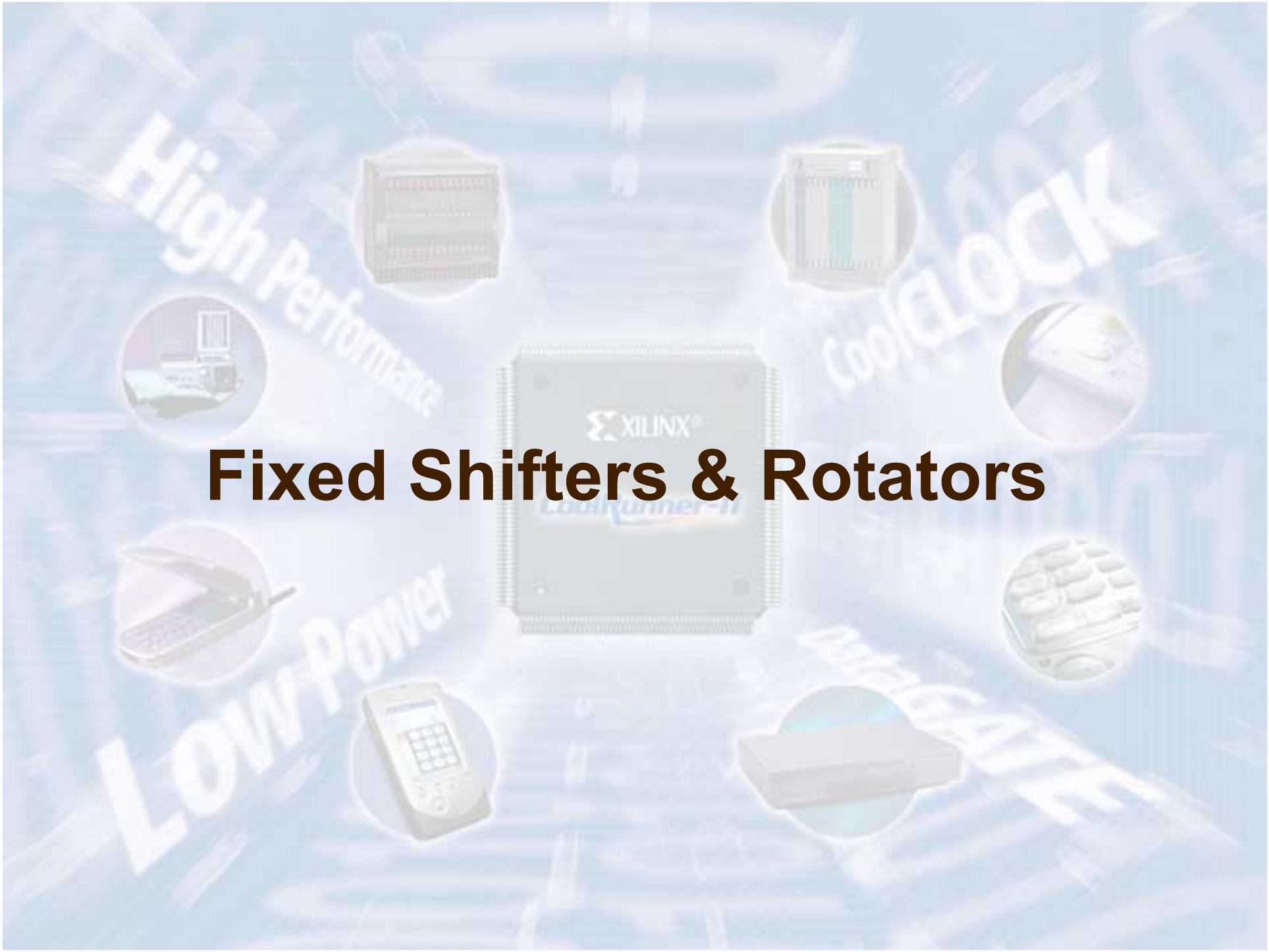
ECE 545
Lecture 5

Data Flow Description of Combinational-Circuit Building Blocks

Required reading

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

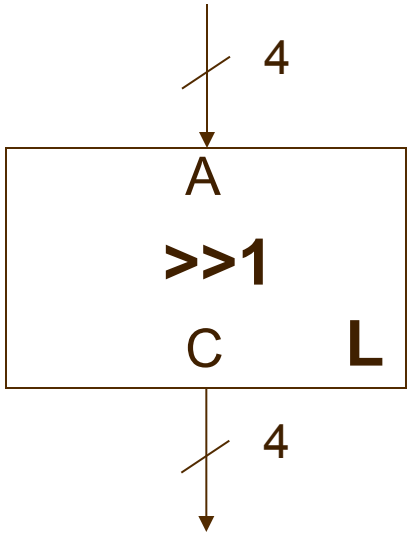
Chapter 7, Combinational Circuit Design: Practice

