EDA322
Digital Design

Lecture 3:
Combinational Logic - VHDL

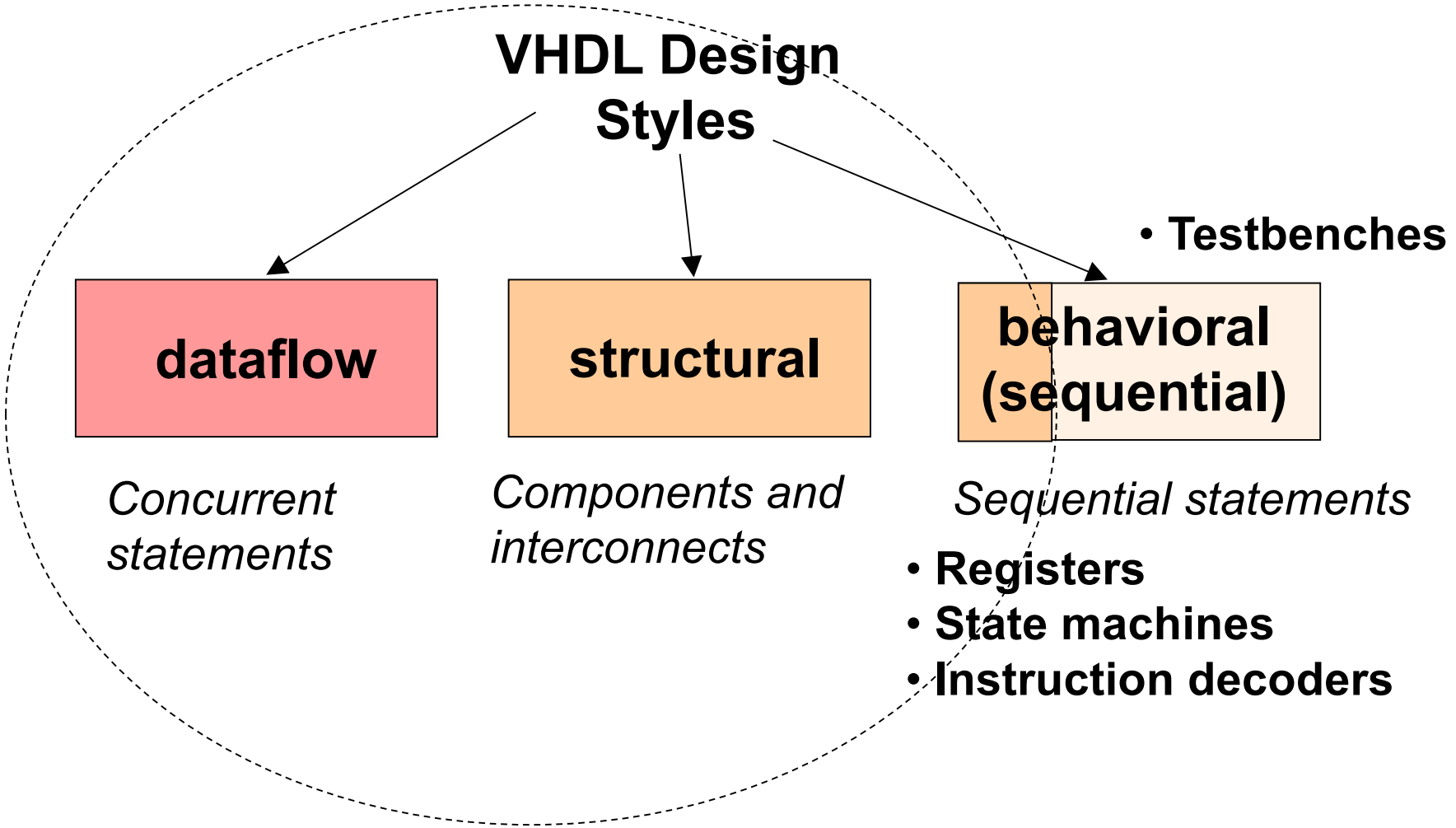
Ioannis Sourdis

Outline of Lecture 3

- VHDL styles
- Dataflow style (combinational logic)
  - Multiplexers, decoders, adders, multipliers, comparators, buffers, encoders
  - Concurrent statements

