ECE 545
Lecture 10

Modeling of Circuits with Regular Structure

Required reading

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

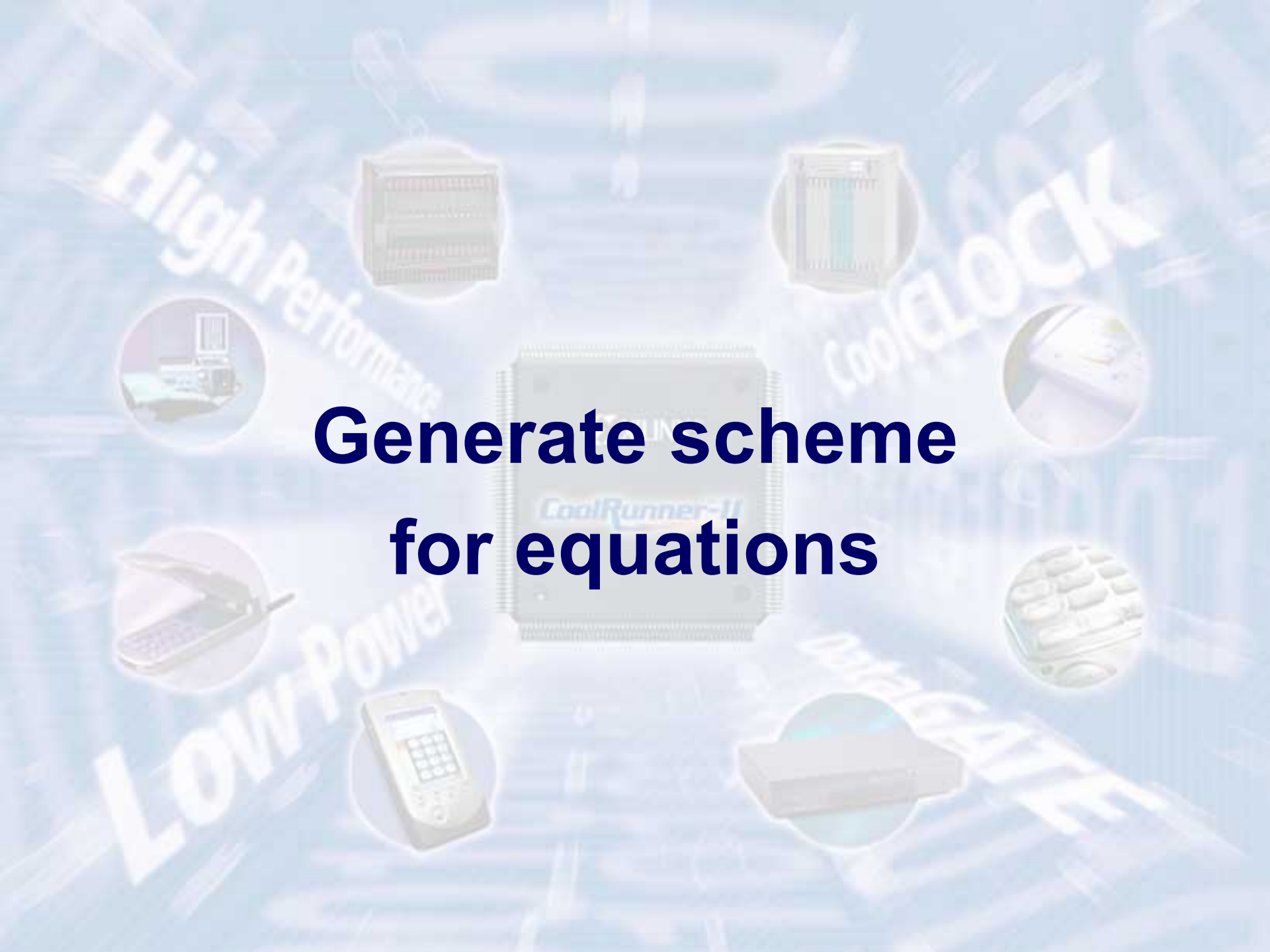
Chapter 14, Parameterized Design: Principle

Chapter 15, (except 15.4) Parameterized Design: Practice

Recommended reading

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

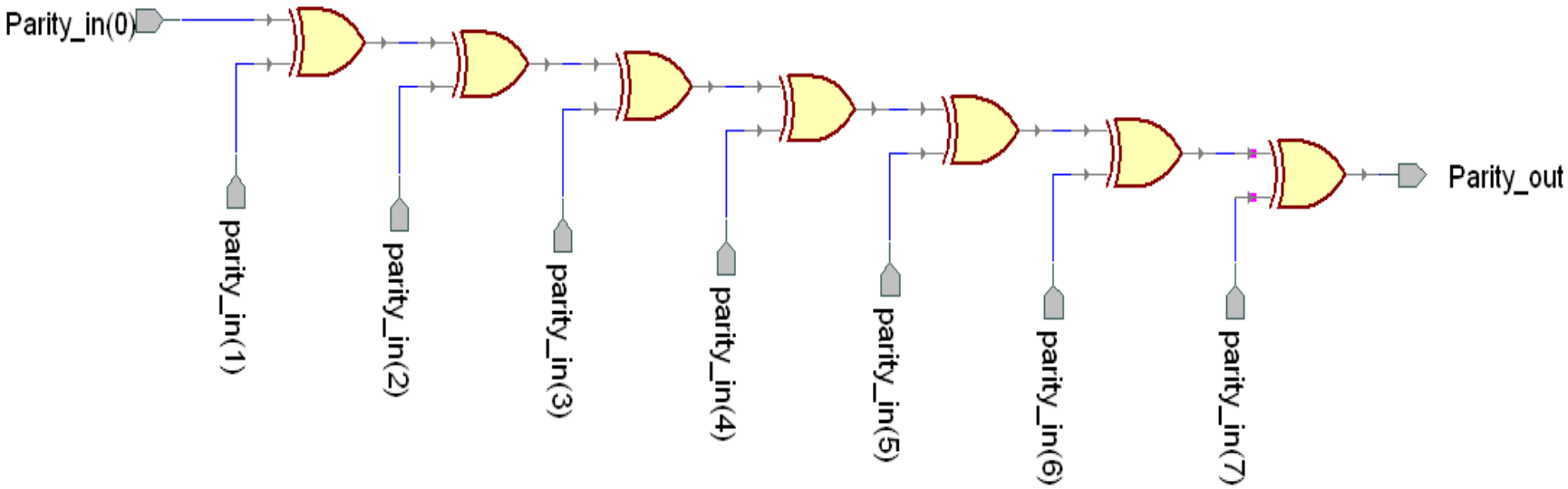
Chapter 15.4, More sophisticated examples