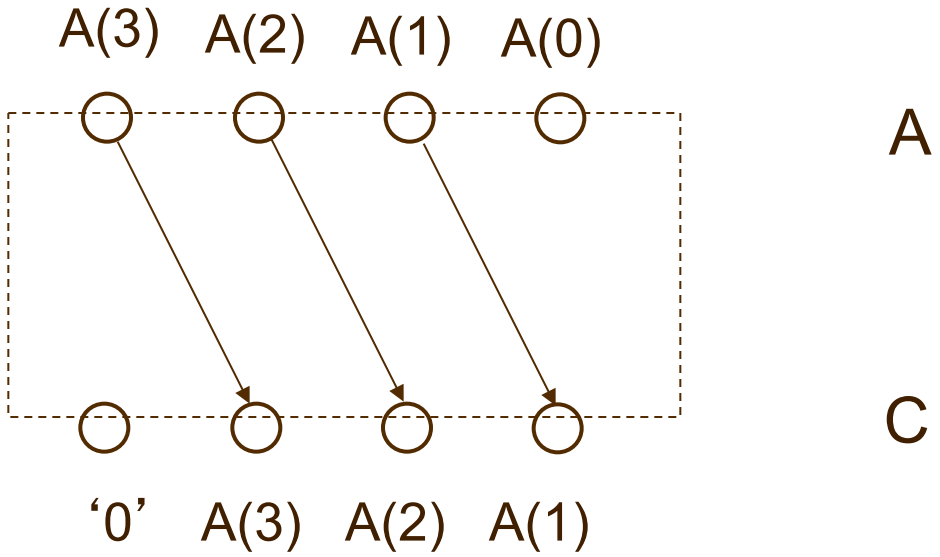


Fixed Shifters & Rotators

Fixed Logical Shift Right in VHDL

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

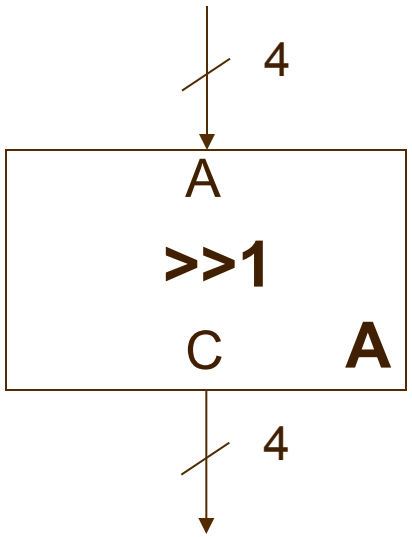


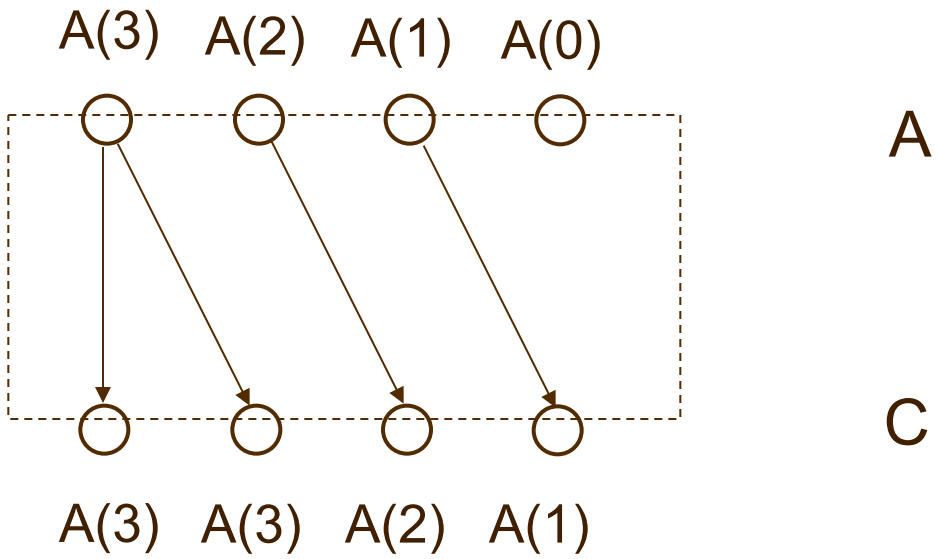


C =

Fixed Arithmetic Shift Right in VHDL

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

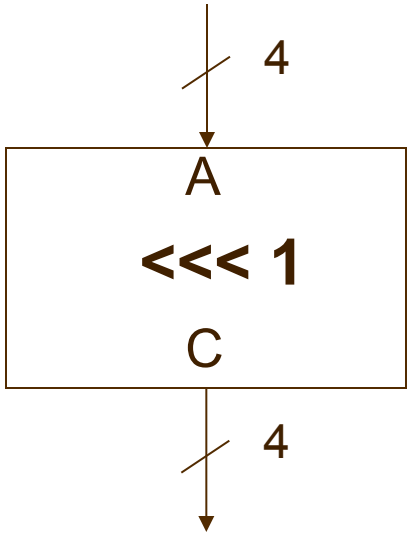


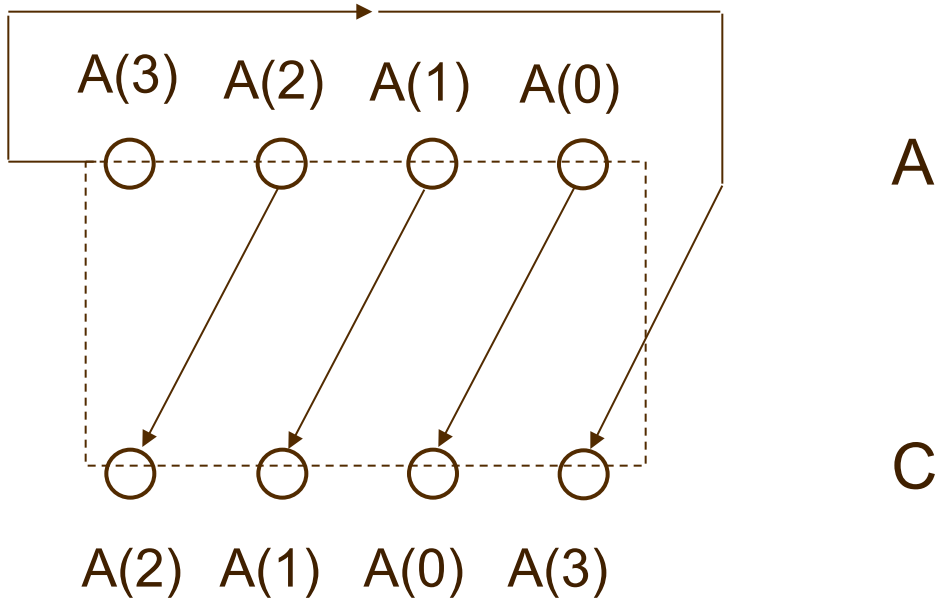


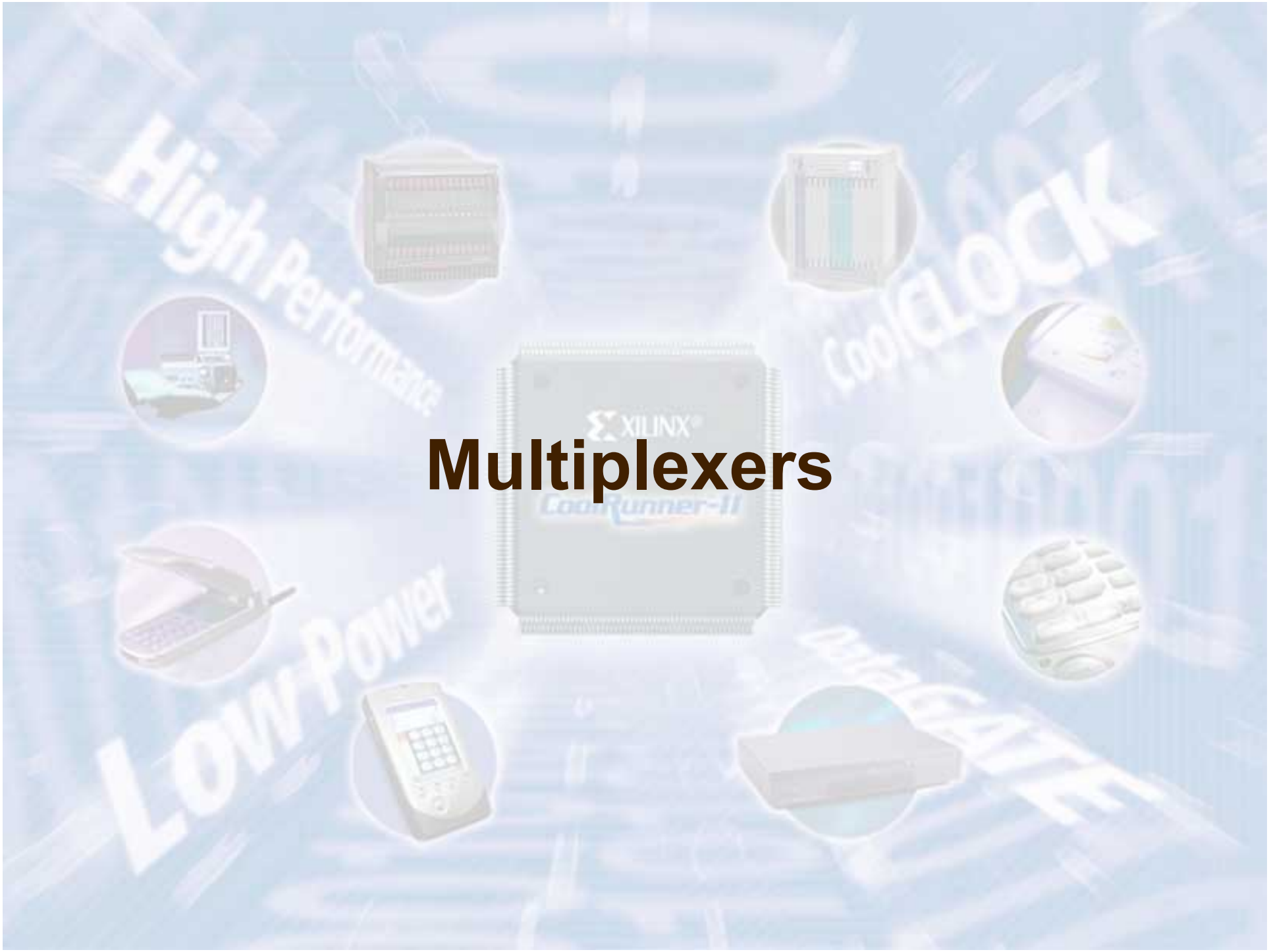
C =

Fixed Rotation in VHDL

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