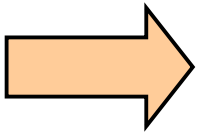
VHDL Design Styles

VHDL Design Styles



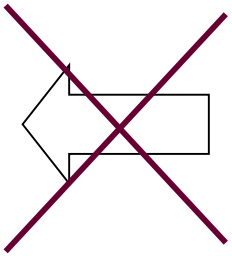
Synthesizable VHDL

Dataflow VHDL Design Style



VHDL code synthesizable

Dataflow VHDL Design Style



VHDL code synthesizable

Data-Flow VHDL

Concurrent Statements

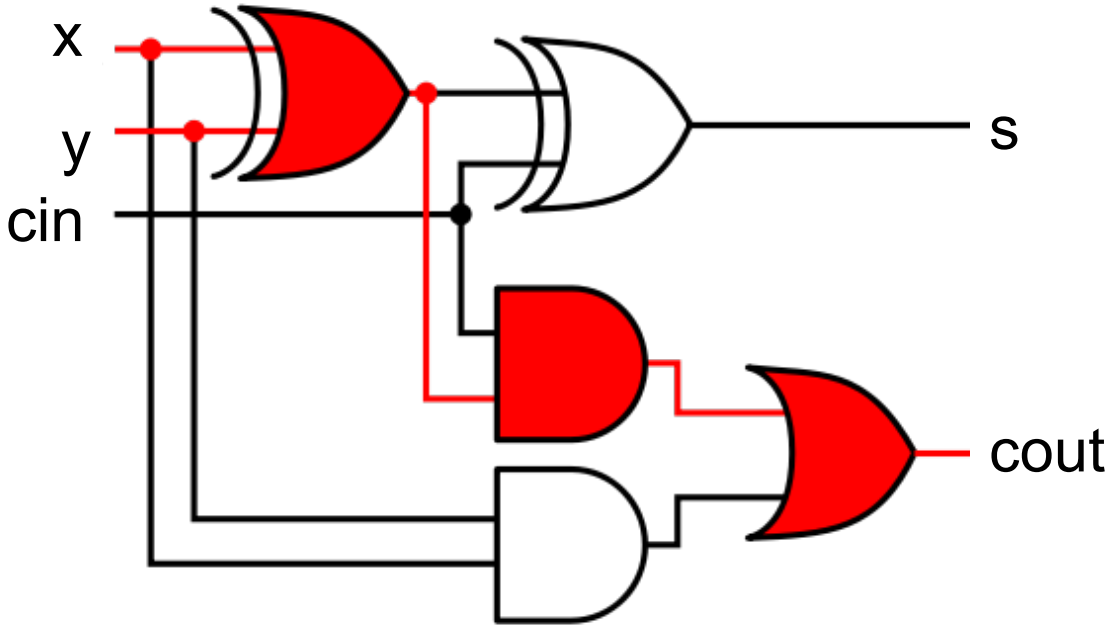
concurrent signal assignment
(⇐)

conditional concurrent signal assignment
(when-else)

selected concurrent signal assignment
(with-select-when)

generate scheme for equations
(for-generate)

Data-flow VHDL: Example



Data-flow VHDL: Example (1)

```
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 XOR y)) ;
END dataflow ;
```

Logic Operators

- Logic operators

only in VHDL-93

- Logic operators precedence

Highest

Lowest

not					
and	or	nand	nor	xor	xnor

No Implied Precedence

Wanted: y **= ab + cd**

Incorrect

y <= a **and** b **or** c **and** d ;

equivalent to

y <= ((a **and** b) **or** c) **and** d ;

equivalent to

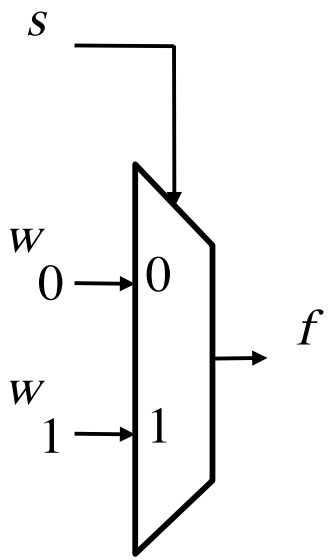
y = (ab + c)d

Correct

y <= (a **and** b) **or** (c **and** d) ;

