ECE 545
Lecture 8
Data Flow Description of Combinational-Circuit Building Blocks
Required reading

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

*Chapter 7, Combinational Circuit Design: Practice*
Fixed Shifters & Rotators
Fixed **Logical** Shift Right in VHDL

```vhdl
SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C : STD_LOGIC_VECTOR(3 DOWNTO 0);
```

![Diagram of Fixed Logical Shift Right in VHDL]
**Fixed Logical Shift Right in VHDL**

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

C = '0' & A(3 downto 1);
Fixed Arithmetic Shift Right in VHDL

SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C:  STD_LOGIC_VECTOR(3 DOWNTO 0);
Fixed Arithmetic Shift Right in VHDL

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

\[
C = A(3) \& A(3 \text{ downto } 1);
\]
Fixed Logical Shift Left in VHDL

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

A
<<1
C

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

A(2) A(1) A(0) ‘0’

A
C
Fixed Logical Shift Left in VHDL

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

\[ A = A(2 \text{ downto } 0) \& '0'; \]
Fixed Rotation Left in VHDL

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

\[ A \ll 1 \]

\[ A(3) \quad A(2) \quad A(1) \quad A(0) \]

\[ A(2) \quad A(1) \quad A(0) \quad A(3) \]
Fixed Rotation Left in VHDL

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

C = A(2 downto 0) & A(3);
Variable Rotators
8-bit Variable Rotator Left

To be covered during one of the future classes
Multiplexers
2-to-1 Multiplexer

(a) Graphical symbol

(b) Truth table

VHDL:
2-to-1 Multiplexer

(a) Graphical symbol

(b) Truth table

**VHDL:**

\[ f \leftarrow w0 \text{ WHEN } s = '0' \text{ ELSE } w1 ; \]

\textbf{or}

\[ f \leftarrow w1 \text{ WHEN } s = '1' \text{ ELSE } w0 ; \]
Cascade of two multiplexers

VHDL:
Cascade of two multiplexers

VHDL:

\[
\begin{align*}
  f & <= w1 \text{ WHEN } s1 = '1' \text{ ELSE } \\
  & w2 \text{ WHEN } s2 = '1' \text{ ELSE } \\
  & w3 ;
\end{align*}
\]
4-to-1 Multiplexer

(a) Graphic symbol

(b) Truth table

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_0$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$w_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$w_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$w_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$w_3$</td>
</tr>
</tbody>
</table>
4-to-1 Multiplexer

(a) Graphic symbol

WITH s SELECT
  f <= w0 WHEN "00",
       w1 WHEN "01",
       w2 WHEN "10",
       w3 WHEN OTHERS ;
2-to-4 Decoder

(a) Truth table

<table>
<thead>
<tr>
<th>$En$</th>
<th>$w_1$</th>
<th>$w_0$</th>
<th>$y_3$</th>
<th>$y_2$</th>
<th>$y_1$</th>
<th>$y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

(b) Graphical symbol

\[ w \rightarrow w_1 \rightarrow y_3 \rightarrow y \]
\[ w \rightarrow w_0 \rightarrow y_2 \rightarrow y \]
\[ y_1 \rightarrow y \]
\[ y_0 \rightarrow y \]
# 2-to-4 Decoder

## (a) Truth table

<table>
<thead>
<tr>
<th>$En$</th>
<th>$w_1$</th>
<th>$w_0$</th>
<th>$y_3$</th>
<th>$y_2$</th>
<th>$y_1$</th>
<th>$y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

## (b) Graphical symbol

![Graphical symbol for 2-to-4 decoder]
2-to-4 Decoder

(a) Truth table

<table>
<thead>
<tr>
<th>$En$</th>
<th>$w_1$</th>
<th>$w_0$</th>
<th>$y_3$</th>
<th>$y_2$</th>
<th>$y_1$</th>
<th>$y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>$x$</td>
<td>$x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) Graphical symbol

$$\begin{align*}
Enw & \leq En \& w ; \\
\text{WITH } Enw \text{ SELECT} \\
y & \leq "0001" \text{ WHEN } "100", \\
& "0010" \text{ WHEN } "101", \\
& "0100" \text{ WHEN } "110", \\
& "1000" \text{ WHEN } "111", \\
& "0000" \text{ WHEN OTHERS } ;
\end{align*}$$
VHDL code for a 2-to-4 Decoder entity