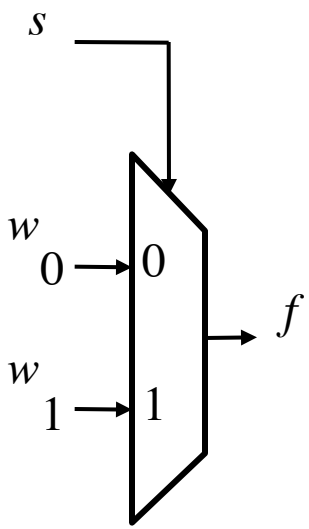






Multiplexers

2-to-1 Multiplexer



(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table

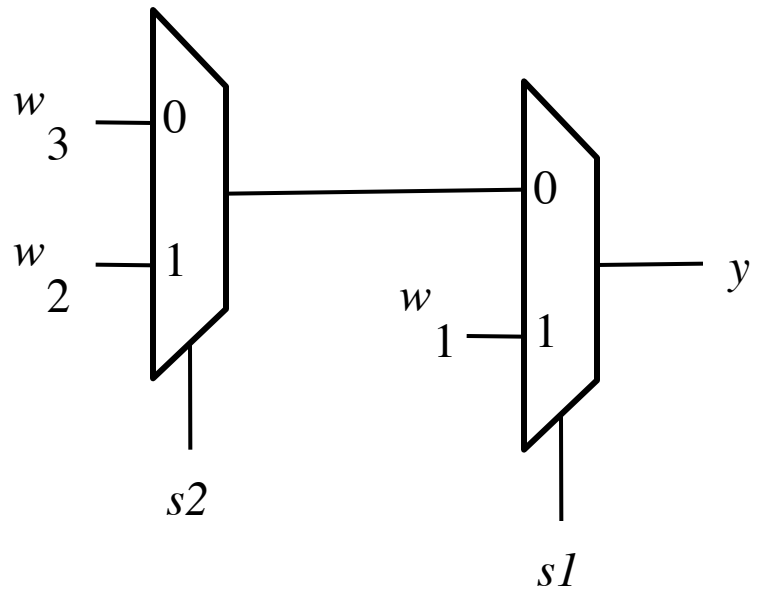
VHDL code for a 2-to-1 Multiplexer

```
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



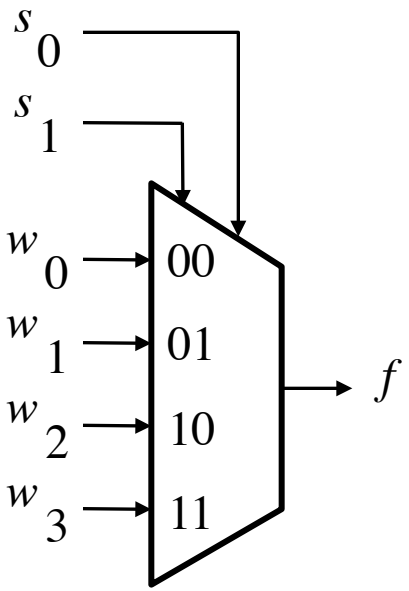
VHDL code for a cascade of two multiplexers

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

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

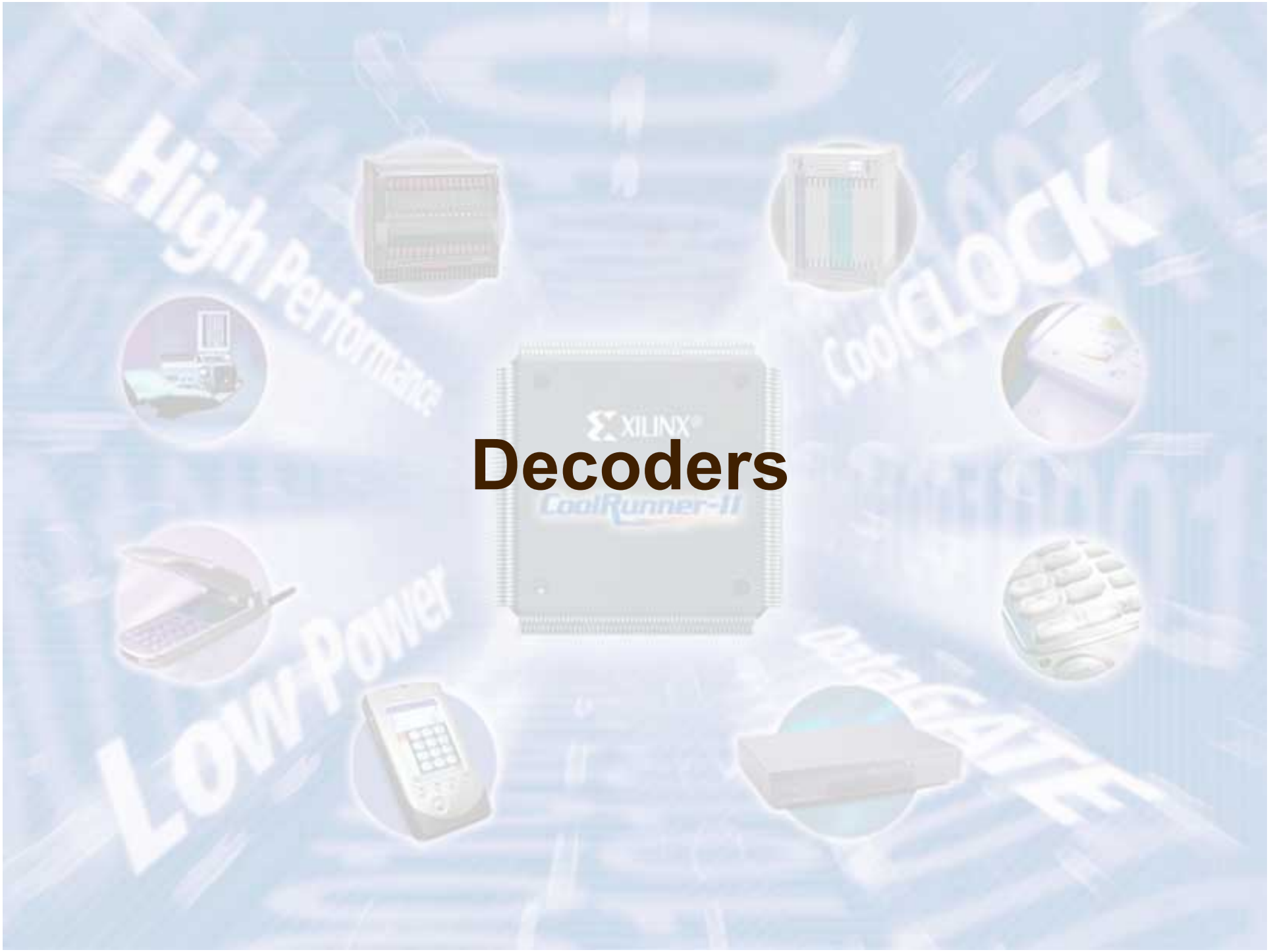
(b) Truth table

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 ;
```

