ECE 448
Lecture 4

Combinational-Circuit Building Blocks

Part 1

Reading

Required

- P. Chu, *FPGA Prototyping by VHDL Examples*

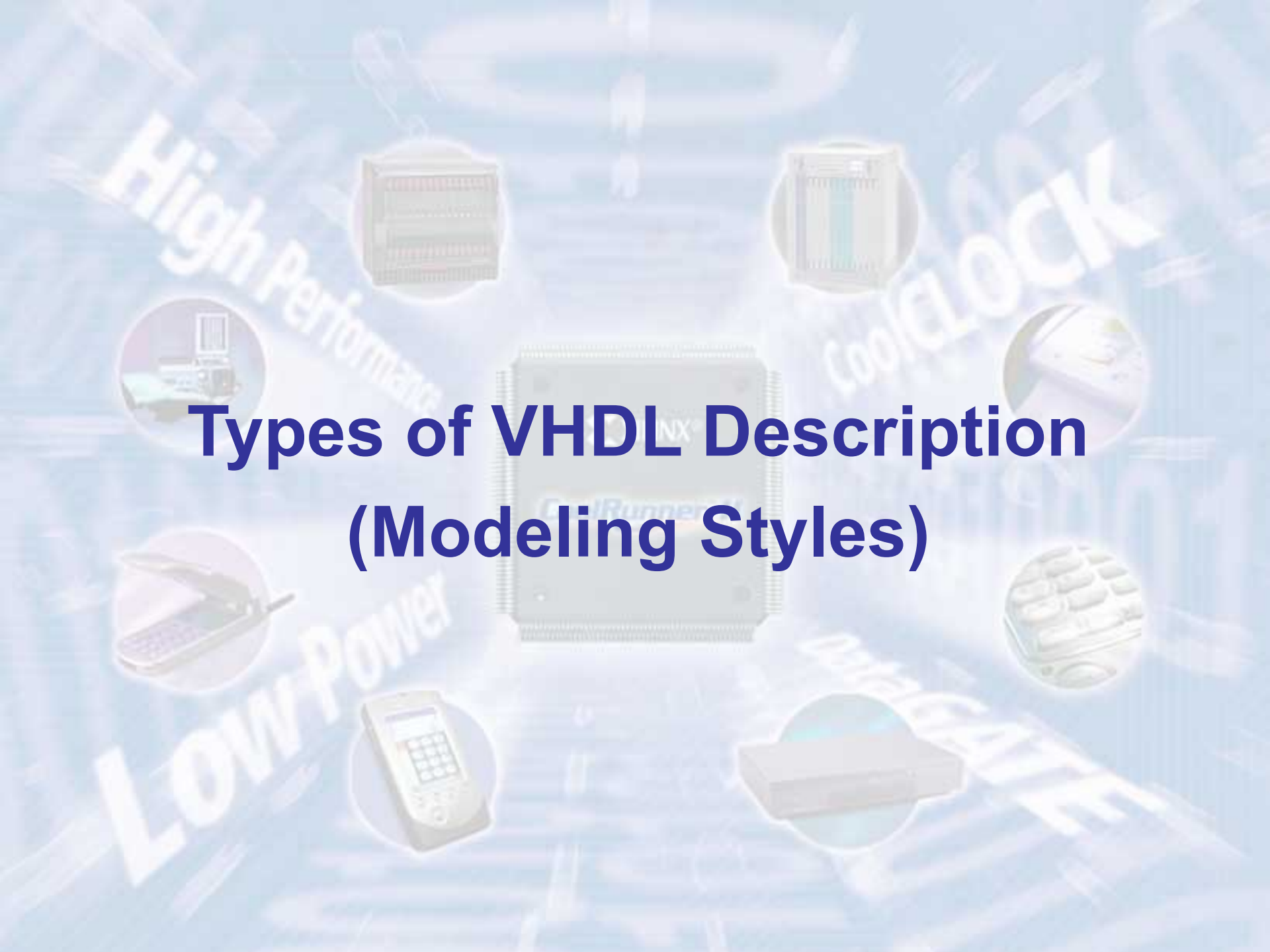
  Chapter 3, RT-level combinational circuit

Recommended

- S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*

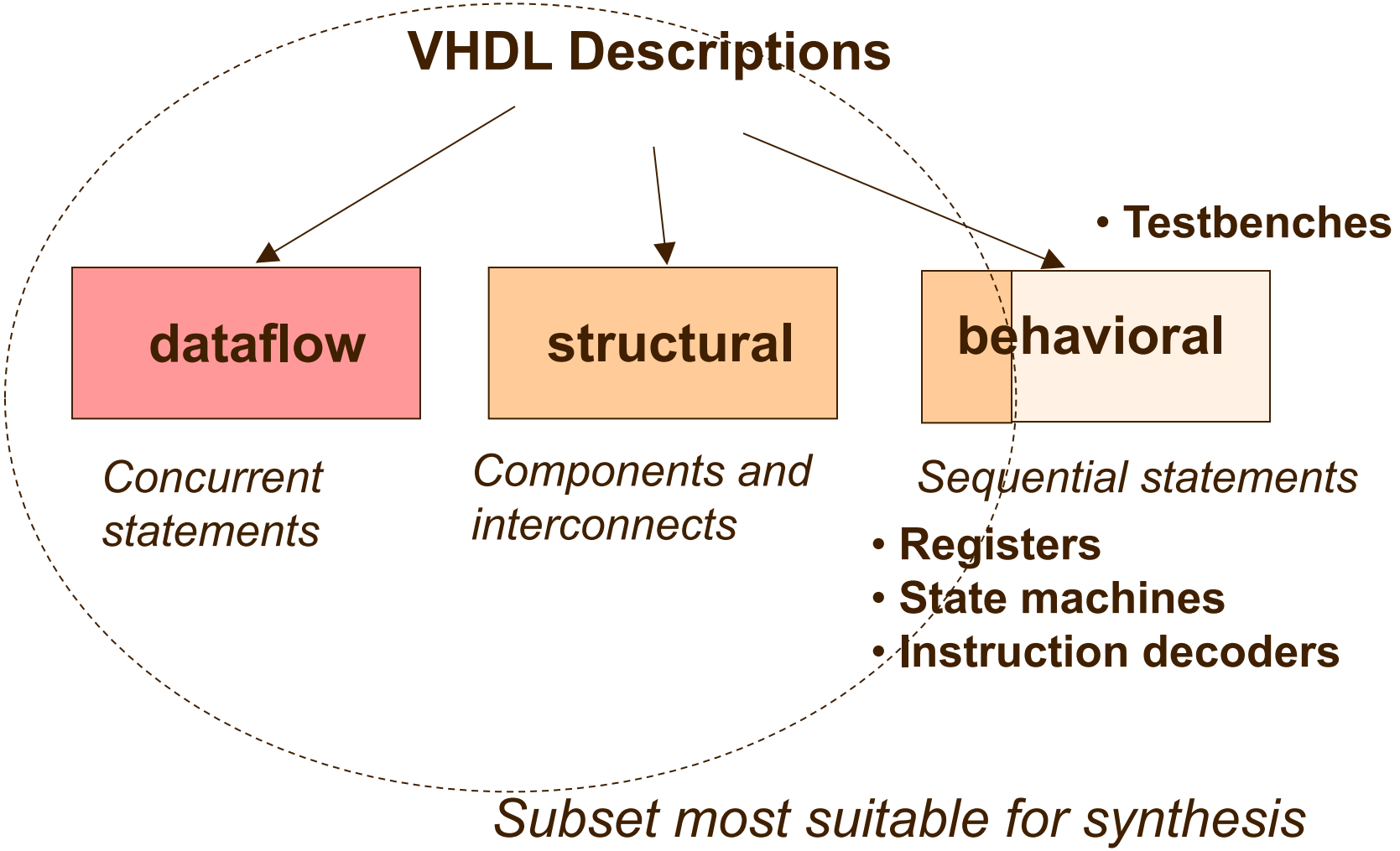
  Chapter 6, Combinational-Circuit Building Blocks

  Chapter 5.5, Design of Arithmetic Circuits Using CAD Tools

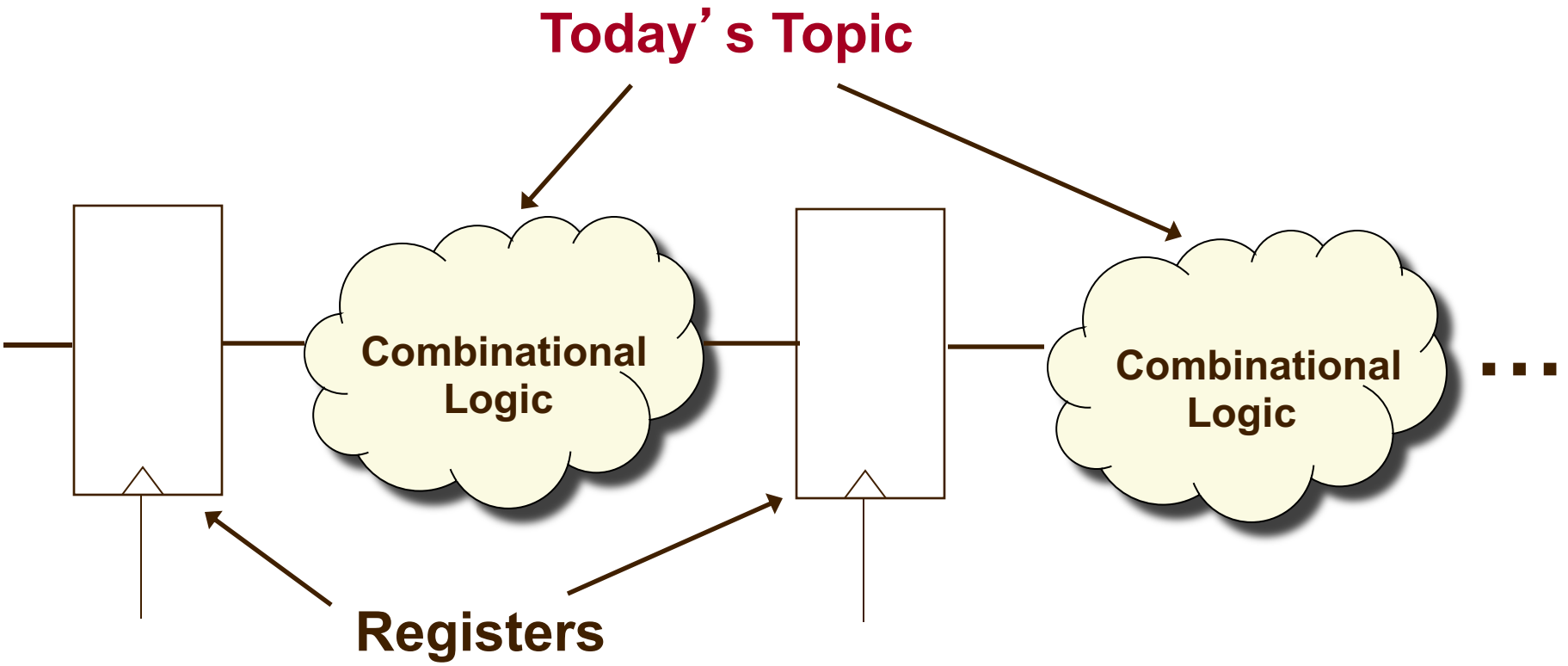


Types of VHDL Description (Modeling Styles)