Generate scheme for equations

PARITY Example

Example 1: Block Diagram

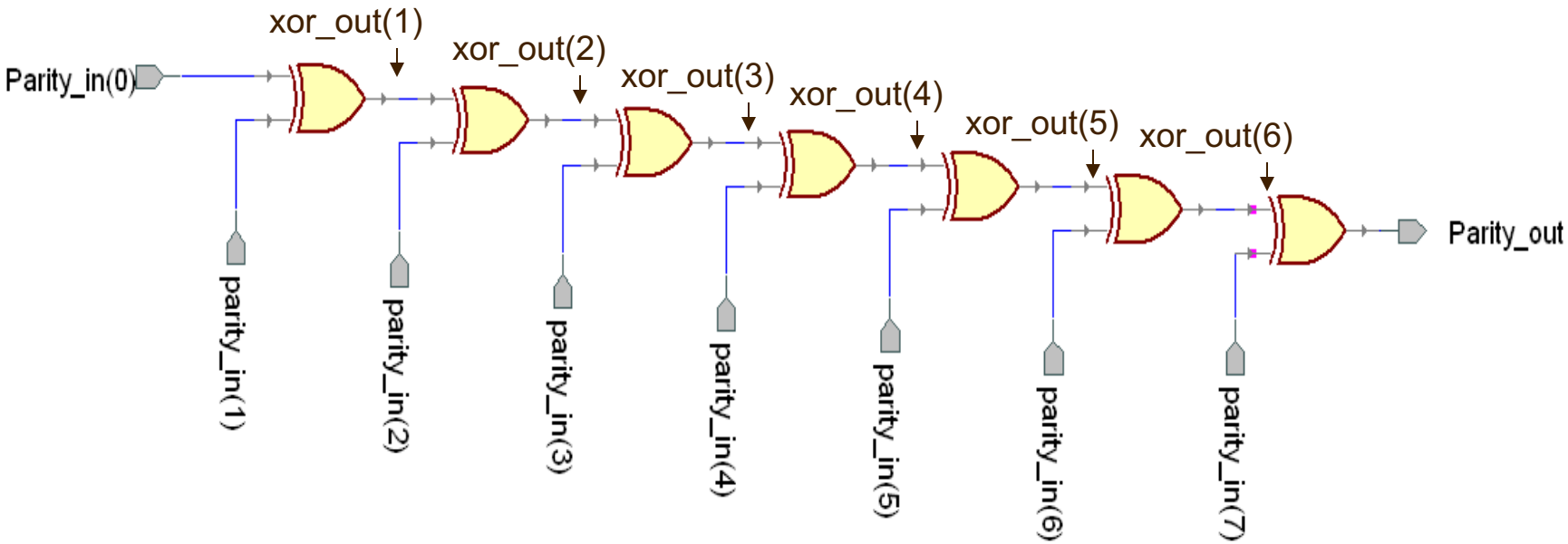


PARITY: Entity Declaration

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

ENTITY parity IS
     PORT(
          parity_in    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          parity_out   : OUT STD_LOGIC
        );
END parity;
```

PARITY: Block Diagram



PARITY: Architecture

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: std_logic_vector (6 downto 1);

BEGIN

    xor_out(1) <= parity_in(0) XOR parity_in(1);
    xor_out(2) <= xor_out(1) XOR parity_in(2);
    xor_out(3) <= xor_out(2) XOR parity_in(3);
    xor_out(4) <= xor_out(3) XOR parity_in(4);
    xor_out(5) <= xor_out(4) XOR parity_in(5);
    xor_out(6) <= xor_out(5) XOR parity_in(6);
    parity_out <= xor_out(6) XOR parity_in(7);

END parity_dataflow;
```

PARITY: Architecture (2a)

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR (6 DOWNTO 1);

BEGIN

    xor_out(1) <= parity_in(0) XOR parity_in(1);

    G1: FOR i IN 2 TO 6 GENERATE
            xor_out(i) <= xor_out(i-1) XOR parity_in(i);
        END GENERATE;

   parity_out <= xor_out(6) XOR parity_in(7);