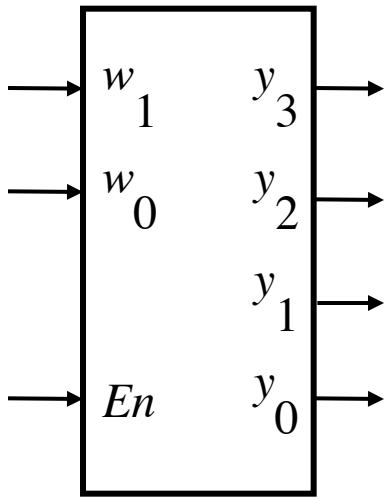


Decoders

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



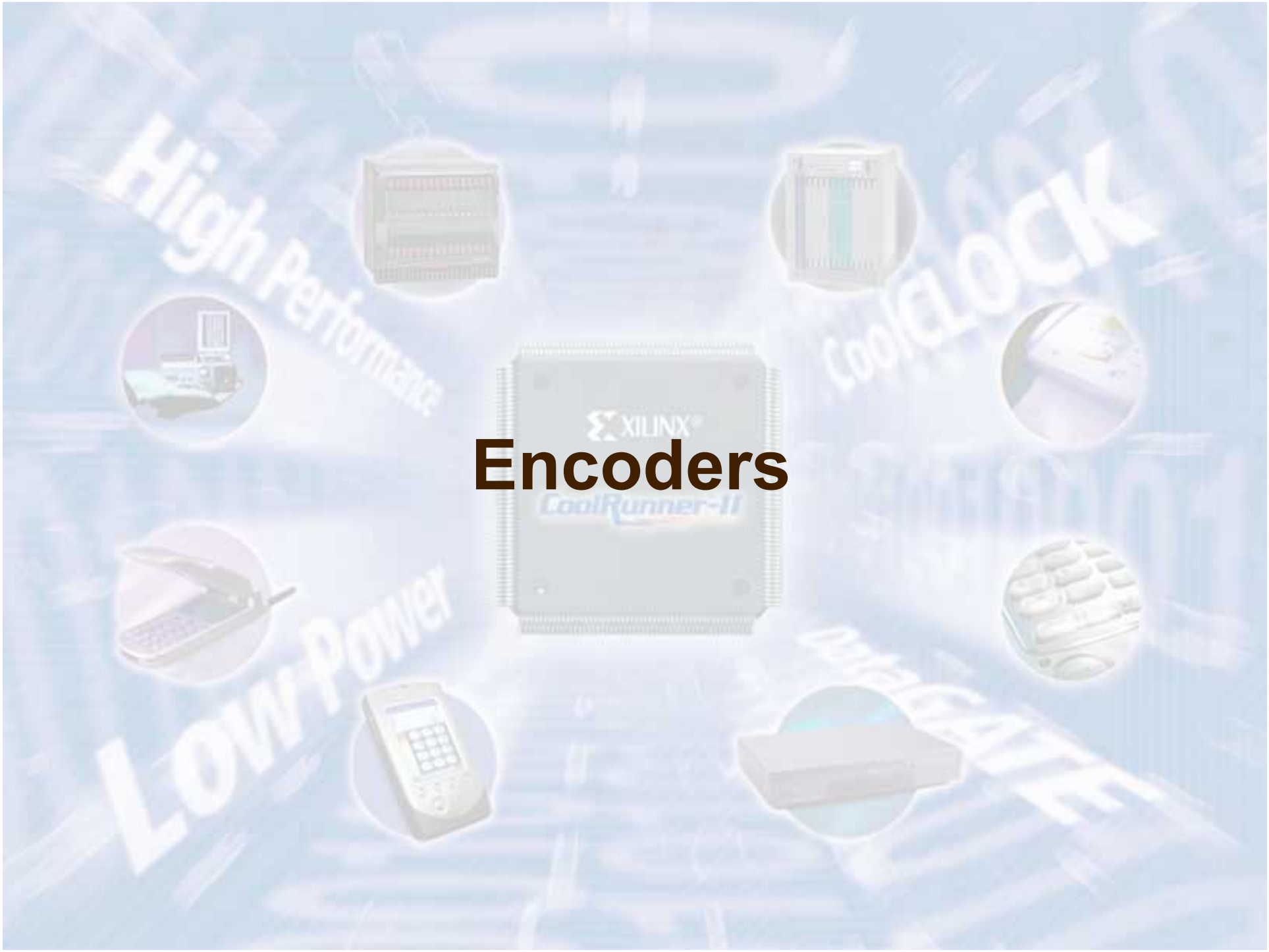
(b) Graphical symbol

VHDL code for a 2-to-4 Decoder

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

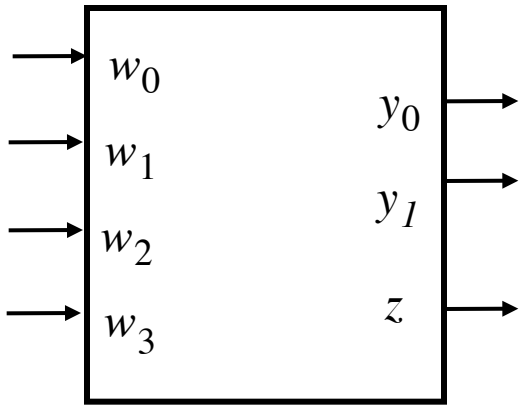
ENTITY dec2to4 IS
      PORT (    w     : IN       STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                En    : IN       STD_LOGIC ;
                y     : OUT      STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END dec2to4 ;