Multiplexers

2-to-1 Multiplexer



(a) Graphical symbol

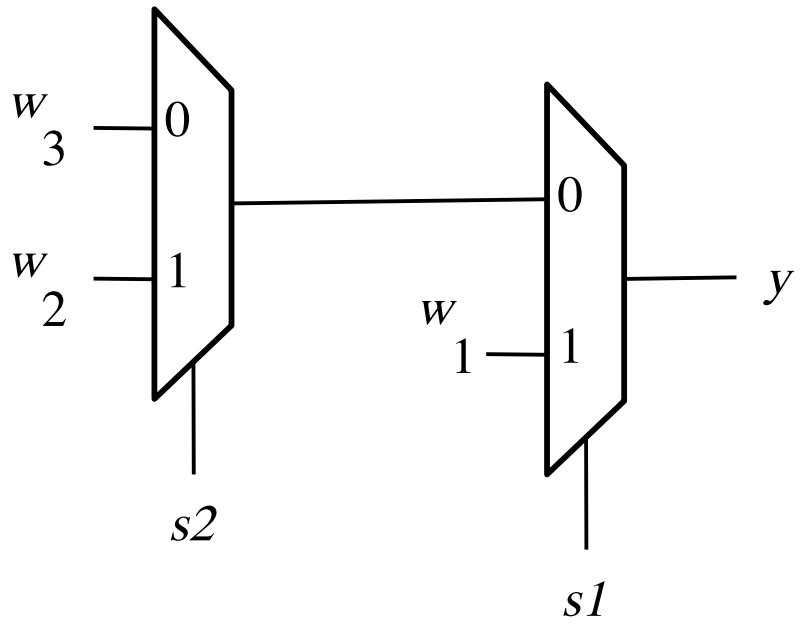
s	f
0	w_0
1	w_1

(b) Truth table

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

Operators

- Relational operators

=	/=	<	<=	>	>=

- Logic and relational operators precedence

Highest

Lowest

			not		
=	/=	<	<=	>	>=
and	or	nand	nor	xor	xnor

Priority of logic and relational operators

compare a = bc

Incorrect

… when a = b **and** c else …

equivalent to

… when (a = b) **and** c else …

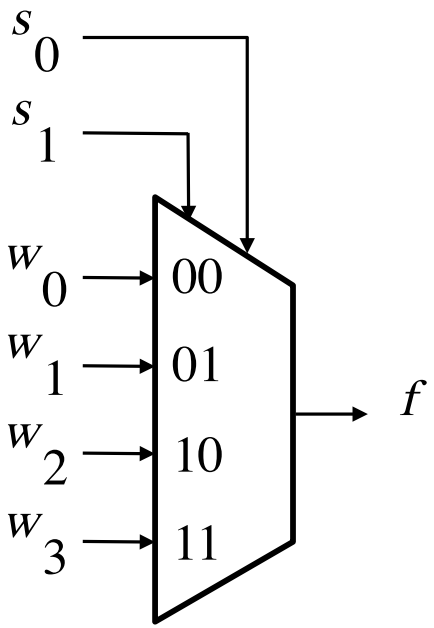
Correct

… when a = (b **and** c) else …

VHDL operators

	Operator Class	Operator
Highest precedence	Miscellaneous	**, ABS, NOT
	Multiplying	*, /, MOD, REM
	Sign	+, −
	Adding	+, −, &
	Shift	SLL, SRL, SLA, SRA, ROL, ROR
	Relational	=, /=, <, <=, >, >=
Lowest precedence	Logical	AND, OR, NAND, NOR, XOR, XNOR

4-to-1 Multiplexer



(a) Graphic symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

VHDL code for a 4-to-1 Multiplexer

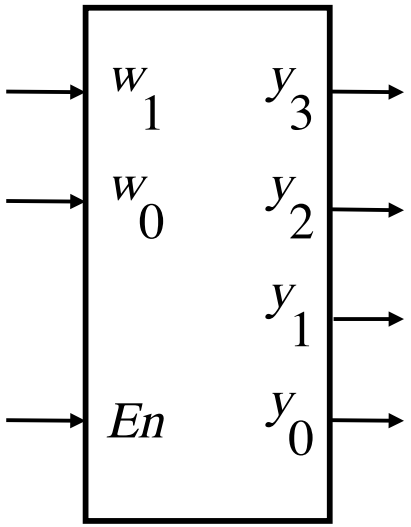
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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

En	w_1	w_0	y_3	y_2	y_1	y_0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	x	x	0	0	0	0