END parity_dataflow;
```

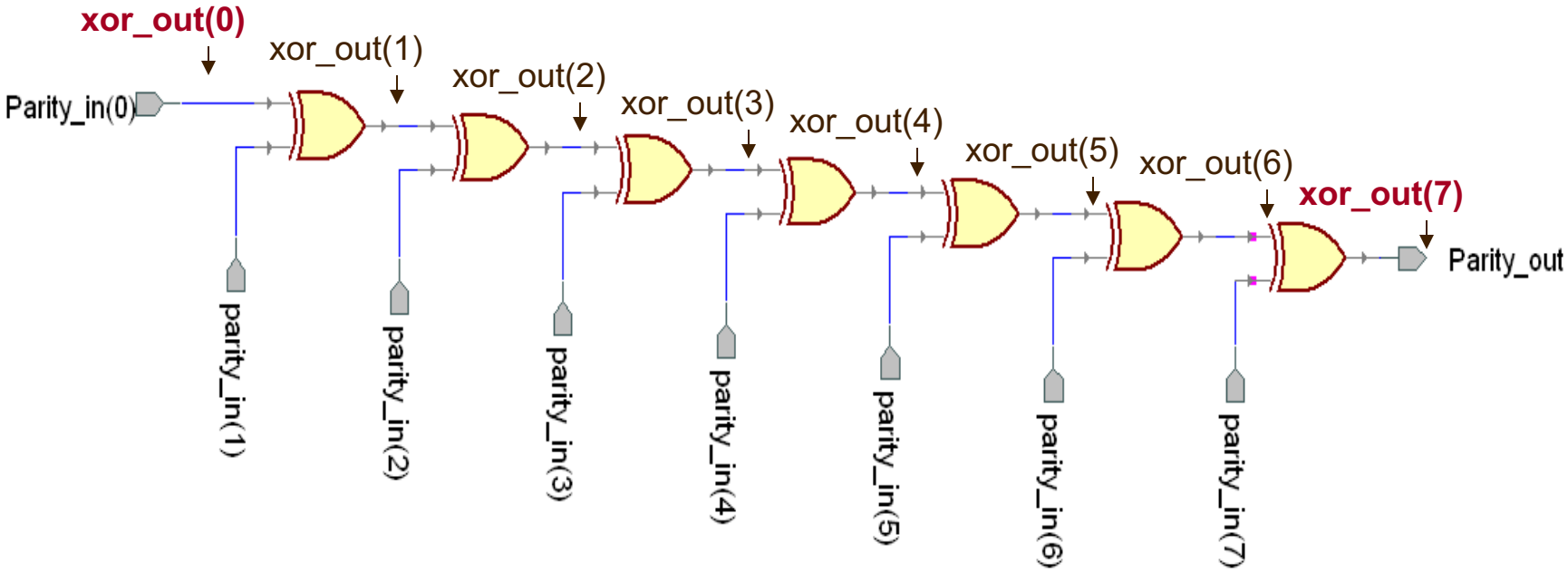
PARITY: Architecture (2b)

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR (6 DOWNTO 1);

BEGIN
    G2: FOR i IN 1 TO 7 GENERATE
          left_xor:  IF i=1 GENERATE
                xor_out(i) <= parity_in(i-1) XOR parity_in(i);
          END GENERATE;
          middle_xor: IF (i >1) AND (i<7) GENERATE
                xor_out(i) <= xor_out(i-1) XOR parity_in(i);
          END GENERATE;
         right_xor: IF i=7 GENERATE
                parity_out <= xor_out(i-1) XOR parity_in(i);
          END GENERATE;
    END GENERATE;
END parity_dataflow;
```

PARITY: Block Diagram (2)



PARITY: Architecture

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR (7 downto 0);

BEGIN

    xor_out(0) <= parity_in(0);
    xor_out(1) <= xor_out(0) XOR parity_in(1);
    xor_out(2) <= xor_out(1) XOR parity_in(2);
    xor_out(3) <= xor_out(2) XOR parity_in(3);
    xor_out(4) <= xor_out(3) XOR parity_in(4);
    xor_out(5) <= xor_out(4) XOR parity_in(5);
    xor_out(6) <= xor_out(5) XOR parity_in(6);
    xor_out(7) <= xor_out(6) XOR parity_in(7);
    parity_out <= xor_out(7);

END parity_dataflow;
```

PARITY: Architecture (2)

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN

    xor_out(0) <= parity_in(0);

    G2: FOR i IN 1 TO 7 GENERATE
          xor_out(i) <= xor_out(i-1) XOR parity_in(i);
    END GENERATE G2;

    parity_out <= xor_out(7);

END parity_dataflow;
```

For Generate Statement

For - Generate

```
label:
FOR identifier IN range GENERATE
     {Concurrent Statements}
END GENERATE;
```

Conditional Generate Statement

If - Generate

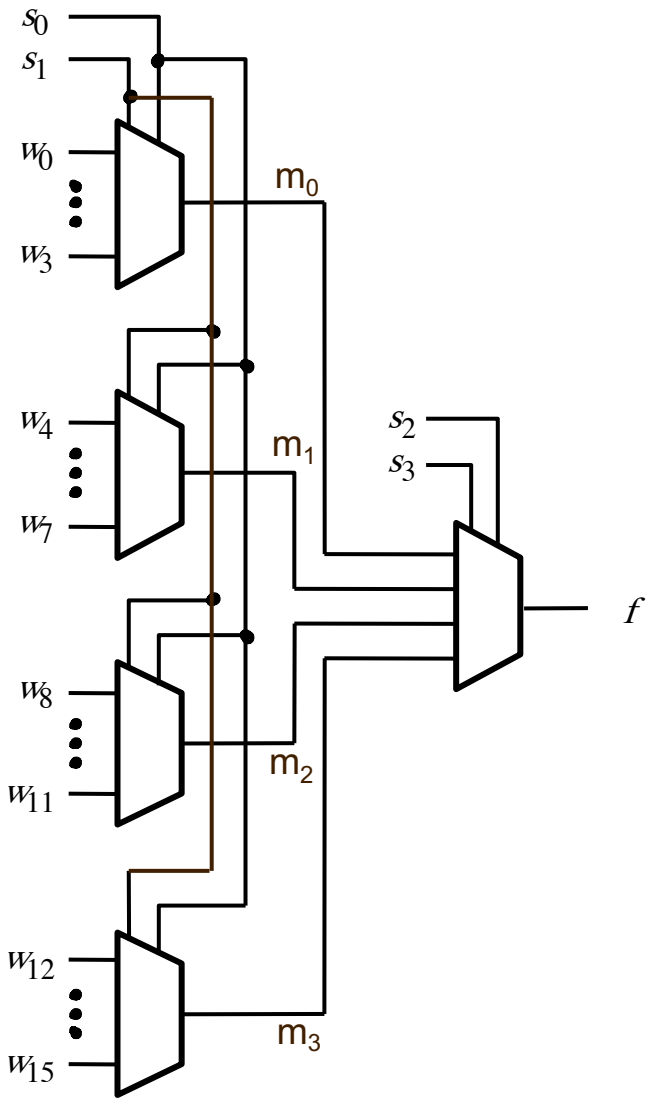
```
label:
IF boolean_expression GENERATE
     {Concurrent Statements}
END GENERATE;
```



Generate scheme for components

Example 1

Example 1



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 ;