Types of VHDL Description

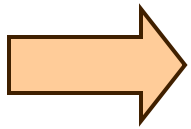


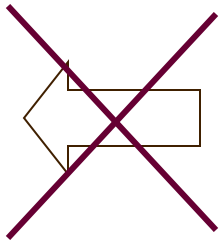
Register Transfer Level (RTL) Design Description



- Use of medium scale-components (adders, multipliers, MUXes, ROMs)
- The designer needs to specify what happens in the circuit in every clock cycle

Synthesizable VHDL

Dataflow VHDL  VHDL code synthesizable

Dataflow VHDL  VHDL code synthesizable

Data-Flow VHDL

Concurrent Statements

- **simple concurrent signal assignment**
  (⇐)
- **conditional concurrent signal assignment**
  (when-else)
- **selected concurrent signal assignment**
  (with-select-when)

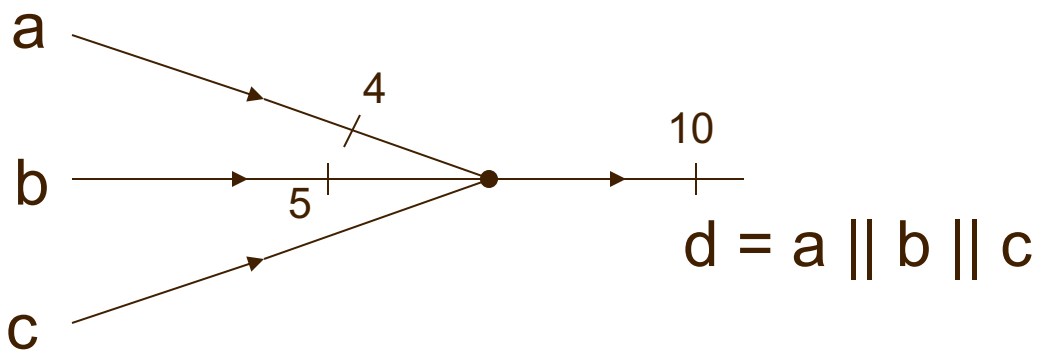
Concurrent signal assignment

<=

```
target_signal <= expression;
```

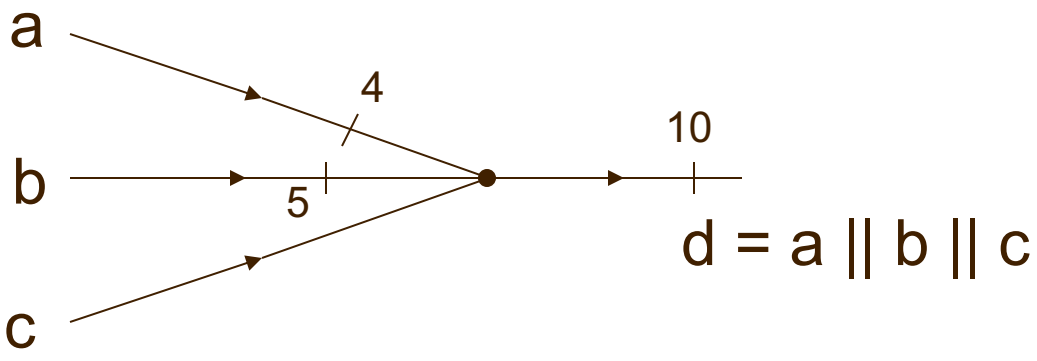
Merging and Splitting Buses & Wires

Merging wires and buses



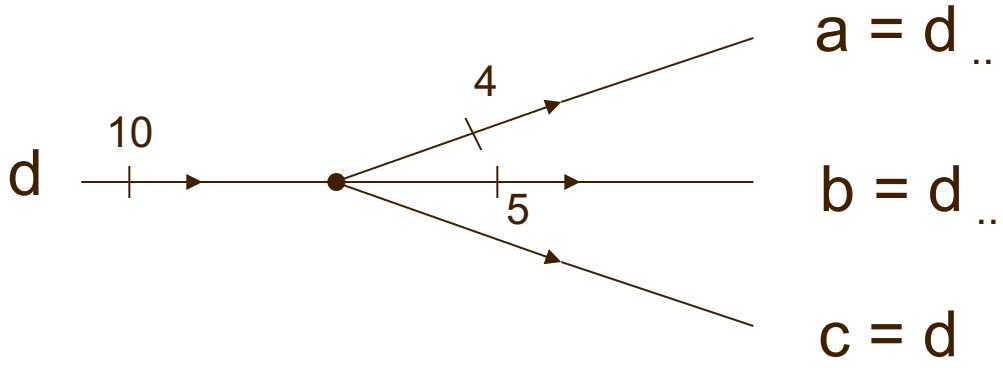

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

d <=
```

Merging wires and buses



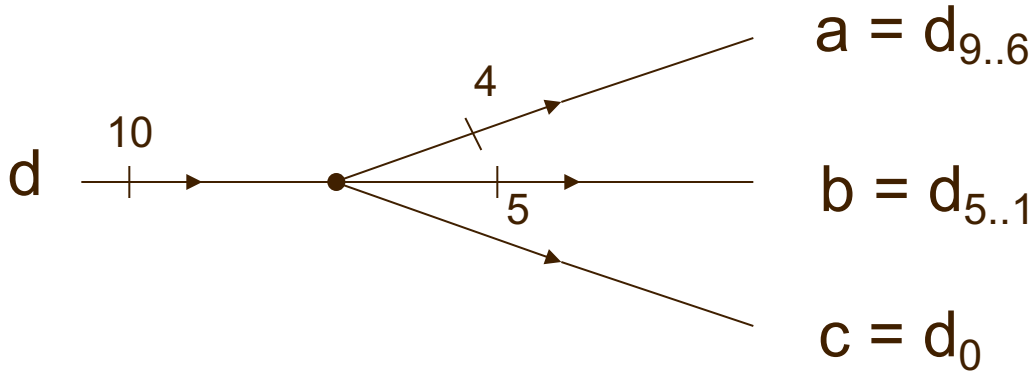

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

d <= a & b & c;
```

Splitting buses

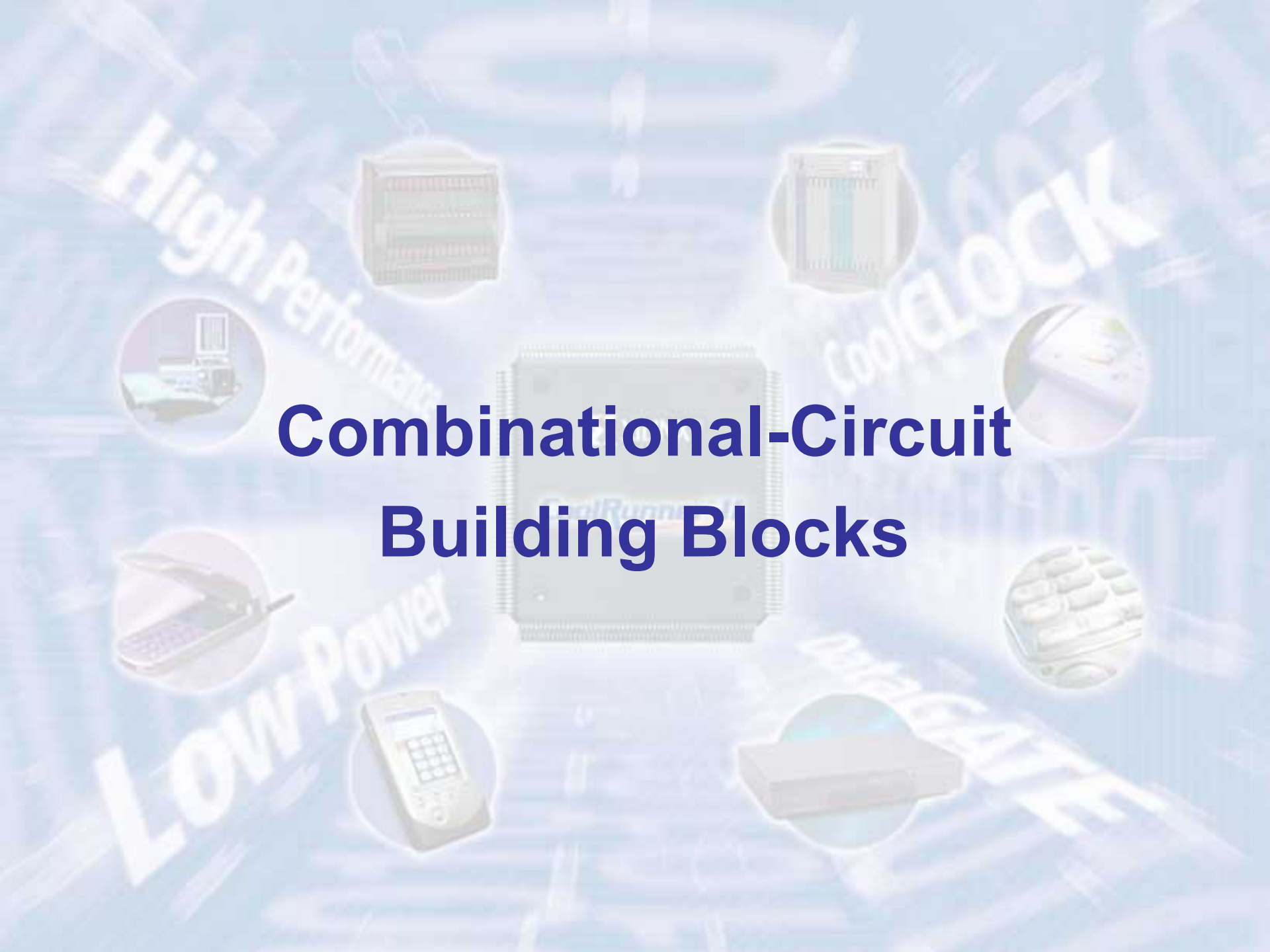



```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

a <=
b <=
c <=
```