(a) Truth table



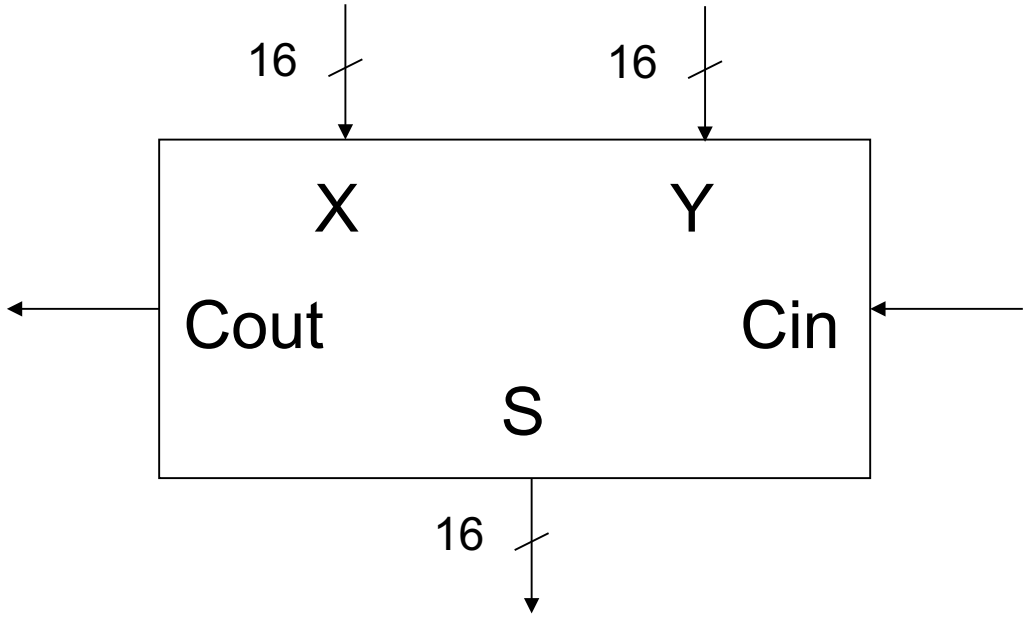
(b) Graphical symbol

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
        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 ;
```

Adders

16-bit Unsigned Adder



Operations on Unsigned Numbers

For operations on unsigned numbers

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

OR

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

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 ;
```

Addition of Unsigned Numbers (2)

```
ARCHITECTURE dataflow OF adder16 IS
    SIGNAL  Xu : UNSIGNED(16 DOWNTO 0);
    SIGNAL  Yu : UNSIGNED(16 DOWNTO 0);
    SIGNAL  Su : UNSIGNED(16 DOWNTO 0) ;
    SIGNAL  Cinu : UNSIGNED(1 DOWNTO 0);
BEGIN
    Xu   <= unsigned('0' & X);
    Yu   <= unsigned( Y);
    Cinu <= unsigned(Cin);
    Su   <= Xu + Yu + Cinu ;
    S    <= std_logic_vector(Su(15 DOWNTO 0)) ;
    Cout <= Su(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

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;

General role:

Use the numeric library, avoid the signed and unsigned libraries.

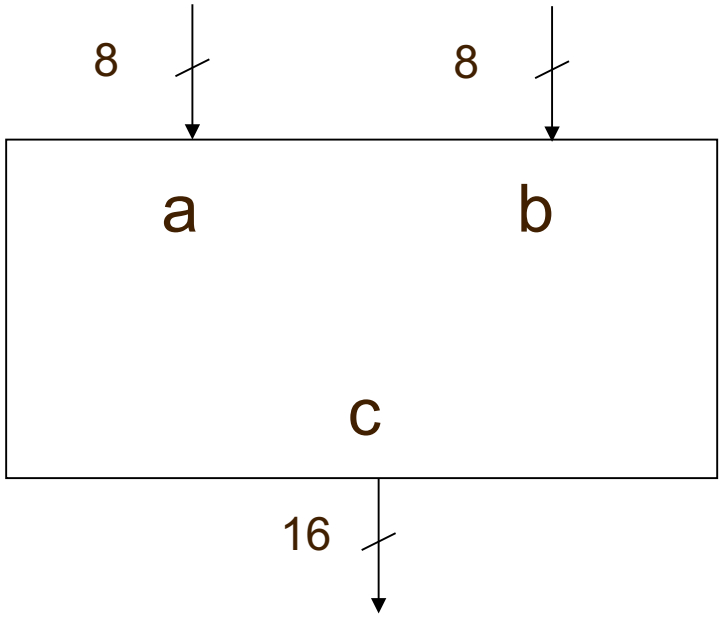
Signed and unsigned libraries have problems when used together in the same file!

Multipliers

Unsigned vs. Signed Multiplication

Unsigned		Signed	
1111	15	1111	-1
x 1111	x 15	x 1111	x -1
11100001	225	00000001	1

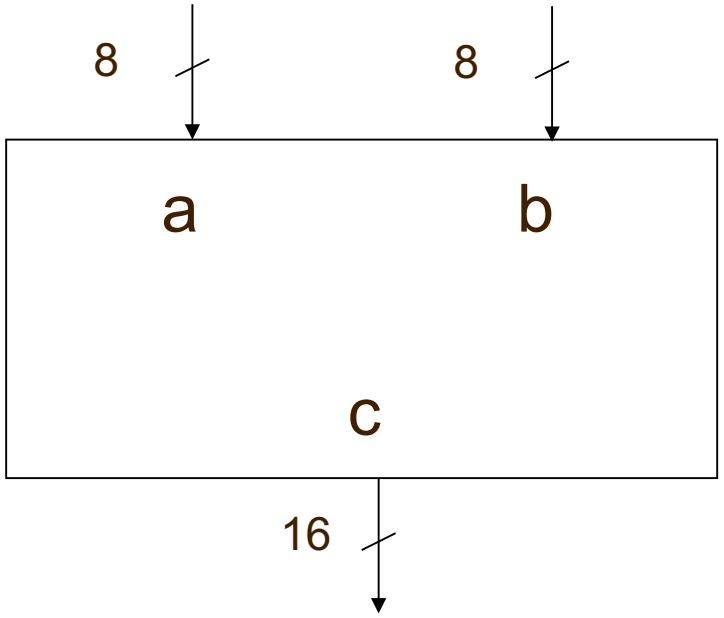
8x8-bit Unsigned Multiplier



Multiplication of 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);
            c : out STD_LOGIC_VECTOR(15 downto 0)
         );
end multiply;

architecture dataflow of multiply is
begin

    c <= STD_LOGIC_VECTOR(UNSIGNED(a)*UNSIGNED(b))

end dataflow;
```