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 ;
```



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 ;

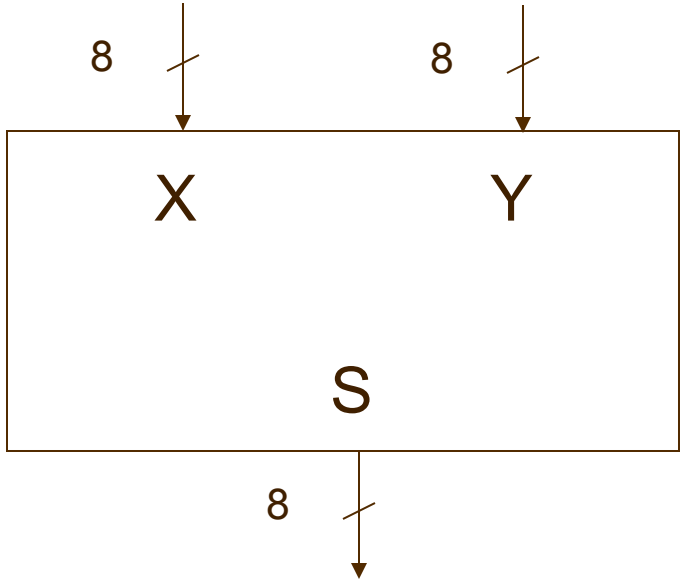
ENTITY priority IS
    PORT ( w    : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           y    : OUT    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
           z    : OUT    STD_LOGIC ) ;
END priority ;

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 ;
```



Adders

Adder mod 2^8



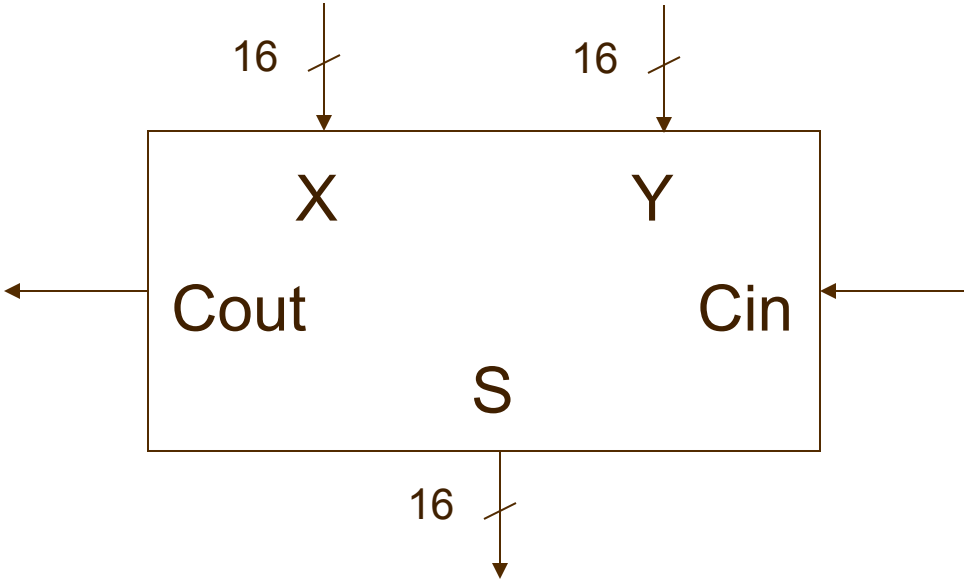
VHDL code for an Adder mod 2^8

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

ENTITY adder16 IS
      PORT (   X                    : IN      STD_LOGIC_VECTOR(7 DOWNTO 0) ;
               Y                    : IN      STD_LOGIC_VECTOR(7 DOWNTO 0) ;
               S                    : OUT     STD_LOGIC_VECTOR(7 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,
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 ;
```

Addition of Unsigned Numbers (2)

```
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 ;
```

Operations on Signed Numbers

For operations on signed numbers

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

OR

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



Multipliers

Unsigned vs. Signed Multiplication

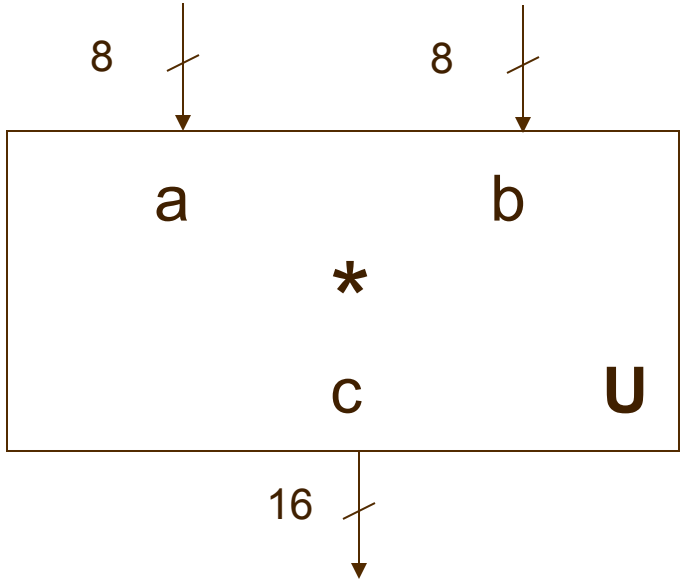
Unsigned

```
    1111        15
  x 1111      x 15
--------    ------
11100001       225
```

Signed

```
    1111        -1
  x 1111      x -1
--------    ------
00000001         1
```

8x8-bit Unsigned Multiplier



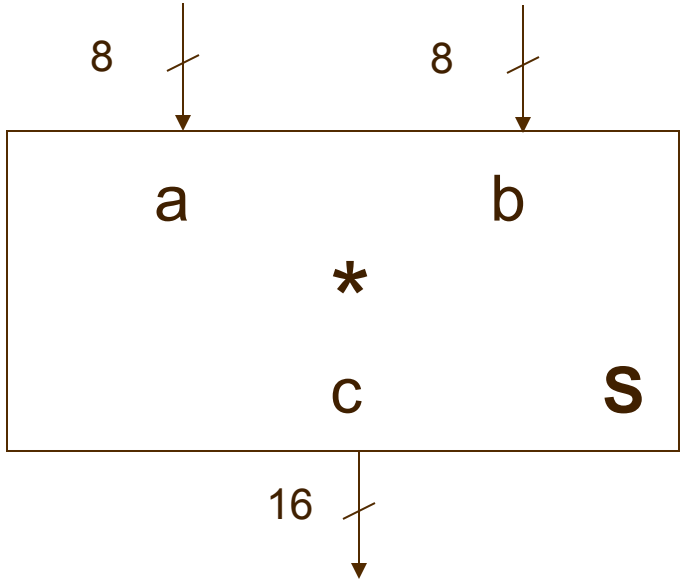
Multiplication of unsigned numbers