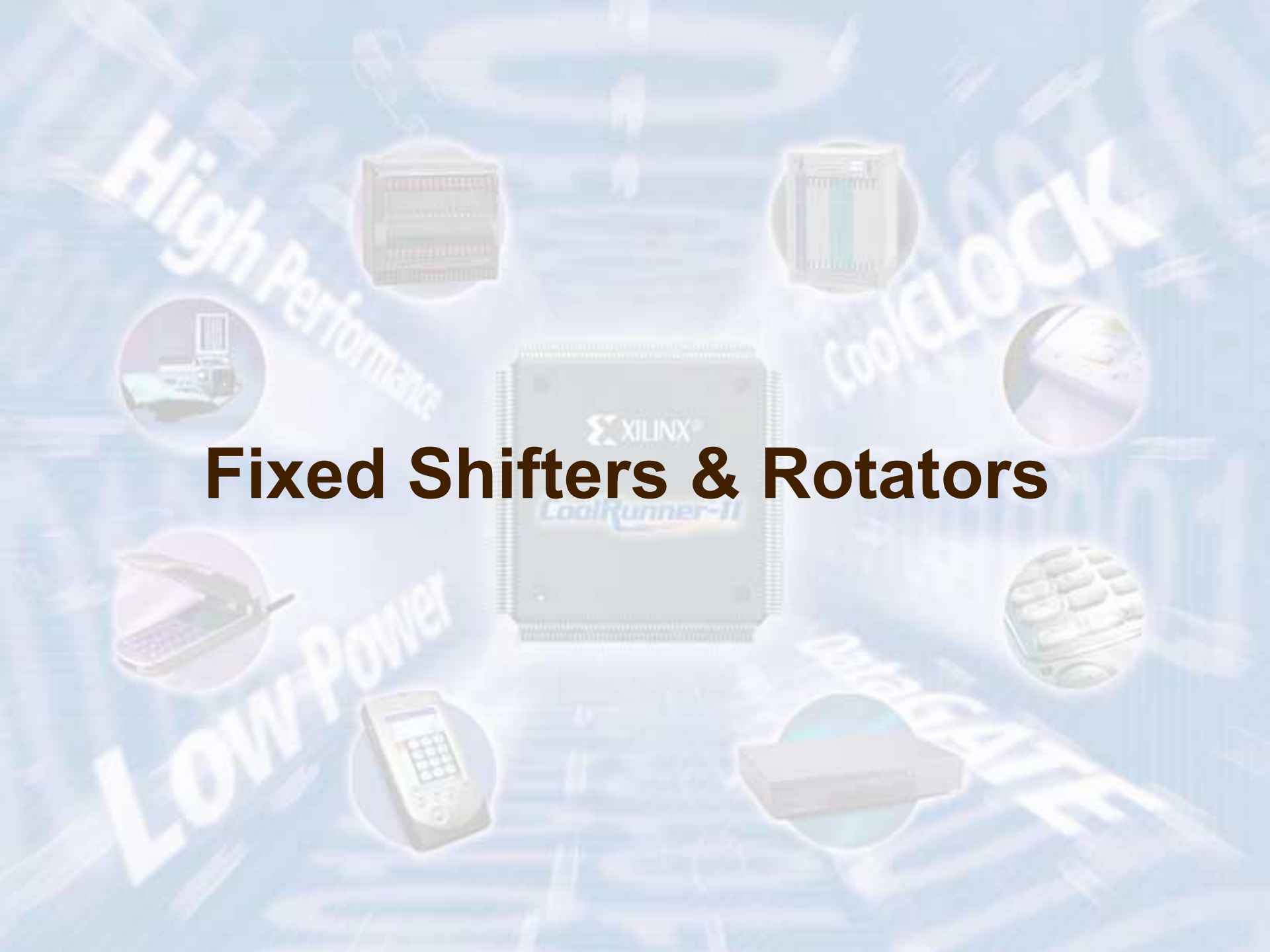
Splitting buses




```
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);
```



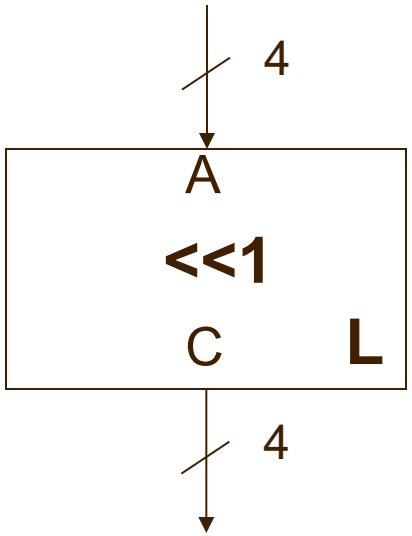
Combinational-Circuit Building Blocks

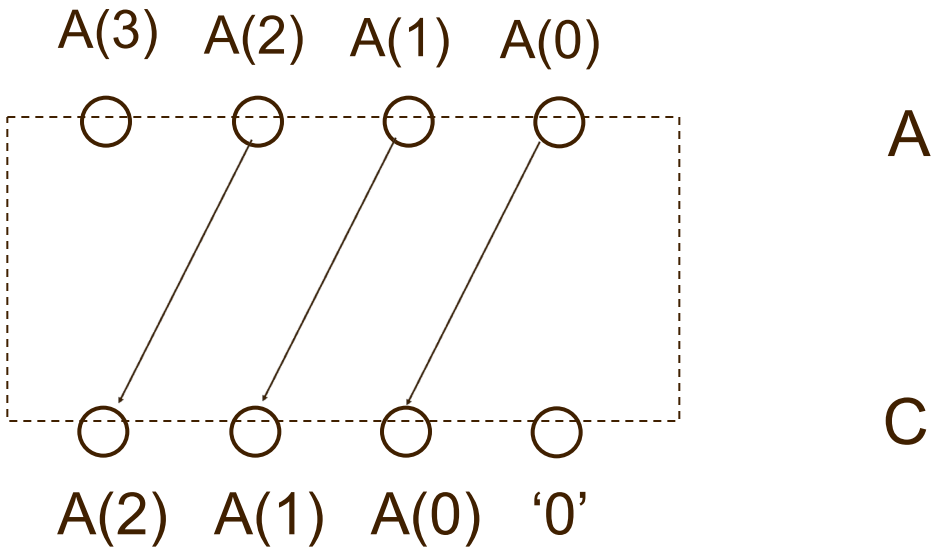


Fixed Shifters & Rotators

Fixed Logical Shift Left in VHDL

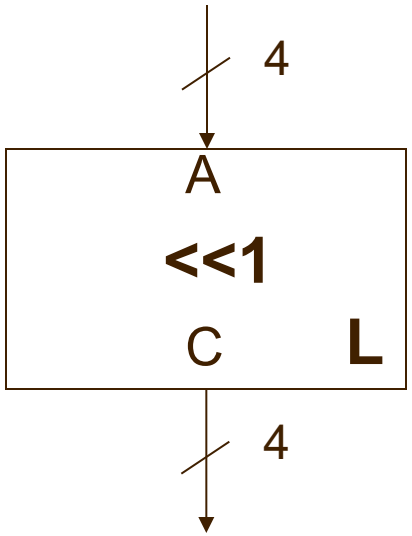
```
SIGNAL  A :     STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL  C:      STD_LOGIC_VECTOR(3 DOWNTO 0);
```

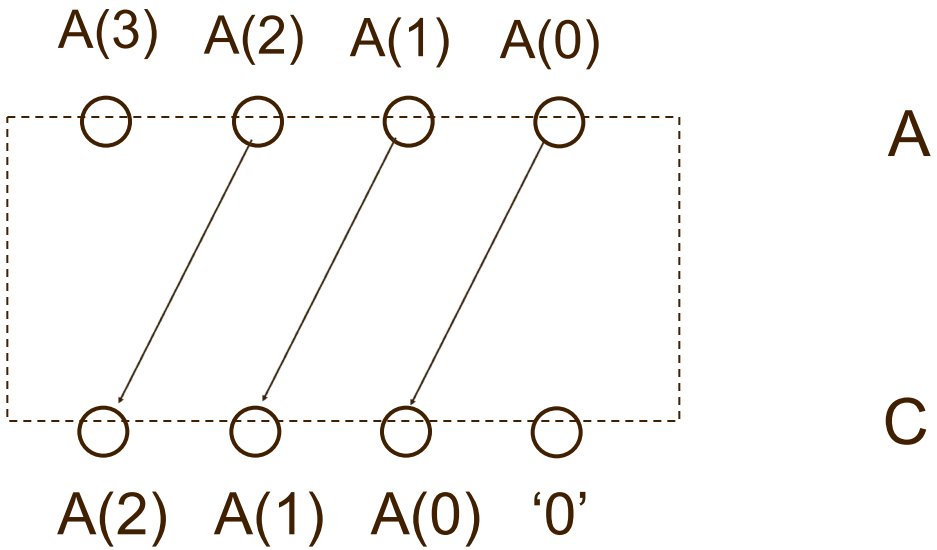




Fixed Logical Shift Left in VHDL

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





C <= A(2 downto 0) & '0';