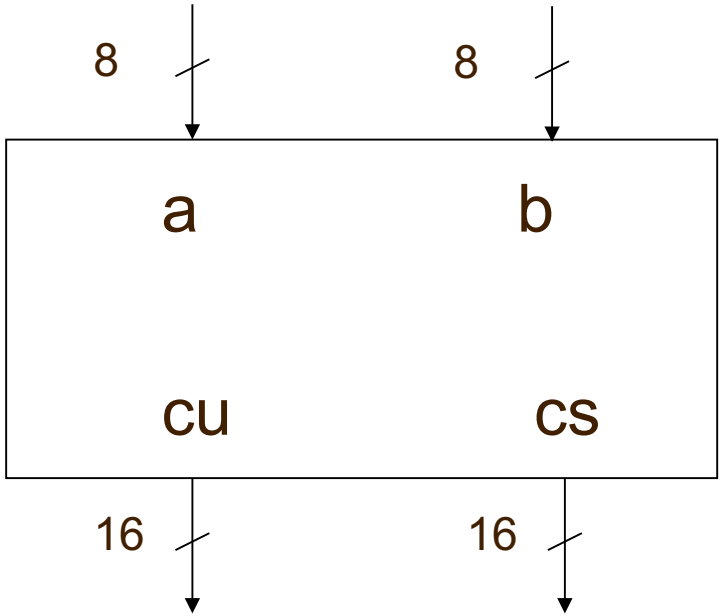
8x8-bit Signed Multiplier



Multiplication of signed 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);
            c : out STD_LOGIC_VECTOR(15 downto 0)
         );
end multiply;

architecture dataflow of multiply is
begin

    c <= STD_LOGIC_VECTOR(SIGNED(a)*SIGNED(b))

end dataflow;
```

8x8-bit Unsigned and Signed Multiplier



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

VHDL code for a 4-bit **Unsigned** Number Comparator

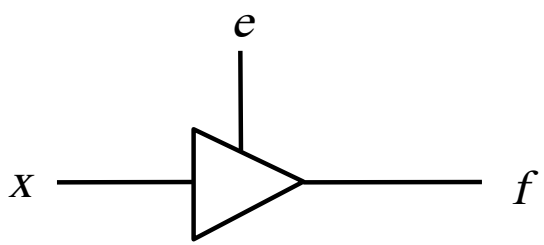
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;

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 unsigned(A)=unsigned(B) ELSE '0' ;
      AgtB <= '1' WHEN unsigned(A)>unsigned(B) ELSE '0' ;
      AltB <= '1' WHEN unsigned(A)<unsigned(B) ELSE '0' ;
END dataflow ;
```