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 ;
Encoders
Priority Encoder

\[
\begin{array}{c|cc|cc|c}
 w_3 & w_2 & w_1 & w_0 & y_1 & y_0 & z \\
 \hline
 0 & 0 & 0 & 0 & 0 & 0 & - \\
 0 & 0 & 0 & 1 & 0 & 0 & - \\
 0 & 0 & 1 & - & 0 & 0 & - \\
 0 & 1 & - & - & 0 & 0 & - \\
 1 & - & - & - & 0 & 0 & - \\
\end{array}
\]
Priority Encoder

\[
\begin{array}{cccc|ccc}
  w_3 & w_2 & w_1 & w_0 & y_1 & y_0 & z \\
  0   & 0   & 0   & 0   & d   & d   & 0   \\
  0   & 0   & 0   & 1   & 0   & 0   & 1   \\
  0   & 0   & 1   & -   & 0   & 1   & 1   \\
  0   & 1   & -   & -   & 1   & 0   & 1   \\
  1   & -   & -   & -   & 1   & 1   & 1   \\
\end{array}
\]
Priority Encoder

\[ w_3 \quad w_2 \quad w_1 \quad w_0 \quad y_1 \quad y_0 \quad z \]

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & d & d & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & - & 0 & 1 & 1 \\
0 & 1 & - & - & 1 & 0 & 1 \\
1 & - & - & - & 1 & 1 & 1 \\
\end{array}
\]

\[
y \leq \begin{cases} 
"11" & \text{WHEN } w(3) = '1' \text{ ELSE } \\
"10" & \text{WHEN } w(2) = '1' \text{ ELSE } \\
"01" & \text{WHEN } w(1) = '1' \text{ ELSE } \\
"00" & \text{ ELSE } \\
\end{cases}
\]

\[
z \leq '0' \text{ WHEN } w = "0000" \text{ ELSE } '1' ;
\]
VHDL code for a Priority Encoder entity

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 ;
Adders
Adder mod $2^{16}$
VHDL code for an Adder mod $2^{16}$

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY adder16 IS
  PORT ( X : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          Y : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
END adder16;

ARCHITECTURE dataflow OF adder16 IS
BEGIN
  S <= X + Y ;
END dataflow ;
16-bit Unsigned Adder
Operations on Unsigned Numbers

For operations on unsigned numbers

USE ieee.std_logic_unsigned.all
and
signals of the type
STD_LOGIC_VECTOR

OR

USE ieee.numeric_std.all
and
signals of the type
UNSIGNED
and conversion functions:
  std_logic_vector(), unsigned()
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;
VHDL code for a 16-bit Unsigned Adder

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY adder16 IS
  PORT ( Cin : IN STD_LOGIC;
         X   : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         Y   : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         S   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
         Cout : OUT STD_LOGIC ) ;
END adder16 ;

ARCHITECTURE dataflow OF adder16 IS
  SIGNAL Sum : STD_LOGIC_VECTOR(16 DOWNTO 0);
BEGIN
  Sum <= ('0' & X) + Y + Cin;
  S  <= Sum(15 DOWNTO 0);
  Cout <= Sum(16);
END dataflow ;
Addition of Unsigned Numbers (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY adder16 IS
  PORT ( Cin : IN STD_LOGIC;
          X : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          Y : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
          Cout : OUT STD_LOGIC ) ;
END adder16 ;
ARCHITECTURE dataflow OF adder16 IS
  SIGNAL Xu : UNSIGNED(15 DOWNTO 0);
  SIGNAL Yu: UNSIGNED(15 DOWNTO 0);
  SIGNAL Su : UNSIGNED(16 DOWNTO 0) ;
BEGIN
  Xu <= unsigned(X);
  Yu <= unsigned(Y);
  Su <= ('0' & Xu) + Yu + unsigned('0' & Cin) ;
  S <= std_logic_vector(Su(15 DOWNTO 0)) ;
  Cout <= Su(16) ;
END dataflow ;
ARCHITECTURE dataflow OF adder16 IS
  signal Sum: STD_LOGIC_VECTOR(16 DOWNTO 0) ;
BEGIN
  Sum <= std_logic_vector( unsigned('0' & X) + unsigned(Y) 
                   + unsigned('0' & Cin) ) ;
  S <= Sum(15 downto 0);
  Cout <= Sum(16) ;
END dataflow ;
Operations on Signed Numbers

For operations on signed numbers

USE ieee.numeric_std.all,
signals of the type
  SIGNED,
and conversion functions:
  std_logic_vector(), signed()

OR

USE ieee.std_logic_signed.all
and signals of the type
  STD_LOGIC_VECTOR
Multipliers
Unsigned vs. Signed Multiplication

Unsigned

\[\begin{array}{c}
1111 \\
\times 1111 \\
\hline
111000001
\end{array}\]