Logical vs. Arithmetic Shift Left: Example

Unsigned

```
16  8 4 2 1
   0011 =  3
0  0110 =  6 (C=0)
0  1100 = 12 (C=0)
1  1000 =  8 (C=1)
```

⇧ C (carry) ⇧ product

Carry indicates going out of range (product invalid)

Signed (2's complement)

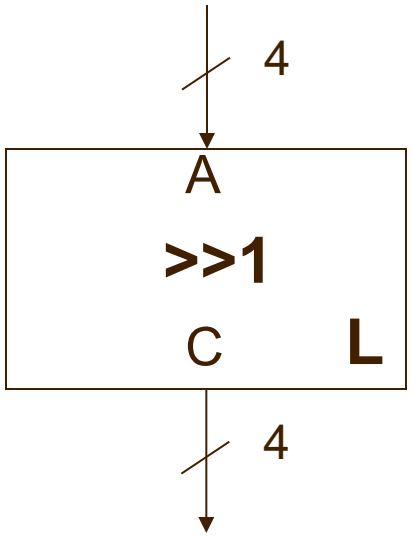
```
-8 4 2 1
 1101 = -3 (V=0)
 1010 = -6 (V=0)
 0100 =  4 (V=1)
```

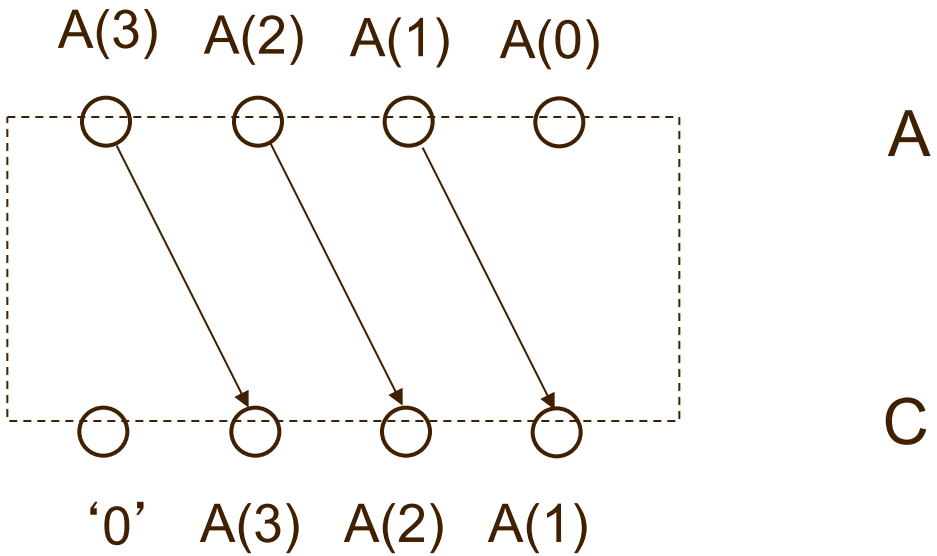
⇧

Change of sign indicates going out of range (product invalid)

Fixed Logical Shift Right in VHDL

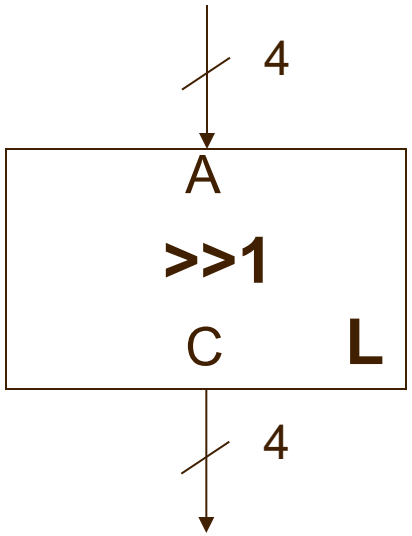
```
SIGNAL  A :     STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL  C:      STD_LOGIC_VECTOR(3 DOWNTO 0);
```

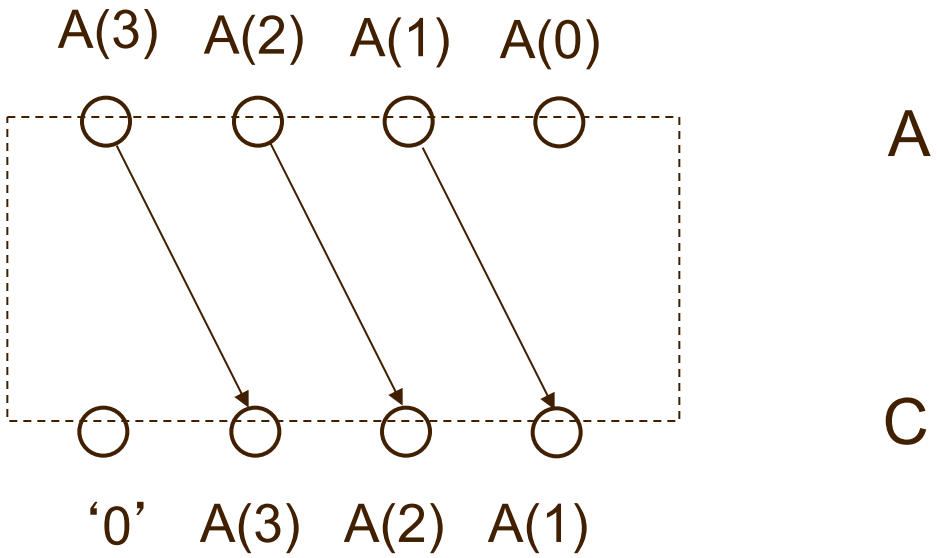




Fixed Logical Shift Right in VHDL

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

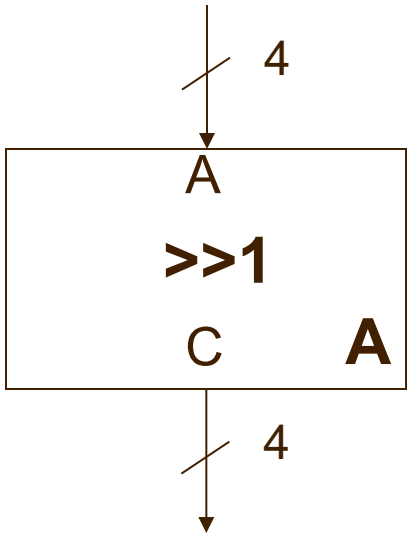


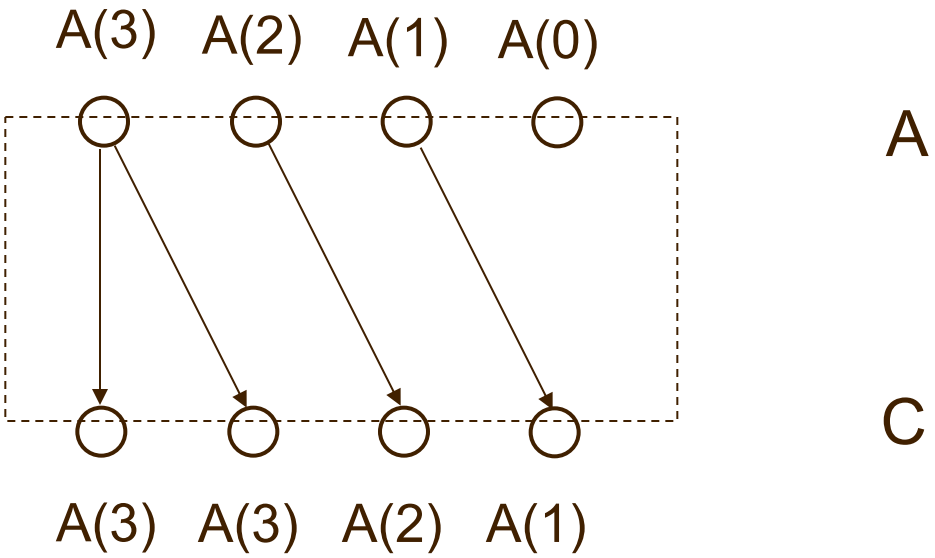


C <= '0' & A(3 downto 1);

Fixed Arithmetic Shift Right in VHDL

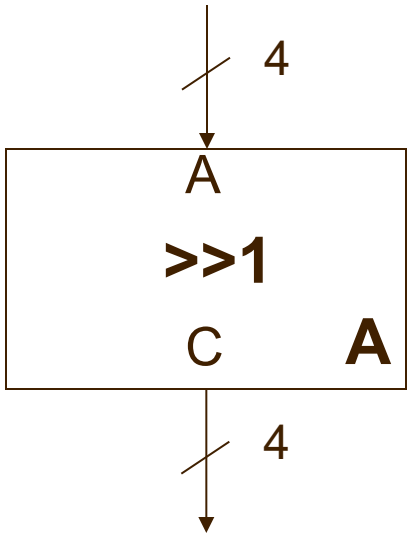
```
SIGNAL  A :     STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL  C:      STD_LOGIC_VECTOR(3 DOWNTO 0);
```

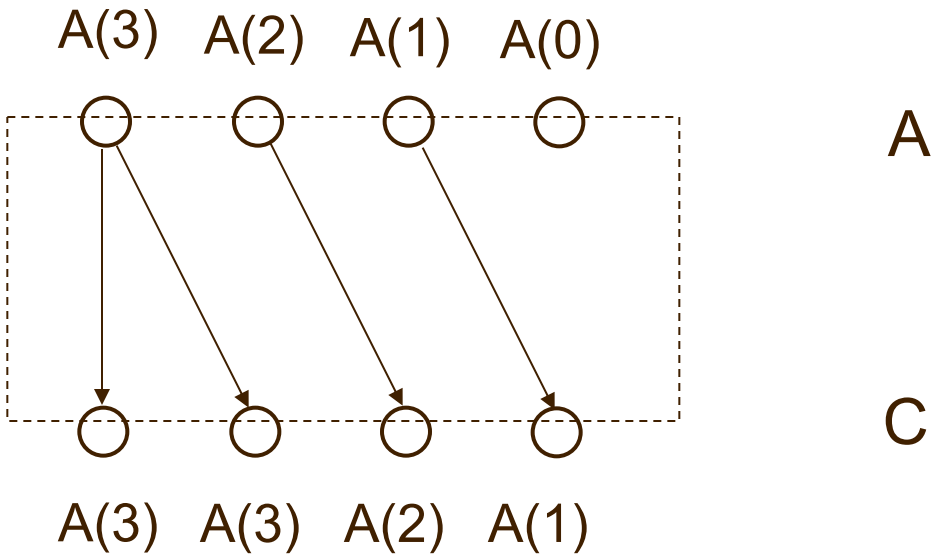




Fixed Arithmetic Shift Right in VHDL

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





C <= A(3) & A(3 downto 1);