VHDL code for a 4-bit **Signed** Number Comparator

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.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  signed(A)= signed(B) ELSE '0' ;
      AgtB  <= '1' WHEN  signed(A)>signed(B) ELSE '0' ;
      AltB  <=  '1'  WHEN  signed(A)<signed(B) ELSE '0' ;
END dataflow ;
```

Buffers

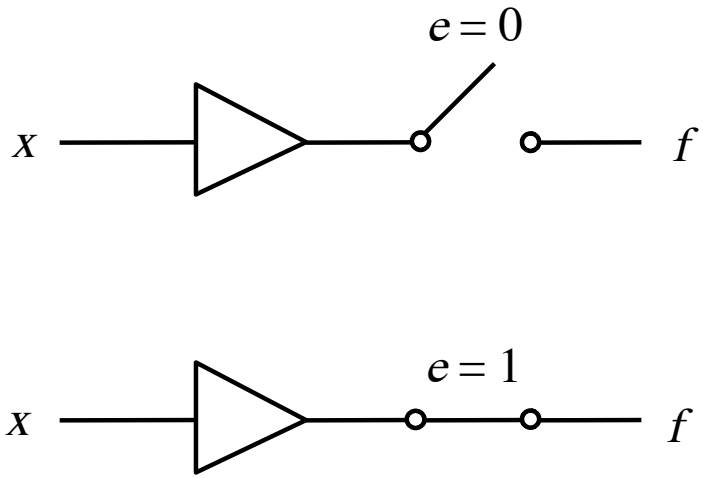
Tri-state Buffer



(a) A tri-state buffer

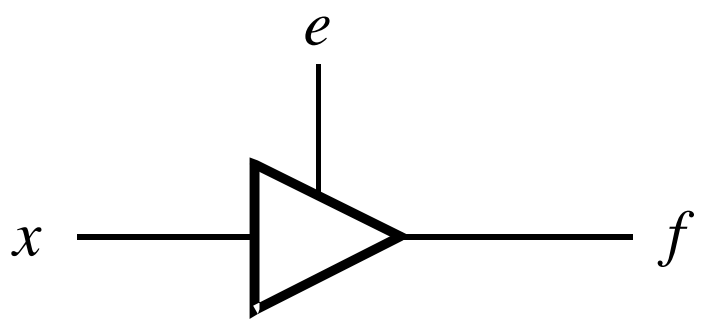
e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

(c) Truth table

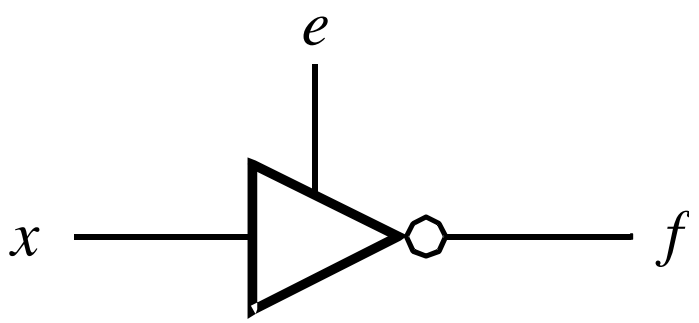


