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 VHDL Design Style

Synthesizable VHDL

Register Transfer Level (RTL) Design Description

Today's Topic
Data-Flow VHDL

**Concurrent Statements**
- concurrent signal assignment (≡)
- conditional concurrent signal assignment (when-else)
- selected concurrent signal assignment (with-select-when)
- generate scheme for equations (for-generate)

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:
  - and
  - or
  - nand
  - nor
  - xor
  - not
  - xnor

- Logic operators precedence:
  - Highest: or, nand
  - Lowest: and

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

E.g.,
\[
\text{status} <= '1';
\]
\[
even <= (p1 \text{ and } p2) \text{ or } (p3 \text{ and } p4);
\]
\[
\text{arith}_{-} \text{out} <= a + b + c - 1;
\]

Implementation of last statement

Signal assignment statement with a closed feedback loop

- A signal appears in both sides of a concurrent assignment statement
- E.g.,
\[
q <= ((\text{not } q) \text{ and } (\text{not } \text{en})) \text{ or } (d \text{ and } \text{en});
\]
- Syntactically correct
- Form a closed feedback loop
- Should be avoided

Conditional concurrent signal assignment

When - Else

\[
\begin{align*}
\text{Target signal} & \leftarrow \text{value1 when condition1 else } \\
& \quad \text{value2 when condition2 else } \\
& \quad \vdots \\
& \quad \text{valueN-1 when conditionN-1 else } \\
& \quad \text{valueN}
\end{align*}
\]

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 \( (\text{for-generate}) \)

Most often implied structure

When - Else
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 x>y else
    c;
```

```
<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
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**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>not</code></td>
<td>Highest</td>
</tr>
<tr>
<td><code>and</code></td>
<td></td>
</tr>
<tr>
<td><code>or</code></td>
<td></td>
</tr>
<tr>
<td><code>nand</code></td>
<td></td>
</tr>
<tr>
<td><code>nor</code></td>
<td></td>
</tr>
<tr>
<td><code>xor</code></td>
<td></td>
</tr>
<tr>
<td><code>xnor</code></td>
<td></td>
</tr>
</tbody>
</table>

  - Not, `and`, `or`, `nand`, `nor`, `xor` have the highest precedence.

  - `not` is evaluated before `and`, `or`, `nand`, `nor`, `xor`.

  - `and` is evaluated before `or`.

Priority of logic and relational operators

- **Incorrect**

  - `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

<table>
<thead>
<tr>
<th>Operator Class</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Highest precedence</strong></td>
<td><code>**</code>, <code>ABS</code>, <code>NOT</code></td>
</tr>
<tr>
<td>Multiplying</td>
<td><code>/</code>, <code>MOD</code>, <code>REM</code></td>
</tr>
<tr>
<td>Sign</td>
<td><code>+</code>, <code>-</code></td>
</tr>
<tr>
<td>Adding</td>
<td><code>+</code>, <code>-</code></td>
</tr>
<tr>
<td>Shift</td>
<td><code>SLL</code>, <code>SRL</code>, <code>SRA</code>, <code>ROL</code>, <code>ROR</code></td>
</tr>
<tr>
<td>Relational</td>
<td><code>=</code>, <code>&lt;=</code>, <code>&gt;=</code>, <code>&gt;</code>, <code>&gt;=</code></td>
</tr>
<tr>
<td><strong>Lowest precedence</strong></td>
<td><code>AND</code>, <code>OR</code>, <code>XOR</code>, <code>AND</code>, <code>XOR</code></td>
</tr>
</tbody>
</table>
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

```
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

```
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

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

Syntax

• Simplified syntax:
  ```
  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 a select
  x <= a when "00",
  b when "01",
  c when "10",
  d when others;
end sel_arch;
```

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;
```

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 "0000"|"0001"|"0010"|"0011",
  '01' when "0100"|"0101"|"0110"|"0111",
  '00' when others;
  active <= r(3) or r(2) or r(1) or r(0);
end sel_arch;
```

• Can we use '-'?

```vhdl
with r select
  x <= "11" when "1--",
  "10" when "01--",
  "01" when "001-",
  "00" when others;
```
### 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", "100", "101", "110", "111";
  end sel_arch;
```

### 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
begin
  y <= a and b;
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`

### E.g.,

```vhdl
signal a, b, r : std_logic_vector(1 downto 0);

with a select
  r <= a+1 when '11',
       a-b-1 when '10',
       a+b when others;
```

### 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
begin
  y <= a and b;
end a;
```
3. Conditional vs. selected signal assignment

- Conversion between conditional vs. selected signal assignment
- Comparison

From selected assignment to conditional assignment

```hls
with sel select
    sig <= value_expr_0 when c0,
            value_expr_1 when c1 or c2 or c3,
            value_expr_2 when c2 or c4,
            value_expr_n when others;
```

From conditional assignment to selected assignment

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

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

    ```hls
    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;

MLU Block Diagram

Modeling Common Combinational Logic Components Using Dataflow VHDL
Wires and Buses

Merging wires and 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);
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);

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
A>>1 0 A(3) A(2) A(1) AshiftR
AshiftR <=
Fixed Rotation in VHDL

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

A<<<1

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

ArotL <=

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

Buffers

(a) A tri-state buffer

(b) Equivalent circuit

(c) Truth table

Four types of Tri-state Buffers

(a) x f e

0 0 Z
0 1 Z
1 0 0
1 1 1

(b) x f = 0
e = 1

(c) Four types of Tri-state Buffers

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)
Multiplexers

2-to-1 Multiplexer

VHDL code for a 2-to-1 Multiplexer

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;

Cascade of two multiplexers

VHDL code for a cascade of two multiplexers

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;

4-to-1 Multiplexer

(a) Graphic symbol
(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
BEGIN
  SIGNAL Enw : std_logic_vector(2 DOWNTO 0);
  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

Encoders

Priority Encoder

LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

ARCHITECTURE dataflow OF priority IS
BEGIN
  SIGNAL Enw : std_logic_vector(2 DOWNTO 0);
  Enw <= En & w;
  WITH Enw SELECT
    y1 <= "0001" WHEN "100",
          "0010" WHEN "101",
          "0100" WHEN "110",
          "0000" WHEN OTHERS;
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 ;