Logical vs. Arithmetic Shift Right: Example

Unsigned

```
 8 4 2 1
1100 = 12
0110 0=6 r.0
0011 0=3 r.0
0001 1=1 r.1
0000 1=0 r.1
0000 0=0 r.0
```

⇧ quotient ⇧ remainder

Signed (2's complement)

```
-8 4 2 1
1100 = -4
1110 0=-2 r.0
1111 0=-1 r.0
1111 1=-1 r.1
1111 1=-1 r.1
```

⇧ quotient ⇧ remainder

Basic Formula for Integer Division

$$a/b = q \text{ remainder } r$$

Denoted by: $a/b = q \text{ r. } r$

$\Updownarrow$

$$a = b \cdot q + r$$

q: quotient

r: remainder

$0 \le r \le b-1$

Special case:

$a/2 = q \text{ r. } r$

$\Updownarrow$

$$a = 2 \cdot q + r$$

q: quotient

r: remainder

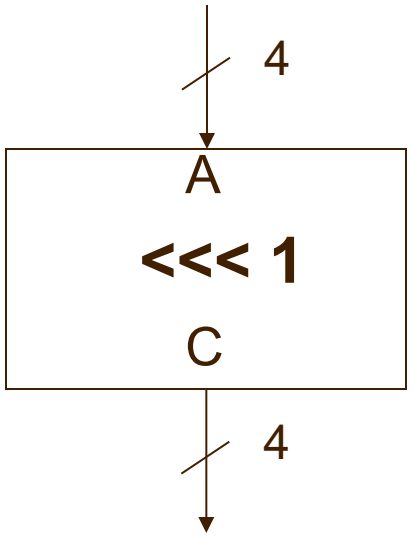
r = 0 or 1

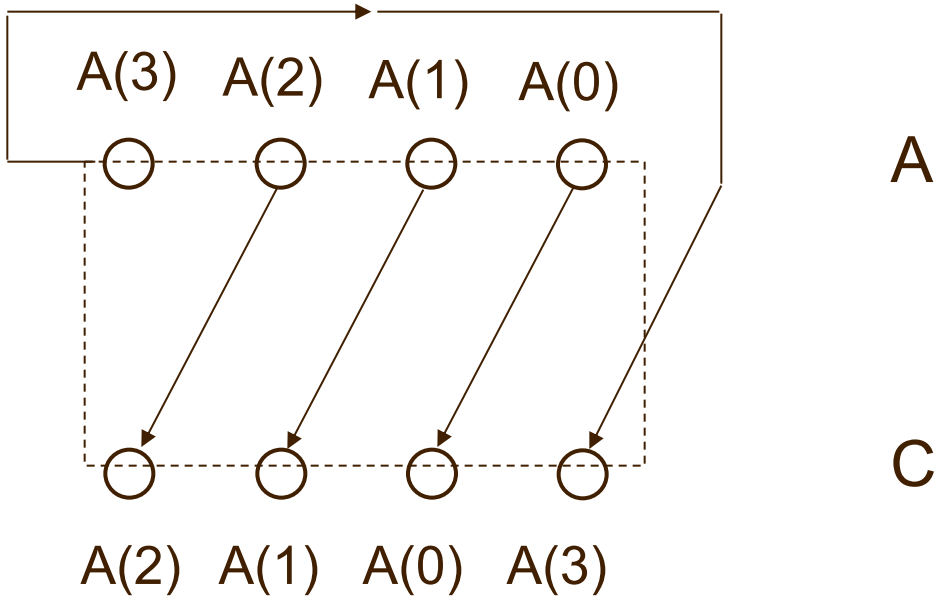
Logical vs. Arithmetic Shift Right: Example

Unsigned	Signed (2's complement)
8 4 2 1	-8 4 2 1
1100 = 12	1100 = -4
12/2=6 r.0 ⇔ 12=2·6+0	-4/2=-2 r.0 ⇔ -4=2·(-2)+0
6/2=3 r.0 ⇔ 6=2·3+0	-2/2=-1 r.0 ⇔ -2=2·(-1)+0
3/2=1 r.1 ⇔ 3=2·1+1	-1/2=-1 r.1 ⇔ -1=2·(-1)+1
1/2=0 r.1 ⇔ 1=2·0+1	-1/2=-1 r.1 ⇔ -1=2·(-1)+1
0/2=0 r.0 ⇔ 0=2·0+0	

Fixed Rotation Left in VHDL

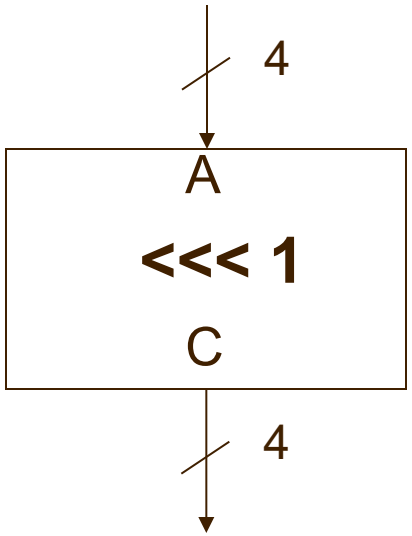
```
SIGNAL  A :     STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL  C:      STD_LOGIC_VECTOR(3 DOWNTO 0);
```