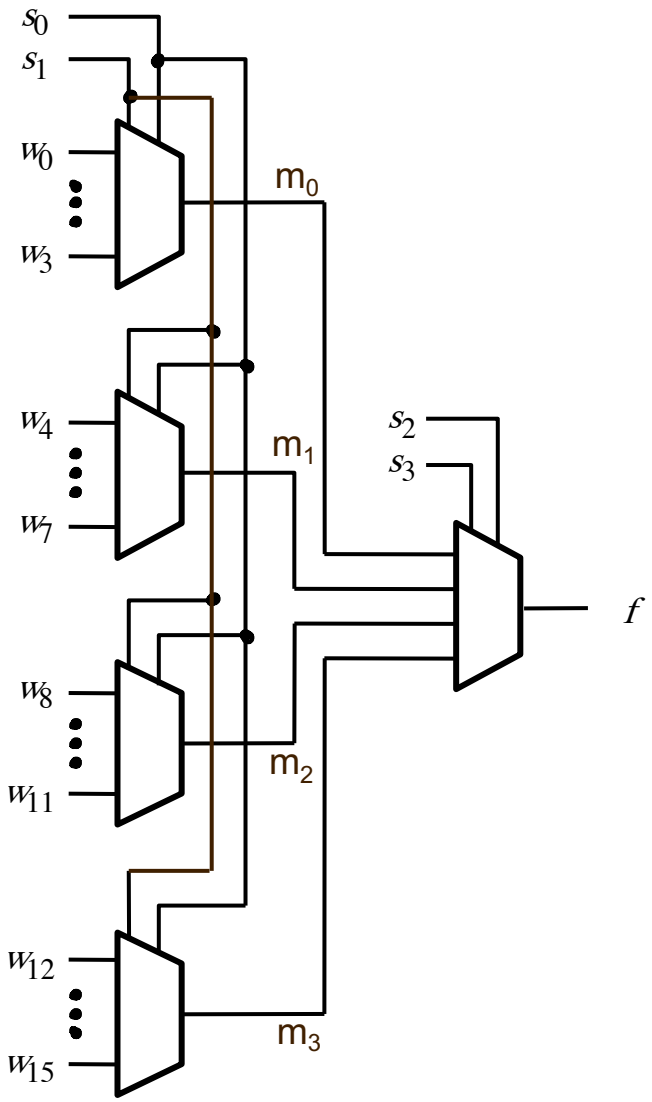
ARCHITECTURE Dataflow OF mux4to1 IS
BEGIN
    WITH s SELECT
        f <= w0 WHEN "00",
             w1 WHEN "01",
             w2 WHEN "10",
             w3 WHEN OTHERS ;
END Dataflow ;
```

Straightforward code for Example 1

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

ENTITY Example1 IS
    PORT (  w      : IN      STD_LOGIC_VECTOR(0 TO 15) ;
            s      : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            f      : OUT     STD_LOGIC ) ;
END Example1 ;
```

Example 1



Straightforward code for Example 1

```
ARCHITECTURE Structure OF Example1 IS

    COMPONENT mux4to1
        PORT (   w0, w1, w2, w3      : IN     STD_LOGIC ;
                 s                   : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                 f                   : OUT    STD_LOGIC ) ;
    END COMPONENT ;

    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

BEGIN
    Mux1: mux4to1 PORT MAP ( w(0),  w(1),  w(2),  w(3),   s(1 DOWNTO 0), m(0) ) ;
    Mux2: mux4to1 PORT MAP ( w(4),  w(5),  w(6),  w(7),   s(1 DOWNTO 0), m(1) ) ;
    Mux3: mux4to1 PORT MAP ( w(8),  w(9),  w(10), w(11),  s(1 DOWNTO 0), m(2) ) ;
    Mux4: mux4to1 PORT MAP ( w(12), w(13), w(14), w(15),  s(1 DOWNTO 0), m(3) ) ;
    Mux5: mux4to1 PORT MAP ( m(0),  m(1),  m(2),  m(3),   s(3 DOWNTO 2),  f  ) ;
END Structure ;
```

Modified code for Example 1

```
ARCHITECTURE Structure OF Example1 IS

    COMPONENT mux4to1
        PORT (    w0, w1, w2, w3      : IN     STD_LOGIC ;
                  s                   : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                  f                   : OUT    STD_LOGIC ) ;
    END COMPONENT ;

    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

BEGIN
    G1: FOR i IN 0 TO 3 GENERATE
        Muxes: mux4to1 PORT MAP (
                  ……, ……., ……., ……., ……., ……. ) ;
    END GENERATE ;
    Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;
END Structure ;
```