(b) Equivalent circuit

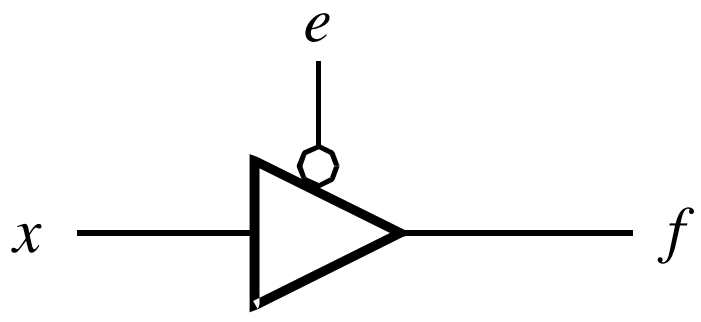
Four types of Tri-state Buffers



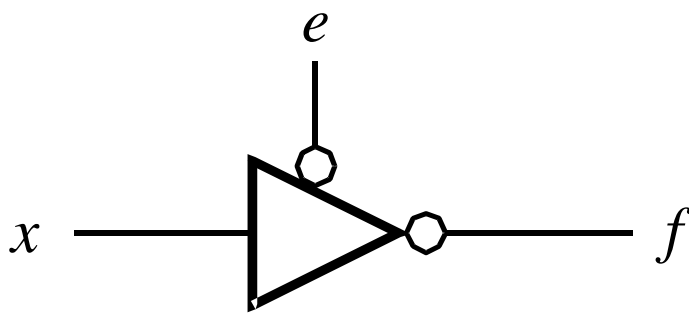
(a)



(b)



(c)



(d)

Tri-state Buffer – example (1)

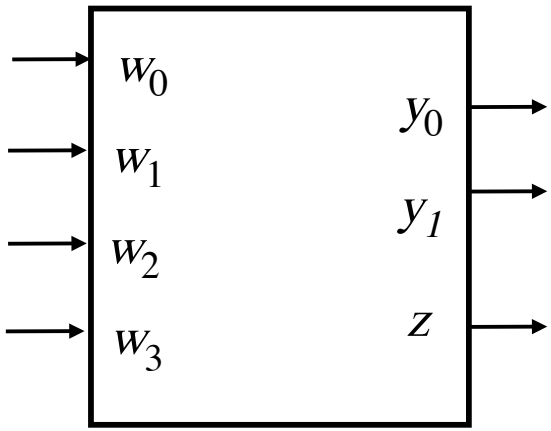
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

Tri-state Buffer – example (2)

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

Encoders

Priority Encoder



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	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

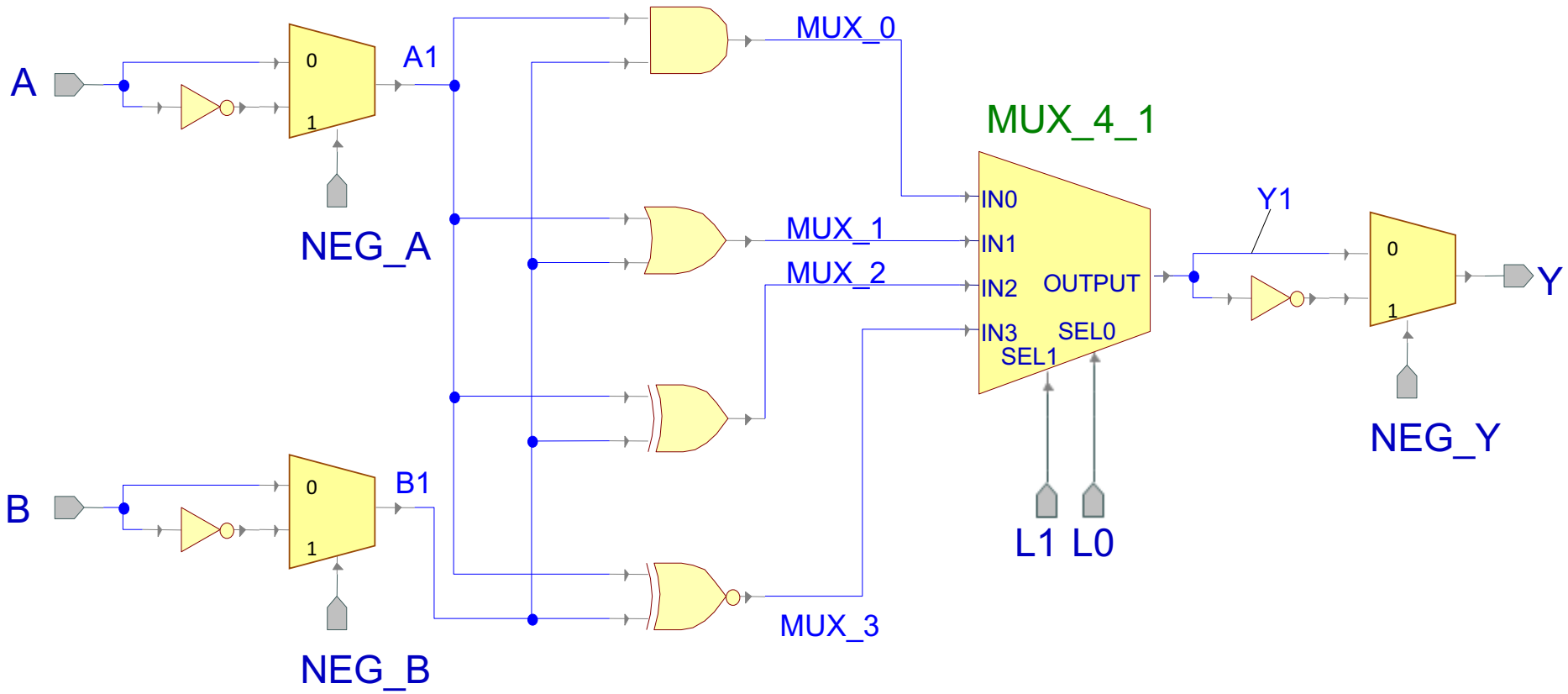
VHDL code for a Priority Encoder