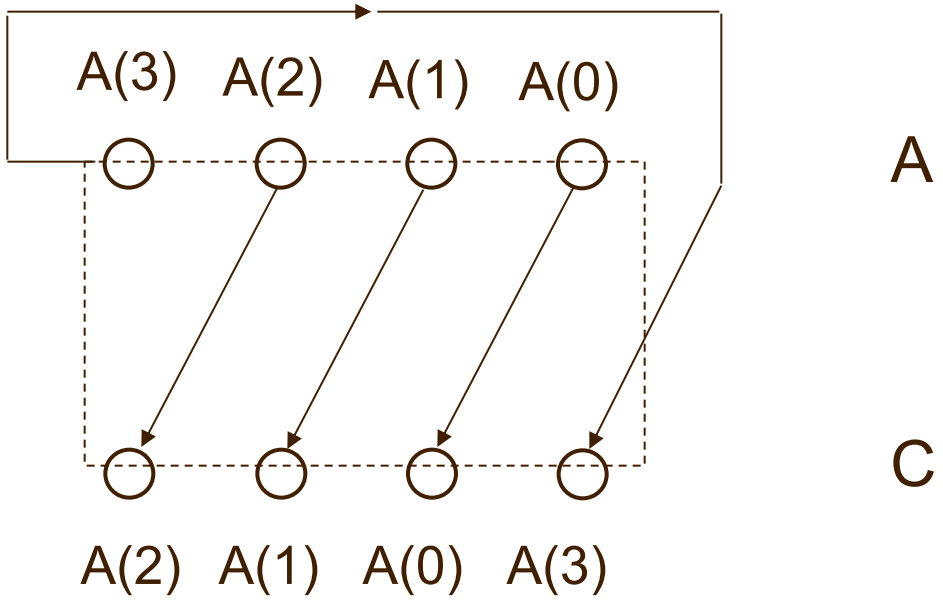




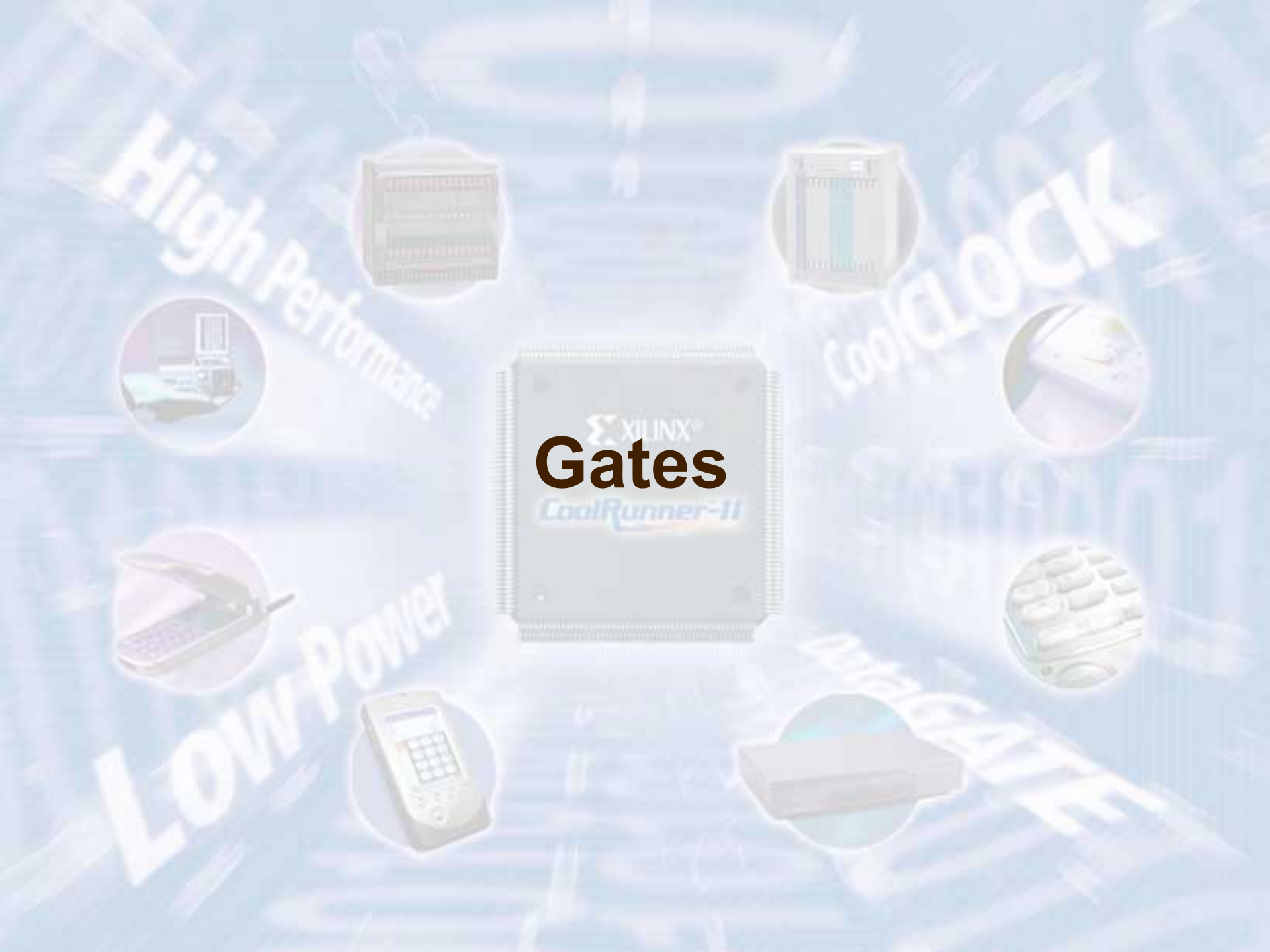
Fixed Rotation Left in VHDL

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



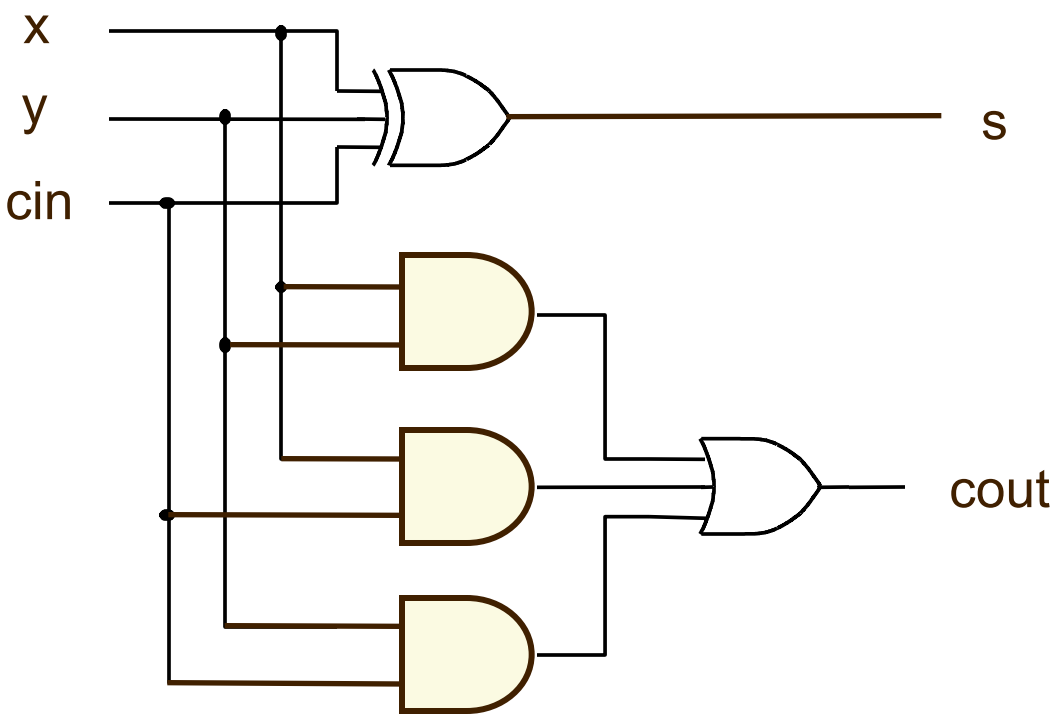


C <= A(2 downto 0) & A(3);



Gates

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) OR (cin AND y) ;
END dataflow ;
```

Logic Operators

- Logic operators

- Logic operators precedence

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

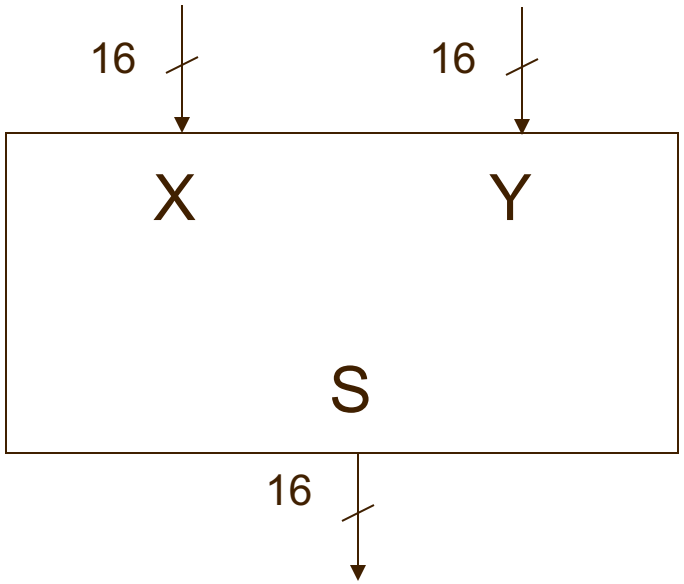
y = (ab + c)d

Correct

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

Adders

Adder w/o carry in or carry out

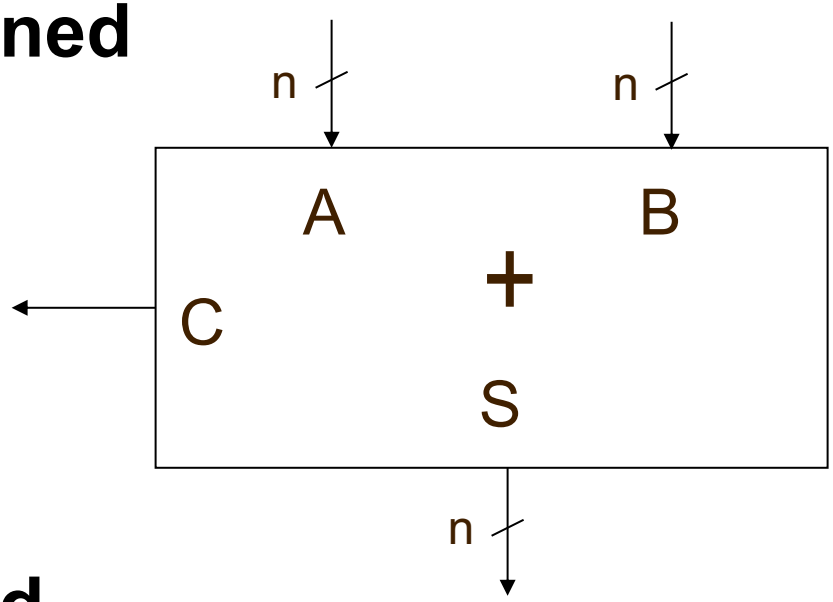


Adder w/o carry in or carry out

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.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 <= std_logic_vector(unsigned(X) + unsigned(Y));
END dataflow ;
```