Signed

\[\begin{array}{c}
1111 \\
\times 1111 \\
\hline
000000001
\end{array}\]
8x8-bit Unsigned Multiplier

\[ a \times b \rightarrow c \rightarrow 16 \]
Multiplication of unsigned numbers

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

entity multiply is
  port(
    a : in STD_LOGIC_VECTOR(7 downto 0);
    b : in STD_LOGIC_VECTOR(7 downto 0);
    c : out STD_LOGIC_VECTOR(15 downto 0)
  );
end multiply;

architecture dataflow of multiply is
begin
  c <= a * b;
end dataflow;
8x8-bit Signed Multiplier
Multiplication of signed numbers

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

entity multiply is
    port(
        a : in STD_LOGIC_VECTOR(7 downto 0);
        b : in STD_LOGIC_VECTOR(7 downto 0);
        c : out STD_LOGIC_VECTOR(15 downto 0)
    );
end multiply;

architecture dataflow of multiply is
begin
    c <= a * b;
end dataflow;
8x8-bit Unsigned and Signed Multiplier

\[
\begin{array}{c}
\text{a} \\
\text{cu} \\
16
\end{array} \quad \begin{array}{c}
b \\
\text{cs} \\
16
\end{array}
\]
Multiplication of signed and unsigned numbers

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity multiply is
  port(
    a : in STD_LOGIC_VECTOR(7 downto 0);
    b : in STD_LOGIC_VECTOR(7 downto 0);
    cu : out STD_LOGIC_VECTOR(15 downto 0);
    cs : out STD_LOGIC_VECTOR(15 downto 0)
  );
end multiply;

architecture dataflow of multiply is
begin

  -- signed multiplication
  cs <= STD_LOGIC_VECTOR(SIGNED(a)*SIGNED(b));

  -- unsigned multiplication
  cu <= STD_LOGIC_VECTOR(UNSIGNED(a)*UNSIGNED(b));
end dataflow;
Multiplication of signed and unsigned numbers

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity multiply is
    port(
        a : in STD_LOGIC_VECTOR(7 downto 0);
        b : in STD_LOGIC_VECTOR(7 downto 0);
        cu : out STD_LOGIC_VECTOR(15 downto 0);
        cs : out STD_LOGIC_VECTOR(15 downto 0)
    );
end multiply;

architecture dataflow of multiply is
begin

    -- signed multiplication
    cs <= STD_LOGIC_VECTOR(SIGNED(a)*SIGNED(b));

    -- unsigned multiplication
    cu <= STD_LOGIC_VECTOR(UNSIGNED(a)*UNSIGNED(b));
end dataflow;
Comparators
4-bit Number Comparator

A > B

A

B

AgtB
4-bit Unsigned Number Comparator

AgtB <= '1' WHEN A > B ELSE '0' ;
VHDL code for a 4-bit **Unsigned** Number Comparator entity

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

ENTITY compare IS
    PORT ( A, B : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           AgtB : OUT STD_LOGIC ) ;
END compare ;

ARCHITECTURE dataflow OF compare IS
BEGIN
    AgtB <= '1' WHEN A > B ELSE '0' ;
END dataflow ;
```
VHDL code for a 4-bit **Signed** Number Comparator entity

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

ENTITY compare IS
    PORT ( A, B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
           AgtB : OUT STD_LOGIC );
END compare;

ARCHITECTURE dataflow OF compare IS
BEGIN
    AgtB <= '1' WHEN A > B ELSE '0';
END dataflow;
```
4-bit Unsigned Number Comparator
VHDL code for a 4-bit ** Unsigned** Number Comparator

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

ENTITY compare IS
  PORT (   A, B     : IN   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            AeqB, AgtB, AltB : OUT  STD_LOGIC ) ;
END compare ;

ARCHITECTURE dataflow OF compare IS
BEGIN
  AeqB <= '1' WHEN A = B ELSE '0' ;
  AgtB <= '1' WHEN A > B ELSE '0' ;
  AltB <= '1' WHEN A < B ELSE '0' ;
END dataflow ;
```
VHDL code for a 4-bit Signed Number Comparator

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY compare IS
  PORT ( A, B   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       AeqB, AgtB, AltB : OUT STD_LOGIC );
END compare;

ARCHITECTURE dataflow OF compare IS
BEGIN
  AeqB <= '1' WHEN A = B ELSE '0';
  AgtB  <= '1' WHEN A > B ELSE '0';
  AltB  <= '1' WHEN A < B ELSE '0';
END dataflow;
Arithmetic operations

Synthesizable arithmetic operations:

- Addition, +
- Subtraction, -
- Comparisons, >, >=, <, <=
- Multiplication, *
- Division by a power of 2, /2**6
  (equivalent to right shift)
Arithmetic operations

The result of synthesis of an arithmetic operation is a
- combinational circuit
- without pipelining.

The exact internal architecture used (and thus delay and area of the circuit) may depend on the timing constraints specified during synthesis (e.g., the requested maximum clock frequency).
Integer Types

Operations on signals of the integer types: INTEGER, NATURAL, and their subtypes, such as