```
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



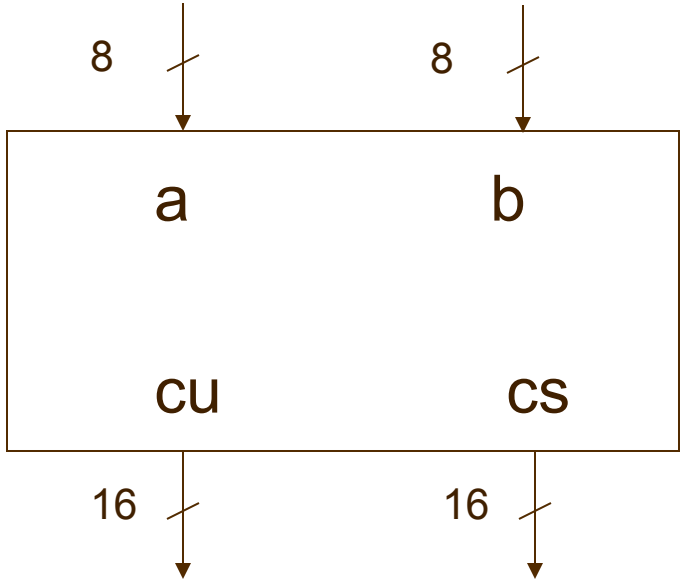
Multiplication of signed numbers

```
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



Multiplication of signed and unsigned numbers

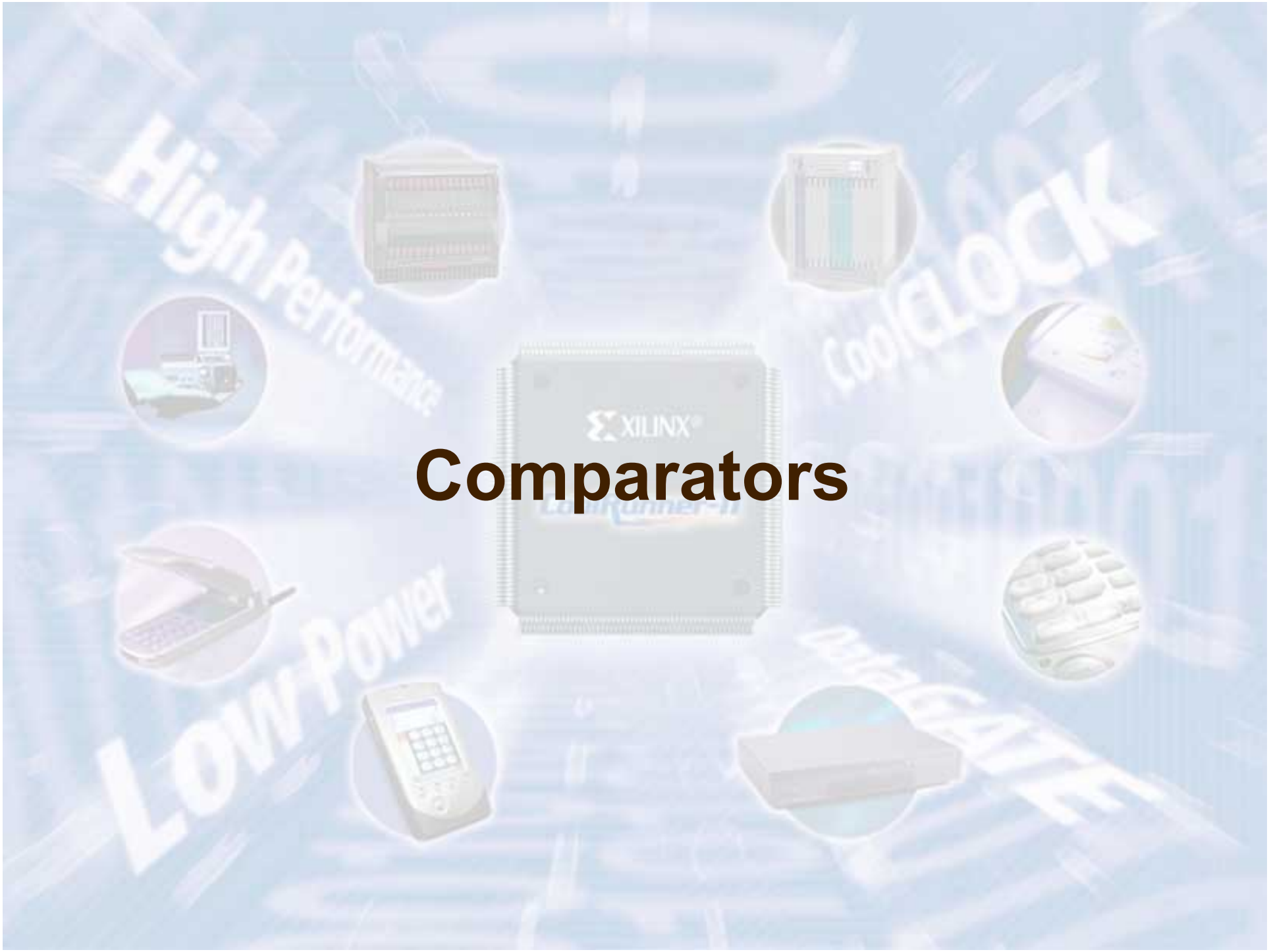
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all ;

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 Unsigned Number Comparator



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

```
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 sybtypes, such as

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

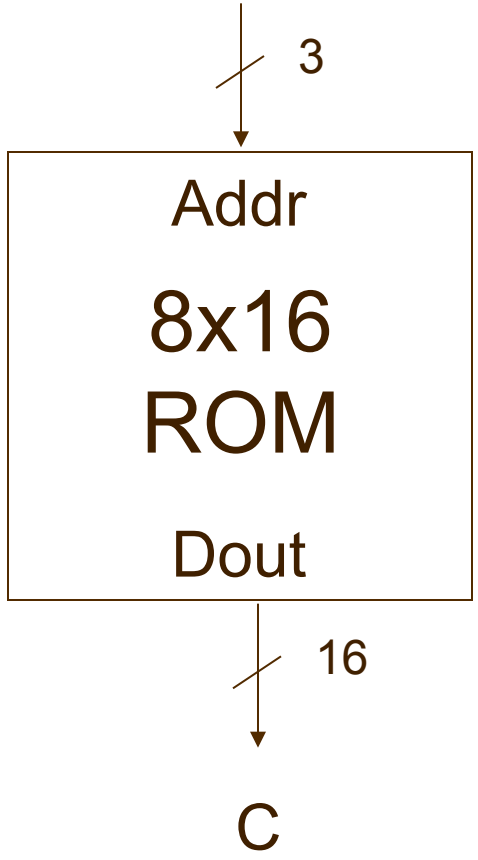
UNSIGNED

SIGNED, and thus

are recommened to be avoided by beginners.