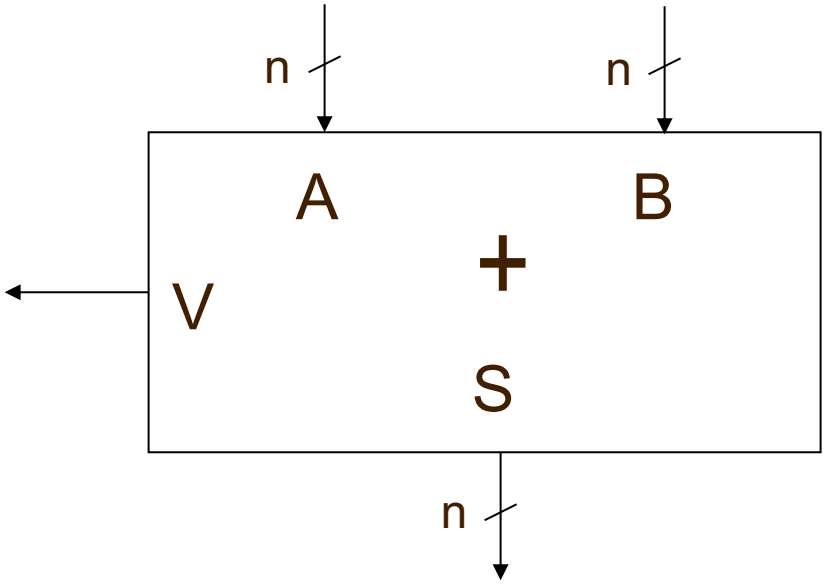
Unsigned Adder vs. Signed Adder

Unsigned



C (carry out):
going out of range
for unsigned numbers

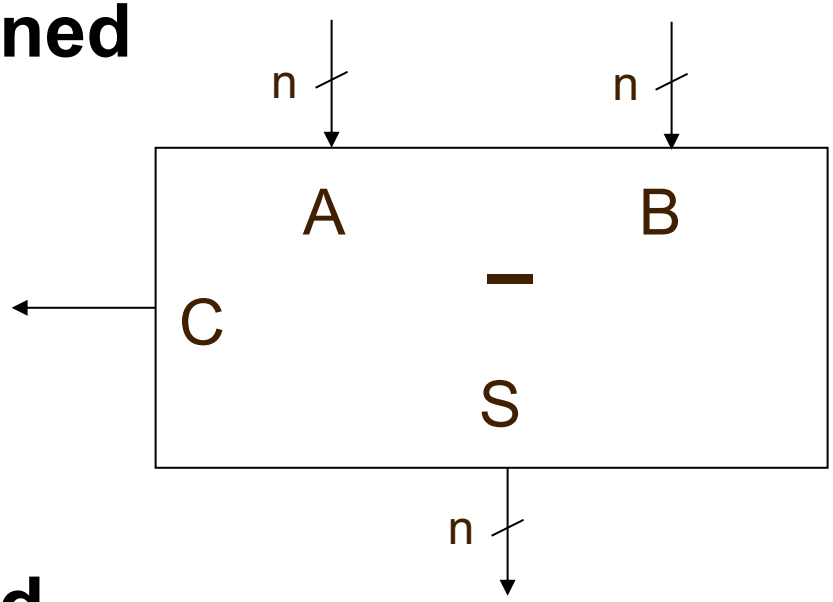
Signed



V (overflow):
going out of range
for signed numbers

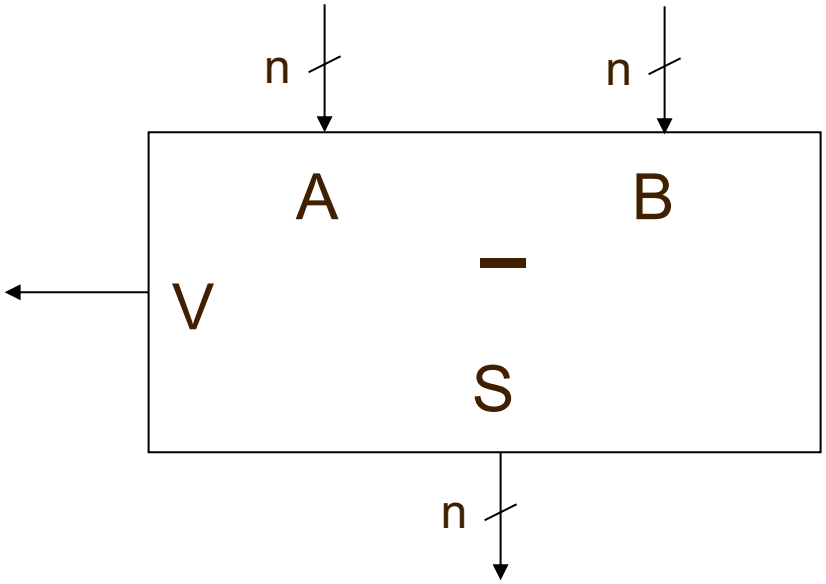
Unsigned Subtractor vs. Signed Subtractor

Unsigned



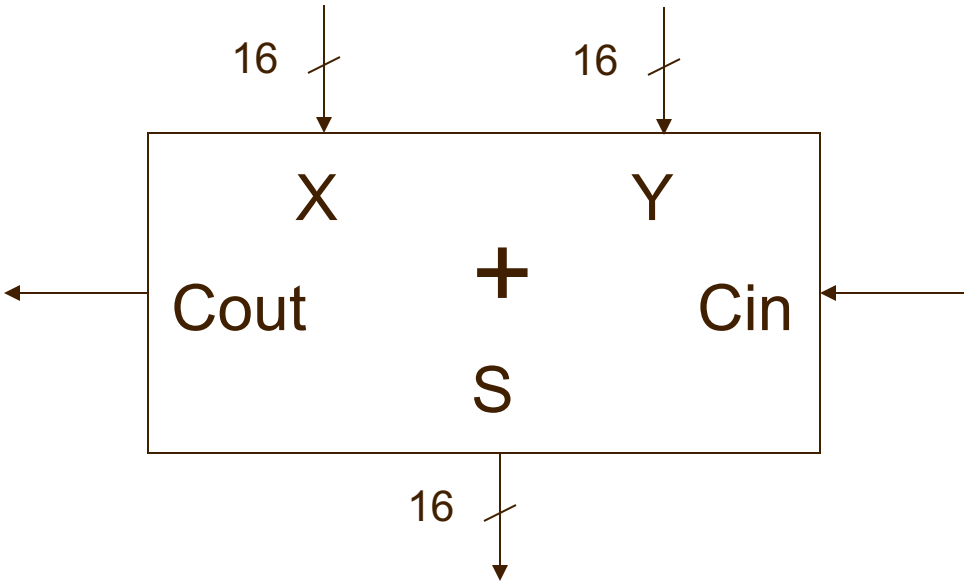
C (carry out):
going out of range
for unsigned numbers

Signed



V (overflow):
going out of range
for signed numbers

16-bit Unsigned Adder



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 (3)

```
ARCHITECTURE dataflow OF adder16 IS
        signal USum: unsigned(16 DOWNTO 0) ;
        signal UCin: unsigned(0 downto 0);
BEGIN
        UCin(0) <= Cin;
        USum <= unsigned('0' & X) + unsigned(Y) + UCin;
        S <= std_logic_vector(USum(15 downto 0));
        Cout <= USum(16) ;
END dataflow ;
```

Overloaded operators in IEEE numeric_std package

overloaded operator	description	data type of operand a	data type of operand b	data type of result
abs a - a	absolute value negation	signed		signed
a * b a / b a **mod** b a **rem** b a + b a - b	arithmetic operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	unsigned unsigned signed signed
a = b a /= b a < b a <= b a > b a >= b	relational operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	boolean boolean boolean boolean