Modified code for Example 1

```
ARCHITECTURE Structure OF Example1 IS

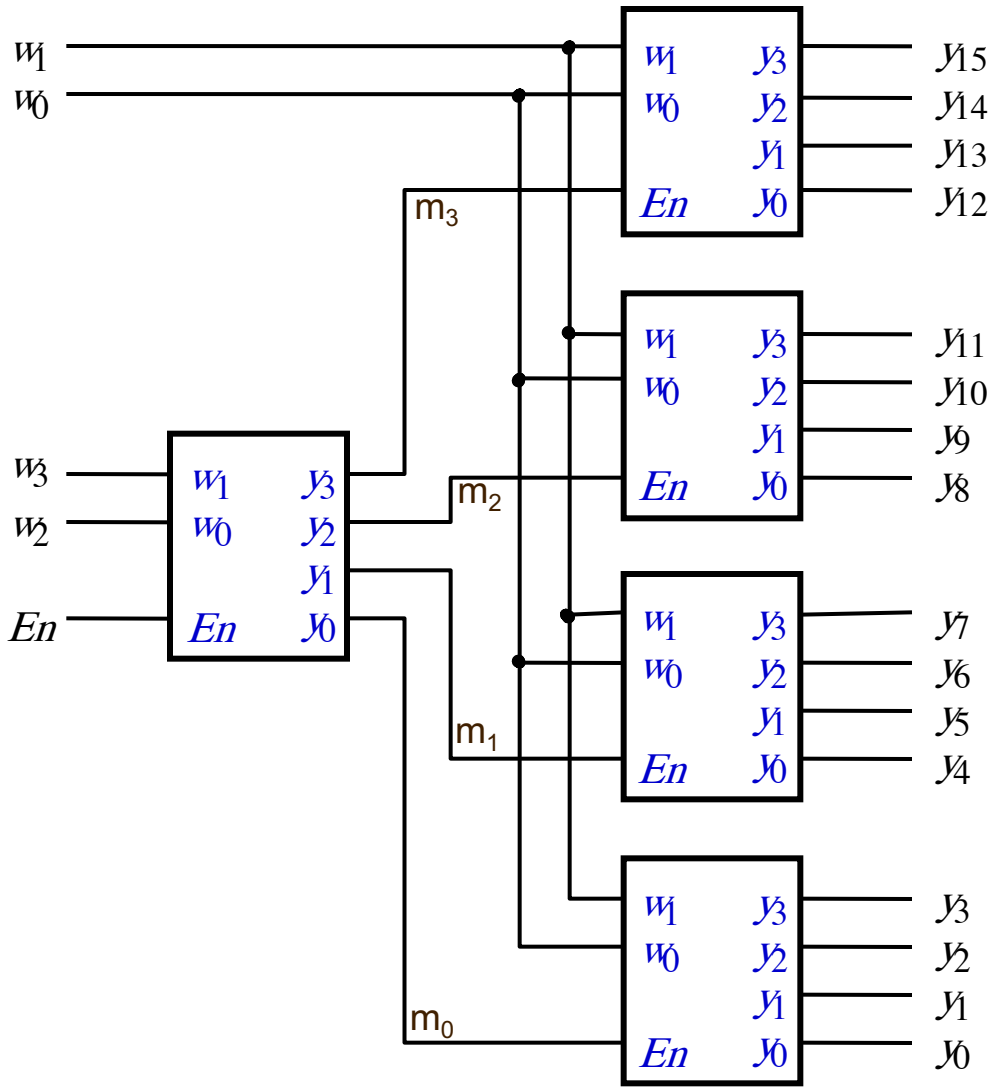
    COMPONENT mux4to1
        PORT (    w0, w1, w2, w3      : IN     STD_LOGIC ;
                  s                   : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                  f                   : OUT    STD_LOGIC ) ;
    END COMPONENT ;

    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

BEGIN
    G1: FOR i IN 0 TO 3 GENERATE
        Muxes: mux4to1 PORT MAP (
                  w(4*i), w(4*i+1), w(4*i+2), w(4*i+3), s(1 DOWNTO 0), m(i) ) ;
    END GENERATE ;
    Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;
END Structure ;
```

Example 2

Example 2


w_1
w_0
w_3
w_2
En
m_3
m_2
m_1
m_0
y_3
y_2
y_1
y_0
y_{15}
y_{14}
y_{13}
y_{12}
y_{11}
y_{10}
y_9
y_8
y_7
y_6
y_5
y_4
y_3
y_2
y_1
y_0

A 2-to-4 binary 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 ;

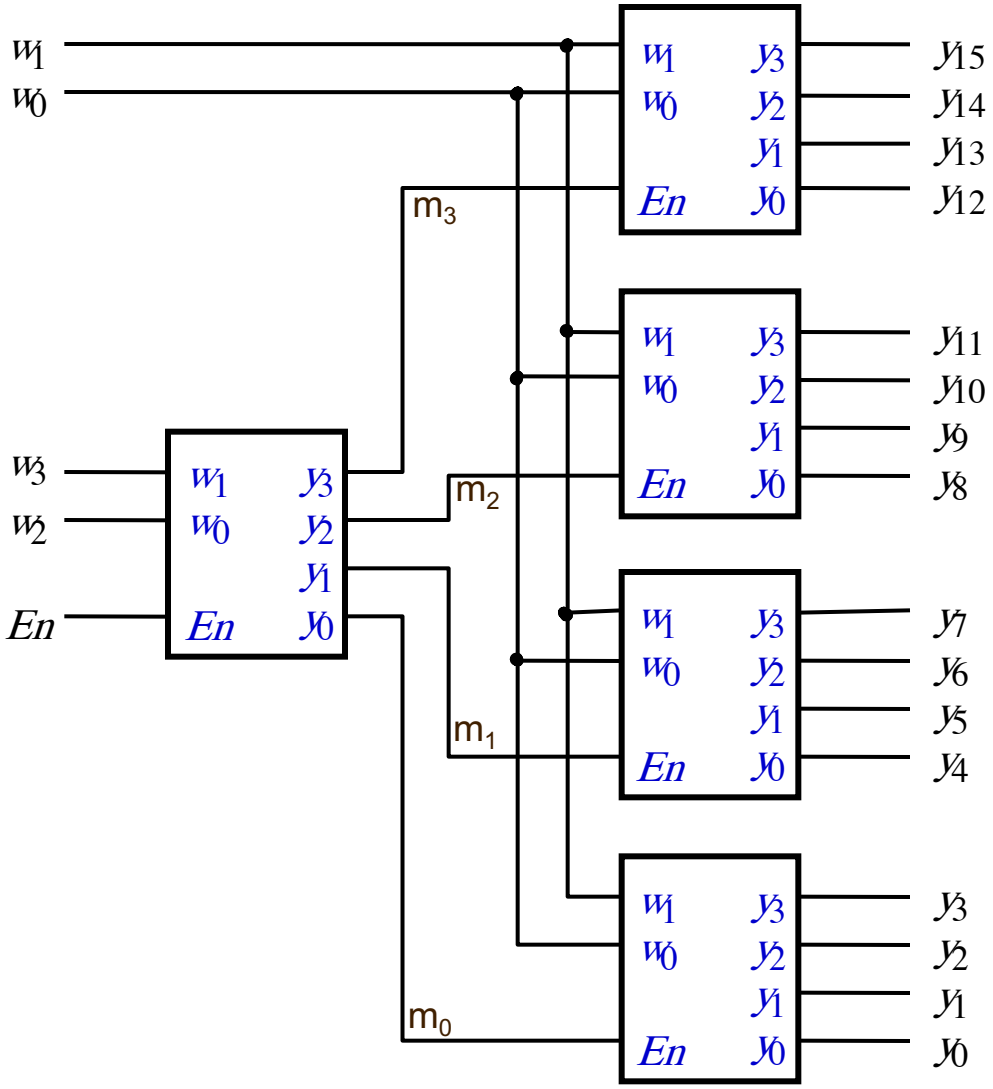
ARCHITECTURE Dataflow OF dec2to4 IS
    SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
BEGIN
    Enw <= En & w ;
    WITH Enw SELECT
           y <= "0001" WHEN "100",
                "0010" WHEN "101",
                "0100" WHEN "110",
                "1000" WHEN "111",
                "0000" WHEN OTHERS ;
END Dataflow ;
```

VHDL code for Example 2 (1)

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

ENTITY dec4to16 IS
    PORT (w     : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          En    : IN     STD_LOGIC ;
          y     : OUT    STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END dec4to16 ;
```