ROM
XILINX
CoolRunner-II
High Performance
Low Power

8-bit Variable Rotator Left



Instruction ROM 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)
            );

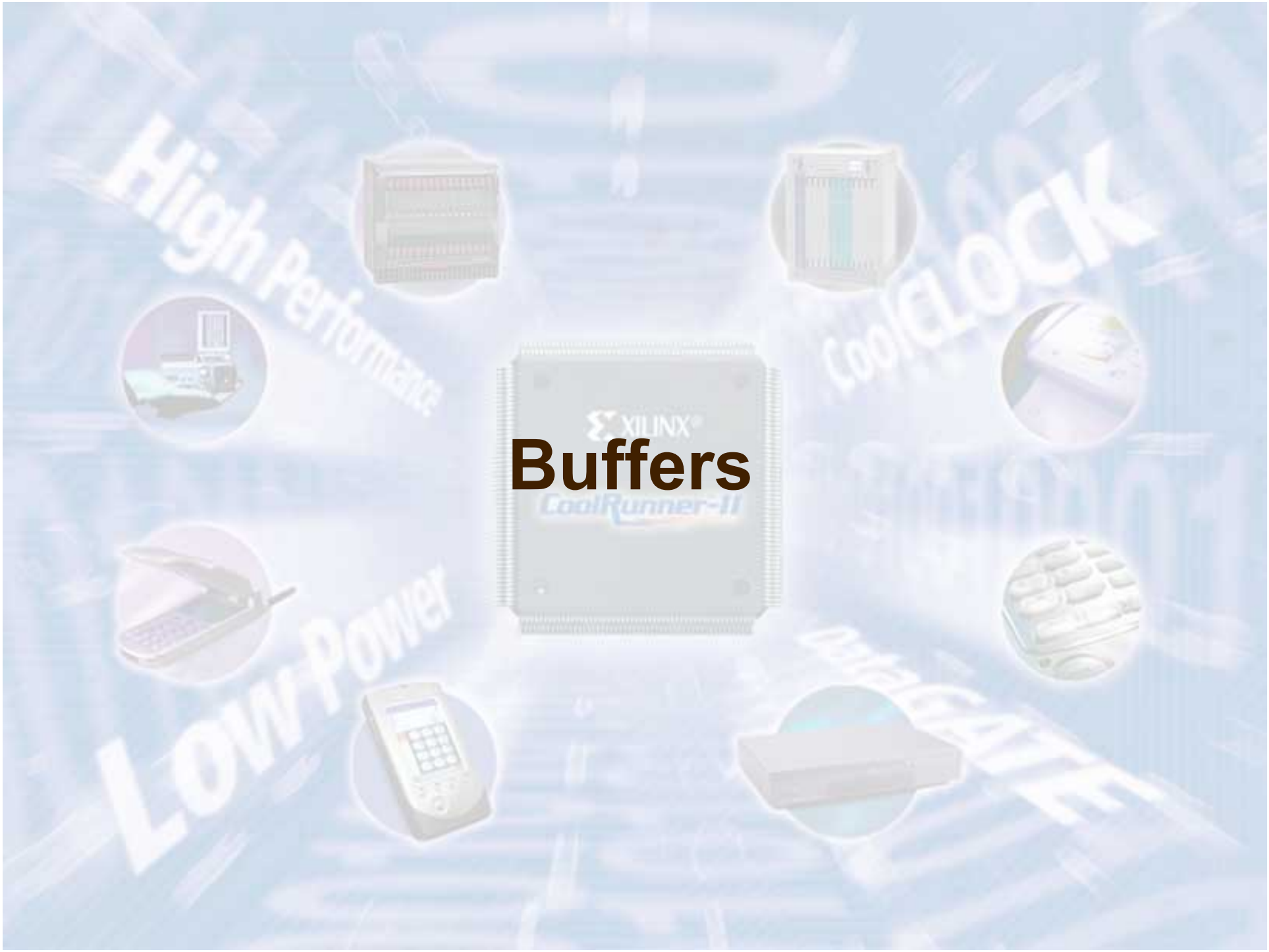
END rom;
```

Instruction ROM example (2)

```
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

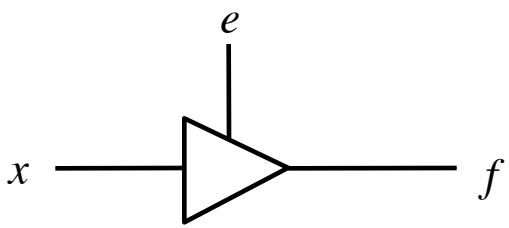
    temp <= to_integer(unsigned(Addr));
    Dout <= memory(temp);