```vhd
TYPE day_of_month IS RANGE 1 TO 31;
```

are synthesizable in the range

\[-(2^{31}-1) .. 2^{31} - 1\] for INTEGERs and their subtypes

\[0 .. 2^{31} - 1\] for NATURALs and their subtypes
Integer Types

Operations on signals (variables) of the integer types:

INTEGER, NATURAL,

are less flexible and more difficult to control than operations on signals (variables) of the type

STD_LOGIC_VECTOR
UNSIGNED
SIGNED, and thus

are recommended to be avoided by beginners.
ROM
ROM 8x16 example (1)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY rom IS
     PORT ( 
          Addr : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          Dout : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
     );

END rom;
ARCHITECTURE dataflow OF rom IS
SIGNAL temp: INTEGER RANGE 0 TO 7;
TYPE vector_array IS ARRAY (0 to 7) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
CONSTANT memory : vector_array :=
   ( X"800A",
     X"D459",
     X"A870",
     X"7853",
     X"650D",
     X"642F",
     X"F742",
     X"F548" );
BEGIN
   temp <= to_integer(unsigned(Addr));
   Dout <= memory(temp);
END dataflow;
ARCHITECTURE dataflow OF rom IS
TYPE vector_array IS ARRAY (0 to 7) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
CONSTANT memory : vector_array :=
  ( X"800A",
    X"D459",
    X"A870",
    X"7853",
    X"650D",
    X"642F",
    X"F742",
    X"F548");
BEGIN
  Dout <= memory(to_integer(unsigned(Addr)));
END dataflow;
Buffers
Tri-state Buffer

(a) A tri-state buffer

(b) Equivalent circuit

(c) Truth table

\[
\begin{array}{c|c|c}
    e & x & f \\
    \hline
    0 & 0 & 0 \\
    0 & 1 & 1 \\
    1 & 0 & 0 \\
    1 & 1 & 1 \\
\end{array}
\]
Tri-state Buffer

(a) A tri-state buffer

(b) Equivalent circuit

(c) Truth table

<table>
<thead>
<tr>
<th>$e$</th>
<th>$x$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Four types of Tri-state Buffers

\[ f \leq x \text{ WHEN (e = '1')} \text{ ELSE 'Z'}; \quad f \leq \text{not } x \text{ WHEN (e = '1')} \text{ ELSE 'Z'}; \]

\[ f \leq x \text{ WHEN (e = '0')} \text{ ELSE 'Z'}; \quad f \leq \text{not } x \text{ WHEN (e = '0')} \text{ ELSE 'Z'}; \]
Tri-state Buffer entity (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY tri_state IS
  PORT ( e: IN STD_LOGIC;
         x: IN STD_LOGIC;
         f: OUT STD_LOGIC
  );
END tri_state;
ARCHITECTURE dataflow OF tri_state IS
BEGIN
  f <= x WHEN (e = ‘1’) ELSE ‘Z’;
END dataflow;
MLU Example
MLU Block Diagram

Diagram shows a block diagram of a Multiplier and Interpolator Unit (MLU) with inputs A and B, and outputs Y, Y1, L0, and L1. The diagram includes AND gates, OR gates, and multiplexers (MUX_0 to MUX_4_1) used to control the flow of signals. The diagram also involves the use of negation gates (NEG_A, NEG_B, NEG_Y) to invert input signals.
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;
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;
Combinational Logic Synthesis for Beginners
Simple rules for beginners

For combinational logic, use only concurrent statements

- concurrent signal assignment \((\Leftarrow)\)
- conditional concurrent signal assignment (when-else)
- selected concurrent signal assignment (with-select-when)
Simple rules for beginners

For circuits composed of
- simple logic operations (logic gates)
- simple arithmetic operations (addition, subtraction, multiplication)
- shifts/rotations by a constant

use
  • concurrent signal assignment \( \leftarrow \)
Simple rules for beginners

For circuits composed of
  - multiplexers
  - decoders, encoders
  - tri-state buffers
use
• conditional concurrent signal assignment
  (when-else) (ending with ELSE)
• selected concurrent signal assignment
  (with-select-when)
  (ending with WHEN OTHERS;)


Example: VHDL code for a 4-to-1 MUX

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 ;
"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).
Left vs. right side of the assignment

Left side \( \leq \) Right side

\( \leq \) when-else
\nwith-select \( \leq \)

- Internal signals (defined in a given architecture)
- Ports of the mode
  - out
  - inout

Expressions including:
- Internal signals (defined in a given architecture)
- Ports of the mode
  - in
  - inout