Example 2


w1
w0
w3
w2
En
m3
m2
m1
m0
y15
y14
y13
y12
y11
y10
y9
y8
y7
y6
y5
y4
y3
y2
y1
y0

VHDL code for Example 2 (2)

```
ARCHITECTURE Structure OF dec4to16 IS

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

    SIGNAL m : STD_LOGIC_VECTOR(3 DOWNTO 0) ;

BEGIN
    Dec_left: dec2to4 PORT MAP ( w(3 DOWNTO 2), En, m ) ;
    Dec_r0: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(0),  y(3 DOWNTO 0)  );
    Dec_r1: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(1),  y(7 DOWNTO 4)  );
    Dec_r2: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(2),  y(11 DOWNTO 8)  );
    Dec_r3: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(3),  y(15 DOWNTO 12)  );
END Structure ;
```

VHDL code for Example 2 (2)

```
ARCHITECTURE Structure OF dec4to16 IS

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

    SIGNAL m : STD_LOGIC_VECTOR(3 DOWNTO 0) ;

BEGIN
    Dec_left: dec2to4 PORT MAP ( w(3 DOWNTO 2), En, m ) ;

    G1: FOR i IN 0 TO 3 GENERATE
        Dec_ri: dec2to4 PORT MAP ( ……., ……., ……..);
    END GENERATE ;
END Structure ;
```

VHDL code for Example 2 (2)

```
ARCHITECTURE Structure OF dec4to16 IS

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

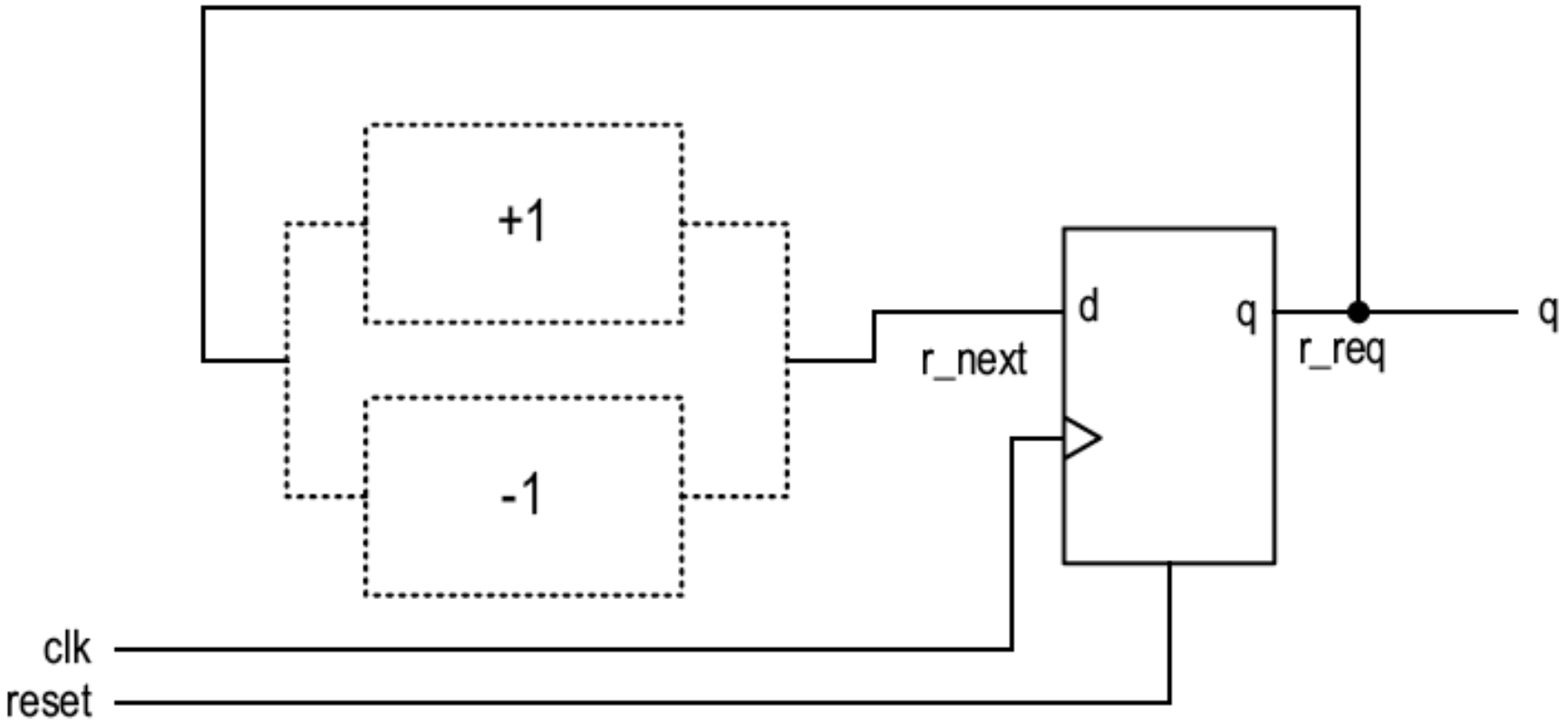
    SIGNAL m : STD_LOGIC_VECTOR(3 DOWNTO 0) ;

BEGIN
    Dec_left: dec2to4 PORT MAP ( w(3 DOWNTO 2), En, m ) ;

    G1: FOR i IN 0 TO 3 GENERATE
        Dec_ri: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(i),  y(4*i+3 DOWNTO 4*i)  );
    END GENERATE ;
END Structure ;
```