END dataflow;
```



Buffers

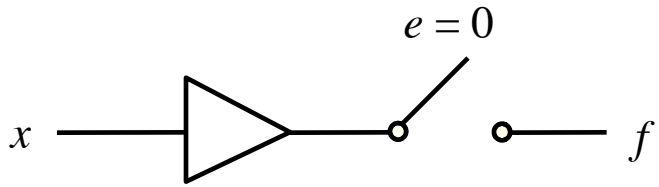
Tri-state Buffer

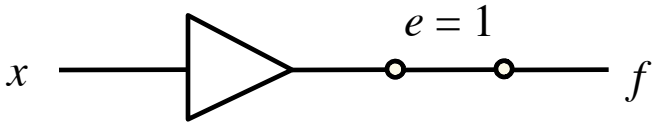


(a) A tri-state buffer

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

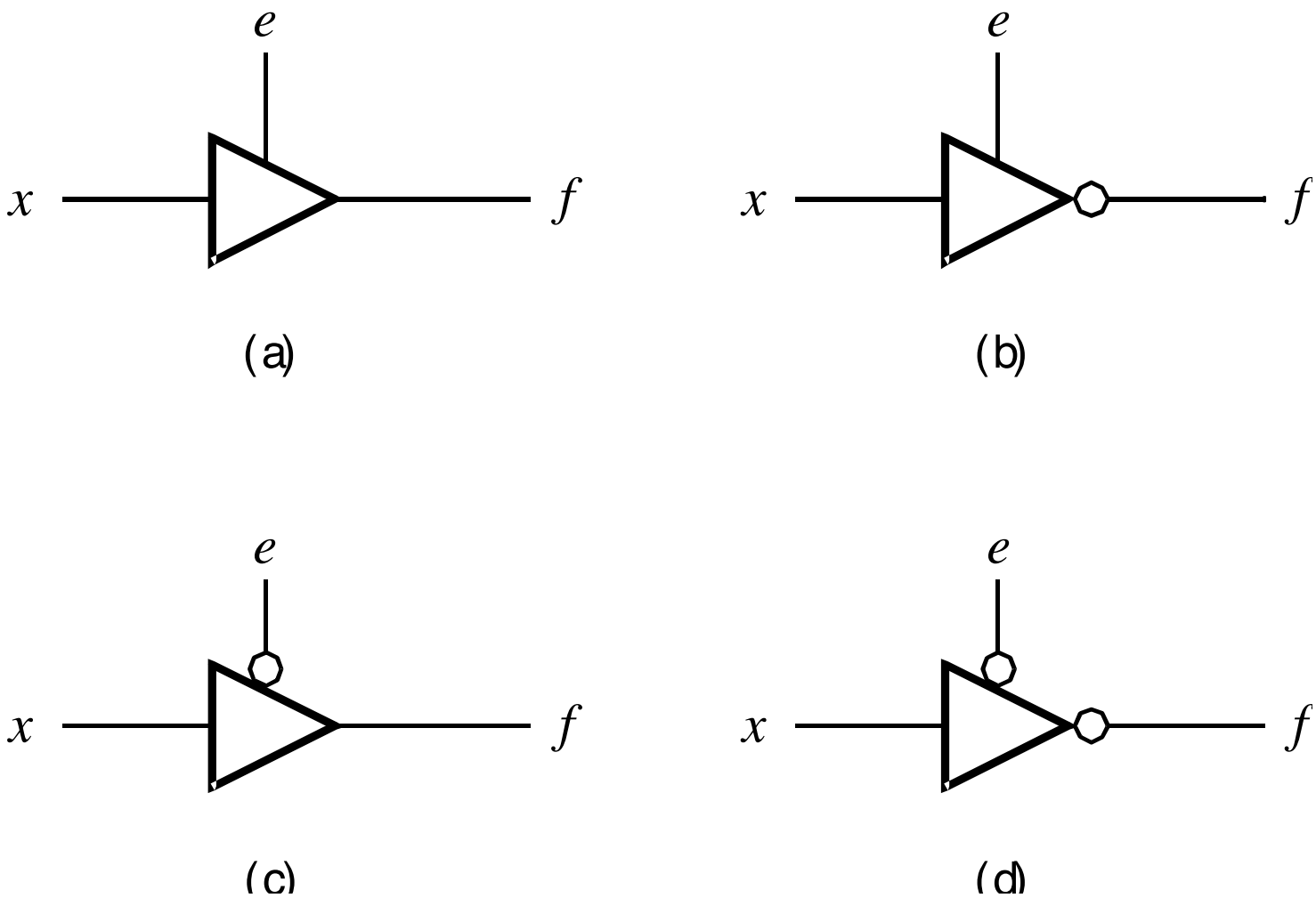
(c) Truth table





(b) Equivalent circuit

Four types of Tri-state Buffers



Tri-state Buffer – example (1)

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

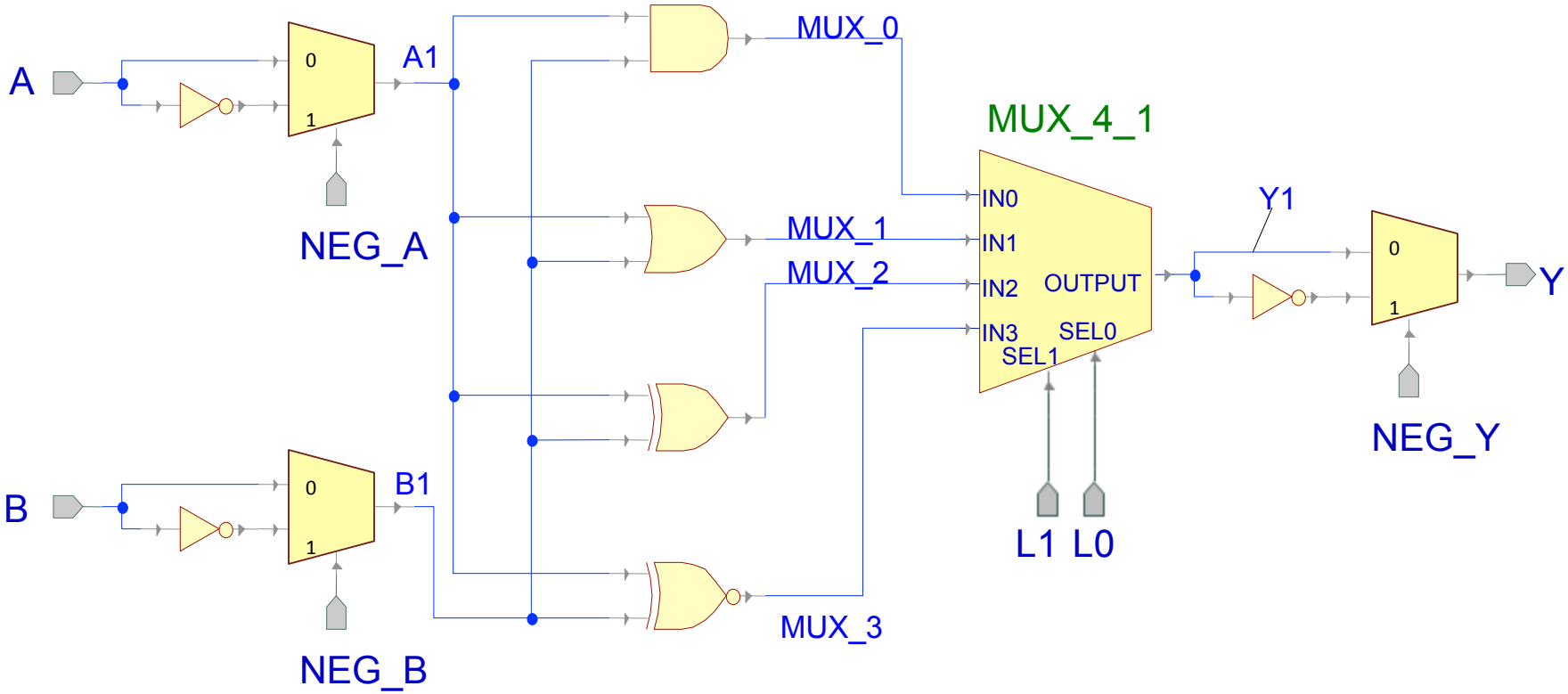
ENTITY tri_state IS
   PORT ( ena:    IN STD_LOGIC;
                input:  IN STD_LOGIC;
                output: OUT STD_LOGIC
               );
END tri_state;
```

Tri-state Buffer – example (2)

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

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;
```



Combinational Logic Synthesis for Beginners

Simple rules for beginners

For combinational logic,

use only concurrent statements

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

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 (⇐)

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 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).

Left vs. right side of the assignment

Left side <= **Right side**

<= when-else

with-select <=

- 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