Multipliers

Unsigned vs. Signed Multiplication

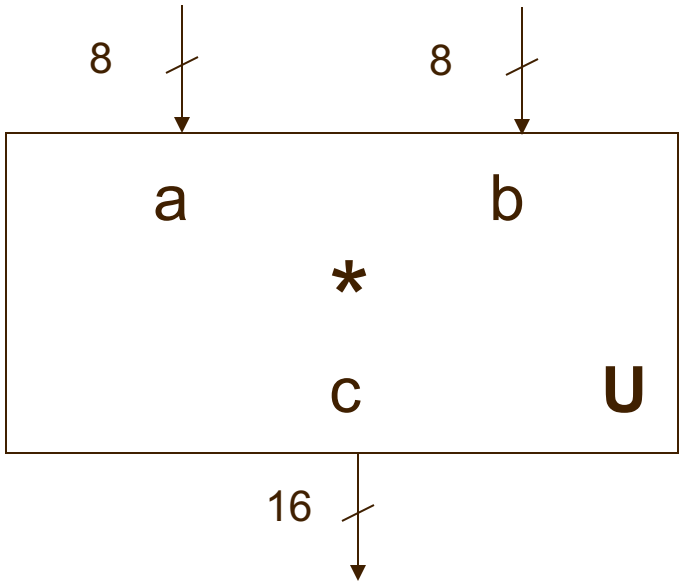
Unsigned

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

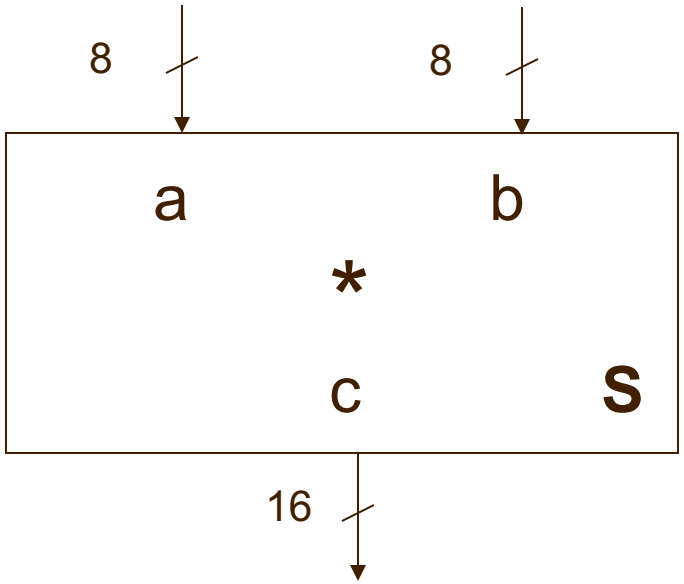
Signed

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

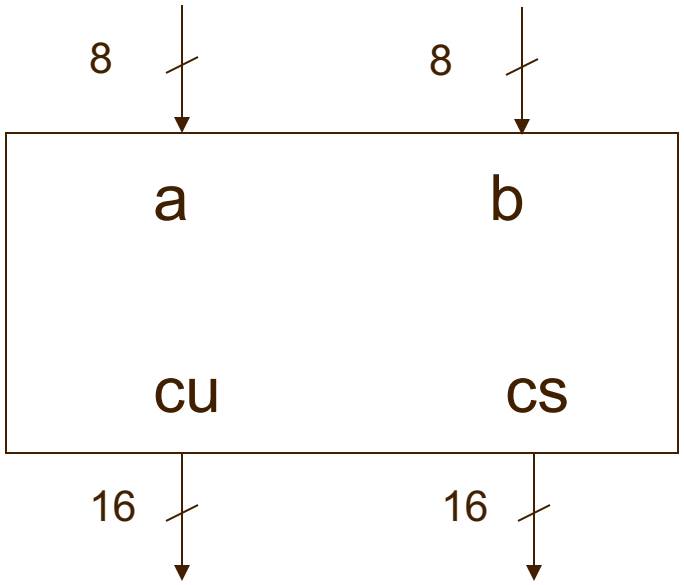
8x8-bit Unsigned Multiplier



8x8-bit Signed Multiplier

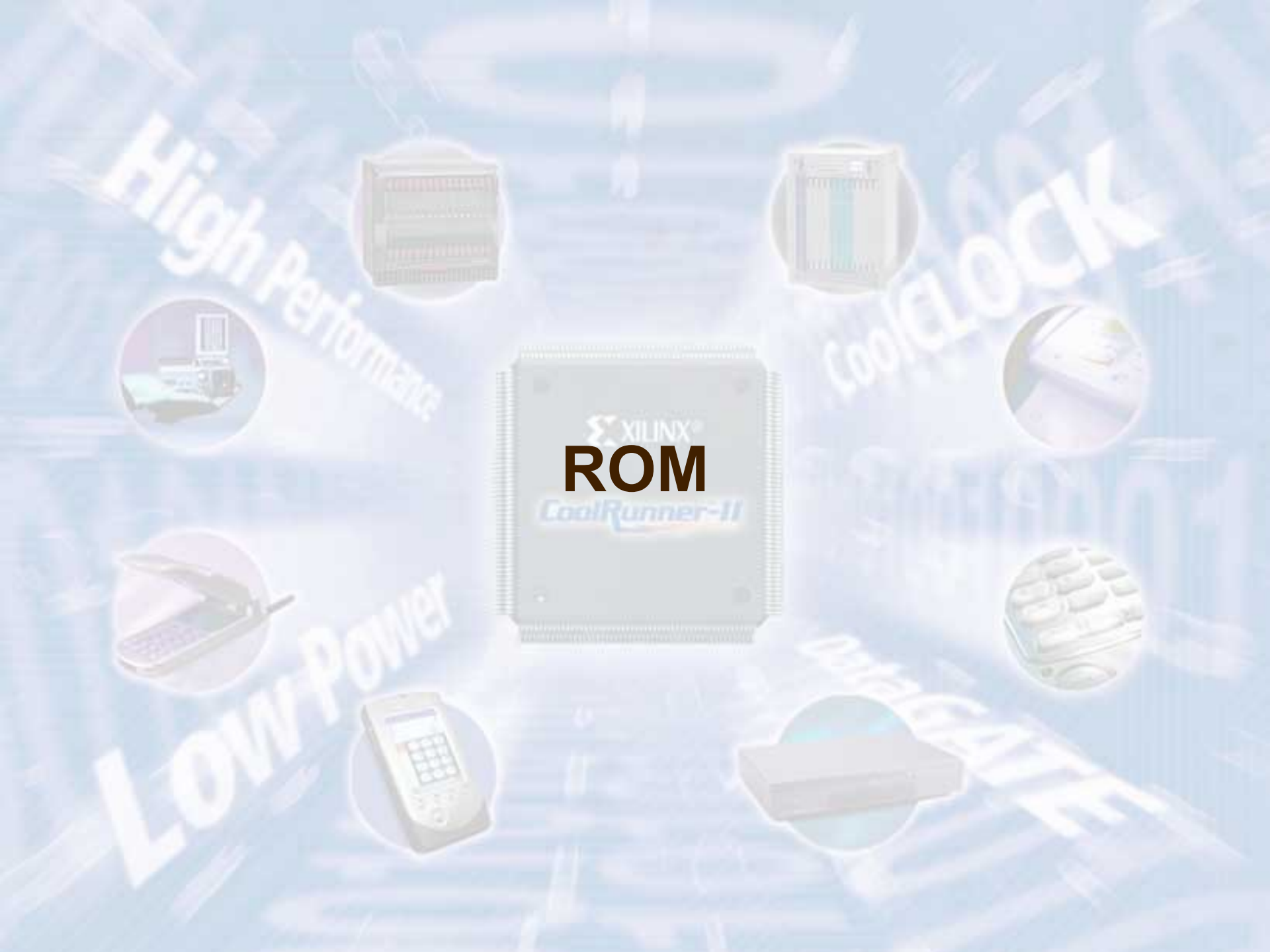


8x8-bit Unsigned and Signed Multiplier



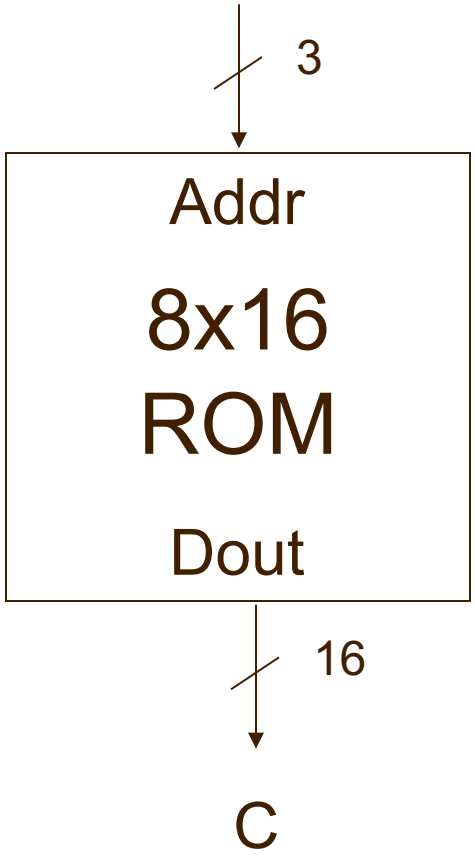
Multiplication of signed and unsigned numbers

```
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;
```



ROM

ROM 8x16 example (1)

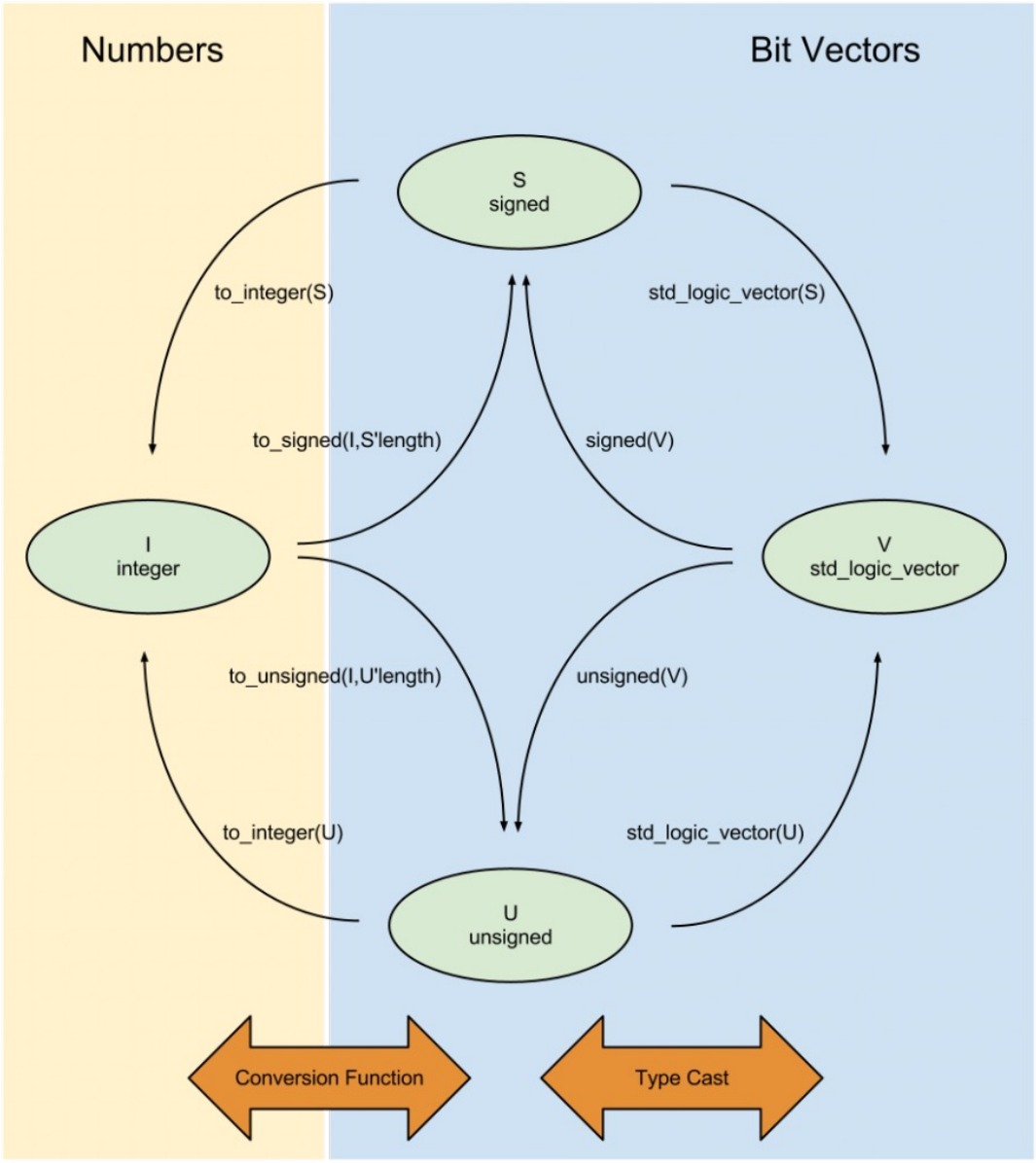


ROM 8x16 example (2)

```
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;
```

ROM 8x16 example (3)

```
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;
```



Source: VHDL Type Conversion
by Shannon Hilbert
available at http://www.bitweenie.com/listings/vhdl-type-conversion

ROM 8x16 example (4)

```
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;
```

Overloaded operators in IEEE numeric_std package

overloaded operator	description	data type of operand a	data type of operand b	data type of result
abs a - a	absolute value negation	signed		signed
a * b a / b a **mod** b a **rem** b a + b a - b	arithmetic operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	unsigned unsigned signed signed
a = b a /= b a < b a <= b a > b a >= b	relational operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	boolean boolean boolean boolean

- E.g.,

```
signal a, b, c, d: unsigned(7 downto 0);
. . .
a <= b + c;
d <= b + 1;
e <= (5 + a + b) - c;
```

- E.g.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
. . .
signal s1, s2, s3, s4, s5, s6: std_logic_vector(3 downto 0);
signal u1, u2, u3, u4, u6, u7: unsigned(3 downto 0);
signal sg: signed(3 downto 0);
```

– Ok

```
u3 <= u2 + u1;  --- ok, both operands unsigned
u4 <= u2 + 1;   --- ok, operands unsigned and natural
```

– Wrong

```
u5 <= sg;  -- type mismatch
u6 <= 5;   -- type mismatch
```

– Fix

```
u5 <= unsigned(sg);     -- type casting
u6 <= to_unsigned(5,4); -- conversion function
```

– Wrong

```
u7 <= sg + u1;   -- + undefined over the types
```

– Fix

```
u7 <= unsigned(sg) + u1; -- ok, but be careful
```

– Wrong

```
s3 <= u3;  -- type mismatch
s4 <= 5;   -- type mismatch
```

– Fix

```
s3 <= std_logic_vector(u3); -- type casting
s4 <= std_logic_vector(to_unsigned(5,4));
```

– Wrong

s5 <= s2 + s1; + undefined over std_logic_vector

s6 <= s2 + 1; + undefined

– Fix

s5 <= std_logic_vector(unsigned(s2) + unsigned(s1));

s6 <= std_logic_vector(unsigned(s2) + 1);