_1,}
\]
\[
\text{value_expr_2 when choice_2,}
\]
\[
\vdots
\]
\[
\text{value_expr_n when choice_n;}
\]
• 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 a 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 a 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>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>

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

```vhdl
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;
  active <= r(3) or r(2) or r(1) or r(0);
end sel_arch;
```

<table>
<thead>
<tr>
<th>r code</th>
<th>active</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-</td>
<td>1</td>
</tr>
<tr>
<td>01-</td>
<td>1</td>
</tr>
<tr>
<td>001-</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
</tr>
</tbody>
</table>

• Can we use '-'?

```vhdl
with a select
x <= "11" when "1- -",
    "10" when "01- -",
    "01" when "001- -",
    "00" when others;
```
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));
  with ctrl select
    result <= inc when "000", new when "001", "101", "110", "111";
  end select
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 truth_table;

architecture a of truth_table is
begin
  y <= '0' when '00', '1' when '01', '1' when '10', '1' when others; -- '1'
end a;

E.g., Truth table

architecture a of truth_table is
begin
  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;

• E.g.,

signal a, r: unsigned(7 downto 0);
signal s: std_logic_vector(1 downto 0);
with a select
  r <= a+1 when '11',
       a-1 when '10',
       a+1 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 sel<1><0,
           value_expr_2 when c2<4>,
           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;
```

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
    ...
```

- May “over-specify” for a functional table based circuit.
  - E.g., mux

```vhdl
x <= a when (a='00') else
    b when (a='01') else
    c when (a='10') else
    d;
```

```vhdl
x <= c when (a='10') else
    a when (a='00') else
    b when (a='01') else
    d;
```

```vhdl
x <= c when (a='10') else
    b when (a='01') else
    a when (a='00') else
    d;
```
MLU Example

MLU: Entity Declaration

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mlu IS
  PORT(
    NEG_A : IN STD_LOGIC;
    NEG_B : IN STD_LOGIC;
    NEG_Y : IN STD_LOGIC;
    A :           IN STD_LOGIC;
    B :           IN STD_LOGIC;
    L1 :         IN STD_LOGIC;
    L0 :         IN STD_LOGIC;
    Y :          OUT STD_LOGIC
  );
END mlu;

MLU: Architecture Declarative Section

ARCHITECTURE mlu_dataflow OF mlu IS
  SIGNAL  A1 :  STD_LOGIC;
  SIGNAL  B1 :  STD_LOGIC;
  SIGNAL  Y1 : STD_LOGIC;
  SIGNAL  MUX_0 : STD_LOGIC;
  SIGNAL  MUX_1 : STD_LOGIC;
  SIGNAL  MUX_2 : STD_LOGIC;
  SIGNAL  MUX_3 : STD_LOGIC;
  SIGNAL  L : STD_LOGIC_VECTOR(1 DOWNTO 0);
  BEGIN
    A1<= NOT A  WHEN (NEG_A='1') ELSE
      A;
    B1<= NOT B  WHEN (NEG_B='1') ELSE
      B;
    Y <= NOT Y1 WHEN (NEG_Y='1') ELSE
      Y1;
    MUX_0 <= A1 AND B1;
    MUX_1 <= A1 OR B1;
    MUX_2 <= A1 XOR B1;
    MUX_3 <= A1 XNOR B1;
    L <= L1 & L0;
    with (L) select
      Y1 <= MUX_0  WHEN "00",
           MUX_1  WHEN "01",
           MUX_2  WHEN "10",
           MUX_3  WHEN OTHERS;
END mlu_dataflow;

MLU - Architecture Body

BEGIN
  A1<= NOT A  WHEN (NEG_A='1') ELSE
    A;
  B1<= NOT B  WHEN (NEG_B='1') ELSE
    B;
  Y <= NOT Y1 WHEN (NEG_Y='1') ELSE
    Y1;
  MUX_0 <= A1 AND B1;
  MUX_1 <= A1 OR B1;
  MUX_2 <= A1 XOR B1;
  MUX_3 <= A1 XNOR B1;
  L <= L1 & L0;
  with (L) select
    Y1 <= MUX_0  WHEN "00",
         MUX_1  WHEN "01",
         MUX_2  WHEN "10",
         MUX_3  WHEN OTHERS;
END mlu_dataflow;

Modeling Common Combinational Logic Components Using Dataflow VHDL
Wires and Buses

Merging wires and buses

Splitting buses

Fixed Shifters & Rotators

Signals

SIGNAL a : STD_LOGIC;

SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 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);
d <= a & b & c;

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);

Fixed Shift in VHDL

SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL AshiftR: STD_LOGIC_VECTOR(3 DOWNTO 0);

A(3) A(2) A(1) A(0)

A>>1

A(3) A(2) A(1) A(0)

AshiftR <=
Fixed Rotation in VHDL

SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL ArotL : STD_LOGIC_VECTOR(3 DOWNTO 0);

A<<<1

Buffers

(a) A tri-state buffer

(b) Equivalent circuit

(c) Truth table

Tri-state Buffer – example (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY tri_state IS
PORT ( ena: IN STD_LOGIC;
input:  IN STD_LOGIC;
output: OUT STD_LOGIC
);
END tri_state;

ARCHITECTURE dataflow OF tri_state IS
BEGIN
output <= input WHEN (ena = '1') ELSE 'Z';
END dataflow;

Tri-state Buffer – example (2)

Four types of Tri-state Buffers

(a) Tri-state Buffer

(b) Equivalent circuit

(c) Truth table

(d) Truth table
Multiplexers

2-to-1 Multiplexer

![Graphical symbol](image)

(a) Graphical symbol

<table>
<thead>
<tr>
<th>s</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) Truth table

VHDL code for a 2-to-1 Multiplexer

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux2to1 IS
    PORT ( w0, w1, s : IN  STD_LOGIC ;
            f : OUT  STD_LOGIC );
END mux2to1;

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
    f <= w0 WHEN s = '0' ELSE w1 ;
END dataflow;
```

VHDL code for a cascade of two multiplexers

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux_cascade IS
    PORT ( w1, w2, w3: IN  STD_LOGIC ;
            s1, s2          : IN  STD_LOGIC ;
            f : OUT  STD_LOGIC );
END mux_cascade;

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
    f <= w1 WHEN s1 = '1' ELSE w2  WHEN s2 = '1' ELSE w3 ;
END dataflow;
```

Cascade of two multiplexers

![Diagram](image)

4-to-1 Multiplexer

(a) Graphic symbol

<table>
<thead>
<tr>
<th>s1</th>
<th>s0</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>w0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>w1</td>
</tr>
</tbody>
</table>

(b) Truth table
VHDL code for a 4-to-1 Multiplexer

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux4to1 IS
PORT ( w0, w1, w2, w3 : IN STD_LOGIC;
       s : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       f : OUT STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE dataflow OF mux4to1 IS
BEGIN
WITH s SELECT
f <= w0 WHEN "00",
    w1 WHEN "01",
    w2 WHEN "10",
    w3 WHEN OTHERS;
END dataflow ;

Decoders

2-to-4 Decoder

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec2to4 IS
PORT ( w : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       En : IN STD_LOGIC;
       y : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END dec2to4 ;

ARCHITECTURE dataflow OF dec2to4 IS
SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
BEGIN
Enw <= En & w ;
WITH Enw SELECT
y <= "0001" WHEN "100",
   "0010" WHEN "101",
   "0100" WHEN "110",
   "1000" WHEN "111",
   "0000" WHEN OTHERS ;
END dataflow ;

VHDL code for a 2-to-4 Decoder

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec2to4 IS
PORT ( w : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       En : IN STD_LOGIC;
       y : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END dec2to4 ;

ARCHITECTURE dataflow OF dec2to4 IS
BEGIN
Enw <= En & w ;
WITH Enw SELECT
y <= "0001" WHEN "100",
   "0010" WHEN "101",
   "0100" WHEN "110",
   "1000" WHEN "111",
   "0000" WHEN OTHERS ;
END dataflow ;

Encoders

Priority Encoder

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY priority IS
PORT ( w0, w1, w2, w3 : IN STD_LOGIC;
       y1, y0, E : OUT STD_LOGIC ) ;
END priority ;

ARCHITECTURE dataflow OF priority IS
BEGIN
y1 <= w0 & w1 & w2 & w3 ;
END dataflow ;
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY priority IS
  PORT ( w : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         y : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
         z : OUT STD_LOGIC );
END priority ;

ARCHITECTURE dataflow OF priority IS
BEGIN
  y <= "11" WHEN w(3) = '1' ELSE
       "10" WHEN w(2) = '1' ELSE
       "01" WHEN w(1) = '1' ELSE
       "00" ;
  z <= '0' WHEN w = "0000" ELSE '1';
END dataflow ;