Example 3
Up-or-down Free Running Counter

Up-or-down Free Running Counter



Up-or-down Free Running Counter (1)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity up_or_down_counter is
   generic(
      WIDTH: natural:=4;
      UP: natural:=0
   );

   port(
      clk, reset: in std_logic;
      q: out std_logic_vector(WIDTH-1 downto 0)
   );
end up_or_down_counter;
```

Up-or-down Free Running Counter (2)

```
architecture mixed of up_or_down_counter is

   signal r_reg: unsigned(WIDTH-1 downto 0);
   signal r_next: unsigned(WIDTH-1 downto 0);

begin
   -- register
   process(clk, reset)
   begin
      if (reset='1') then
         r_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         r_reg <= r_next;
      end if;
   end process;
```

Up-or-down Free Running Counter (3)

```
-- next-state logic
   inc_gen:  -- incrementor
   if UP=1 generate
      r_next <= r_reg + 1;
   end generate;

   dec_gen:  --decrementor
   if UP/=1 generate
      r_next <= r_reg – 1;
   end generate;

   -- output logic
   q <= std_logic_vector(r_reg);

end mixed;
```

New in VHDL-2008:

More powerful if-generate statements

```
-- old coding style
label1: if condition
   generate
    -- first alternative
    ...
end generate ;

label2: if not condition
   generate
    -- second alternative
    ...
end generate ;
```



```
-- same thing in new style
label: if condition generate
    -- first alternative
    ...
else generate
    -- second alternative
    ...
end generate ;
```





New in VHDL-2008:

Full If-generate and case-generate syntax

```
-- syntax similar to
-- regular if clause
label: if condition1
   generate
    -- first alternative
    ...
elsif condition2 generate
    -- second alternative
    ...
    ...
else generate
    -- last default
   alternative
    ...
end generate ;
```

```
-- syntax similar to
-- regular case clauses
label: case
   tested_expression
   generate
    when choice1 =>
        -- first alternative
        ...
    when choice2 =>
        -- second alternative
        ...
    when others =>
        -- last default
   alternative
        ...
end generate ;
```



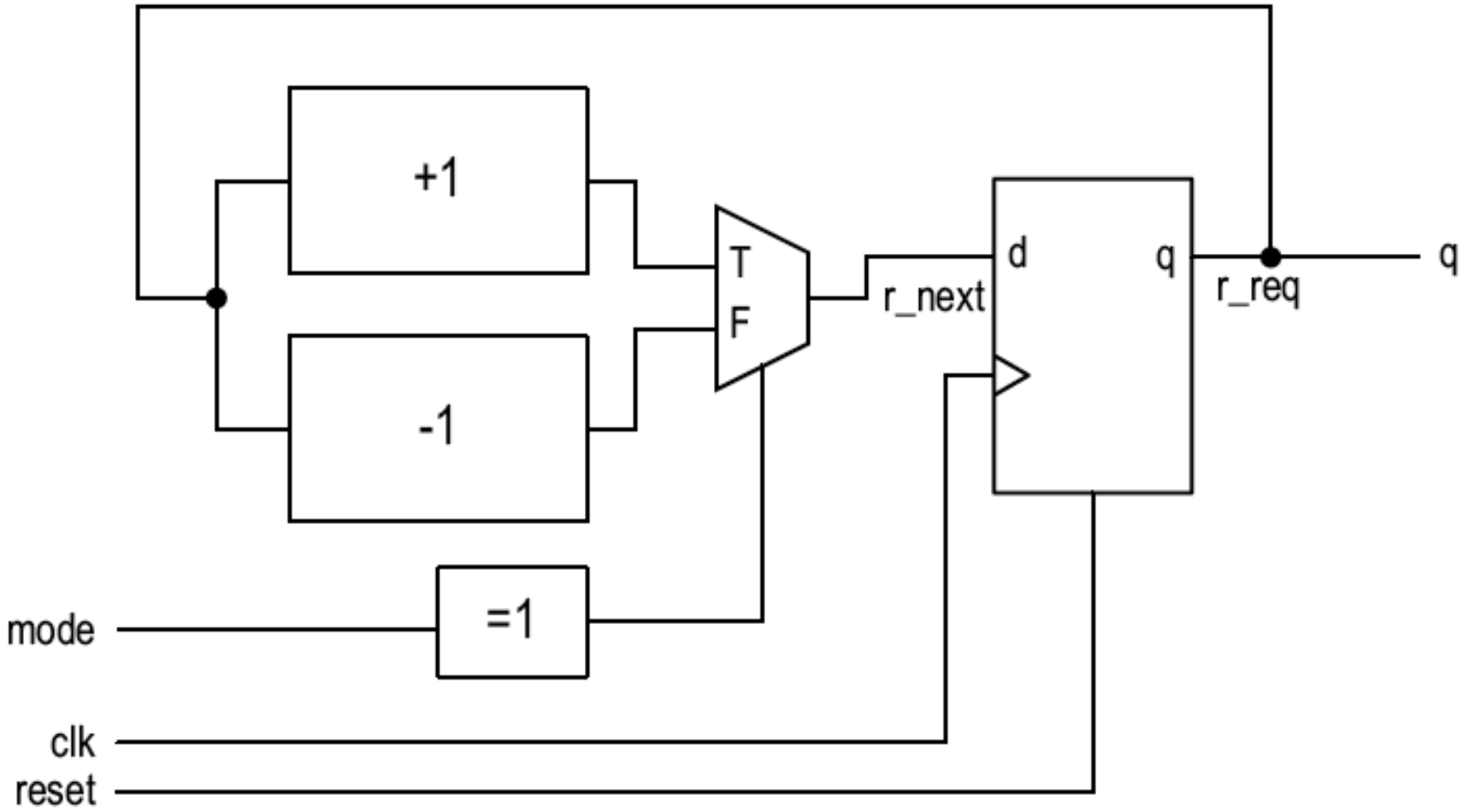

New in VHDL-2008:
Example of case-generate statement