```
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 ;
```

Describing Combinational Logic Using Dataflow Design Style

Multiple Logical Unit (MLU) Example

MLU Block Diagram



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);
```

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;
```

Logic Implied Most Often by Conditional and Selected Concurrent Signal Assignments

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)

Conditional concurrent signal assignment

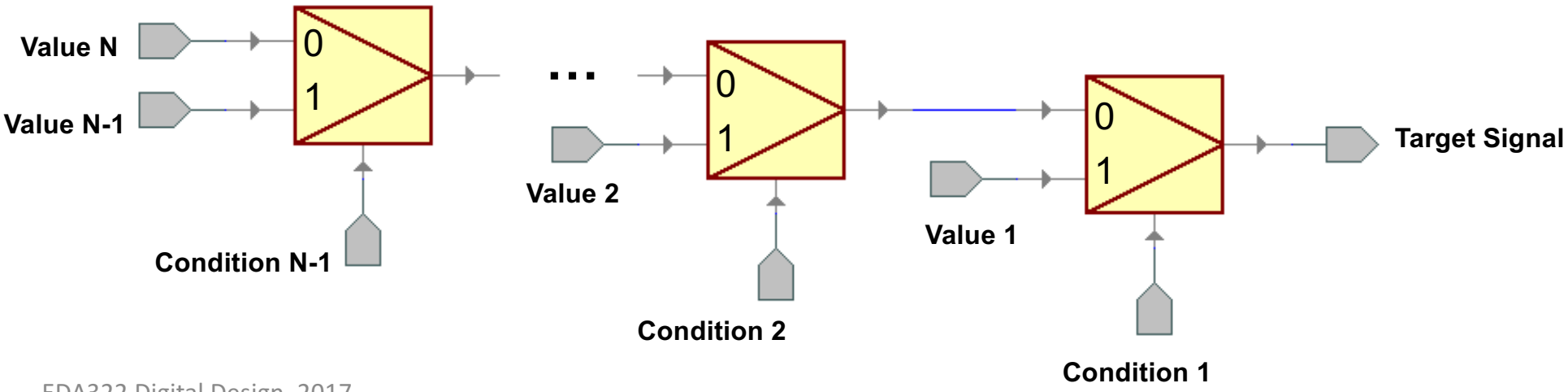
When - Else

```
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;
```



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

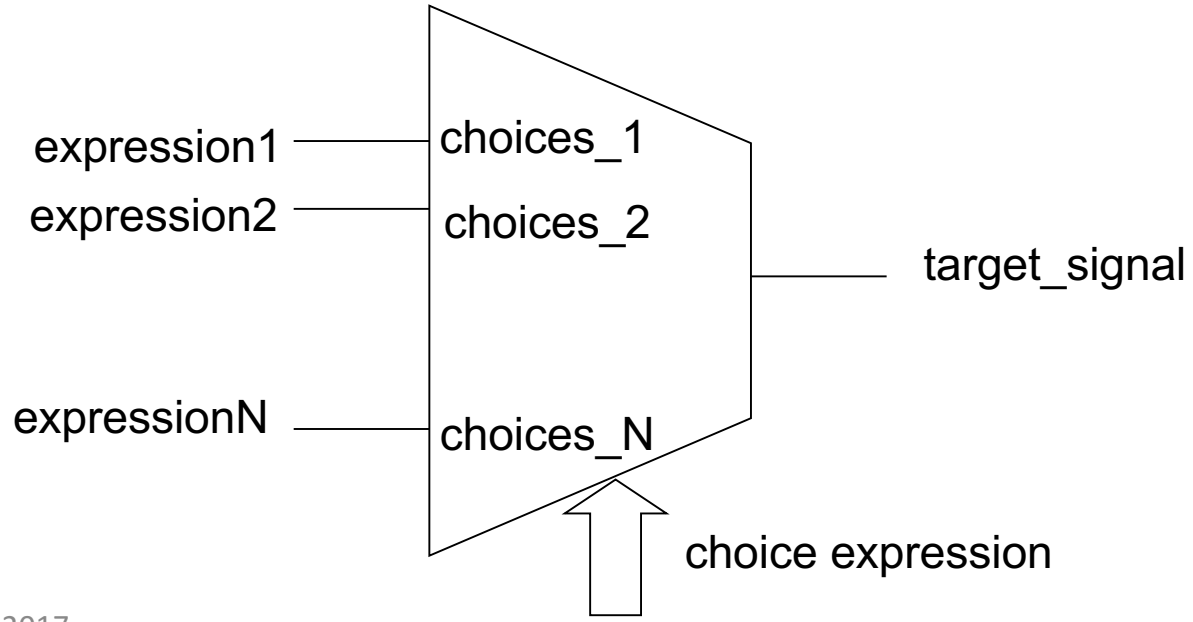
With –Select-When

```
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;
```

Summary of Lecture 3

- VHDL styles
- Dataflow style (combinational logic)
  - Multiplexers, decoders, adders, multipliers, comparators, buffers, encoders
  - Concurrent statements

- Book Chapters: 3.6, 7.1.5-7.1.8

- Next Lecture 4:
  - Lab Processor, VHDL for regular structures