```
    -- constant version_number : integer := 1 ;
    constant version_number : integer := 2 ;
    ...
    ...
begin
    gencase: case version_number generate
      when 1 =>
        U1: comp1 port map( resetN , clk, din , dout ) ;
      when 2 =>
        U2: comp2 port map( resetN , clk, din , dout ) ;
      when others =>
        assert false
           report "Incorrect version number"
           severity error ;
    end generate ;
```




Example 4
Up-and-down Free Running Counter

Up-and-down Free Running Counter



Up-and-down Free Running Counter (1)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity up_and_down_counter is
   generic(WIDTH: natural:=4);
   port(
      clk, reset: in std_logic;
      mode: in std_logic;
      q: out std_logic_vector(WIDTH-1 downto 0)
   );
end up_and_down_counter;
```

Up-and-down Free Running Counter (2)

```
architecture arch of up_and_down_counter is

   signal r_reg: unsigned(WIDTH-1 downto 0);
   signal r_next: unsigned(WIDTH-1 downto 0);

begin
   -- register
   process(clk, reset)
   begin
      if (reset='1') then
         r_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         r_reg <= r_next;
      end if;
   end process;
```

Up-and-down Free Running Counter (3)

```
 -- next-state logic

   r_next <= r_reg + 1 when mode='1' else
                r_reg  - 1;

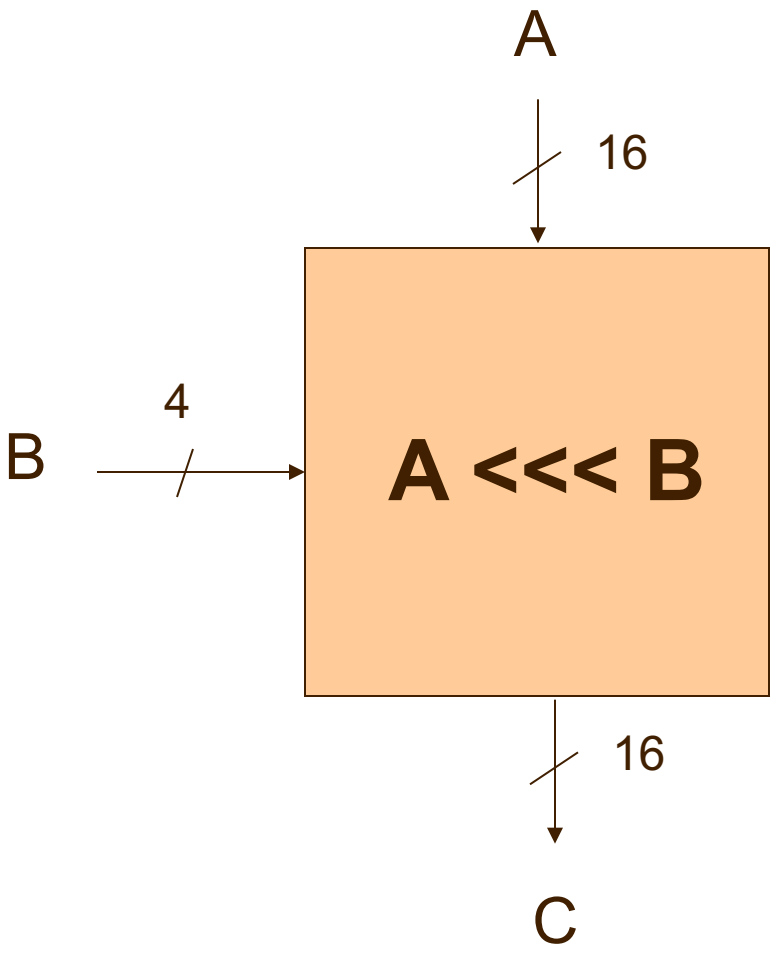
   -- output logic

   q <= std_logic_vector(r_reg);

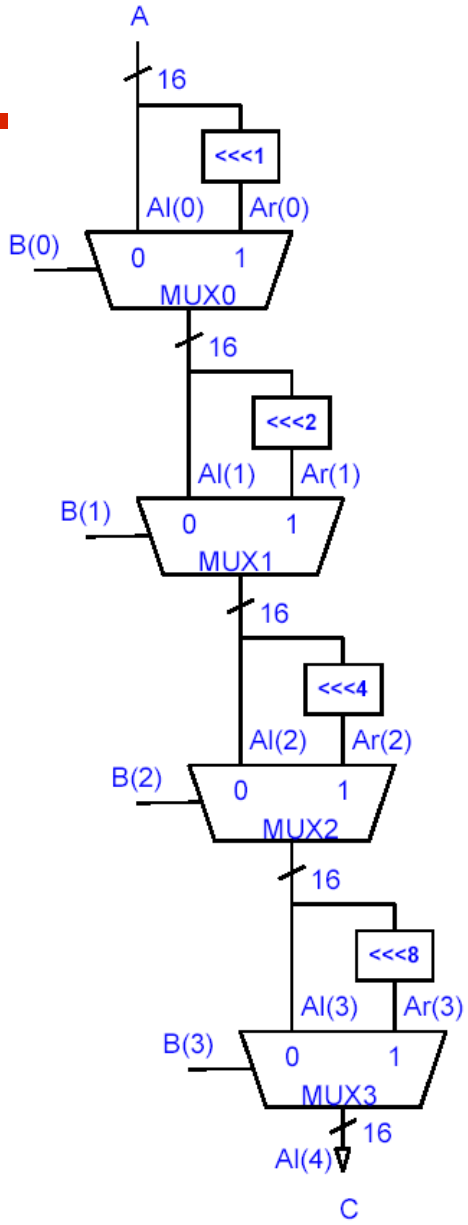
end arch;
```

Example 5
Variable Rotator

Example 5: Variable rotator - Interface



Block diagram



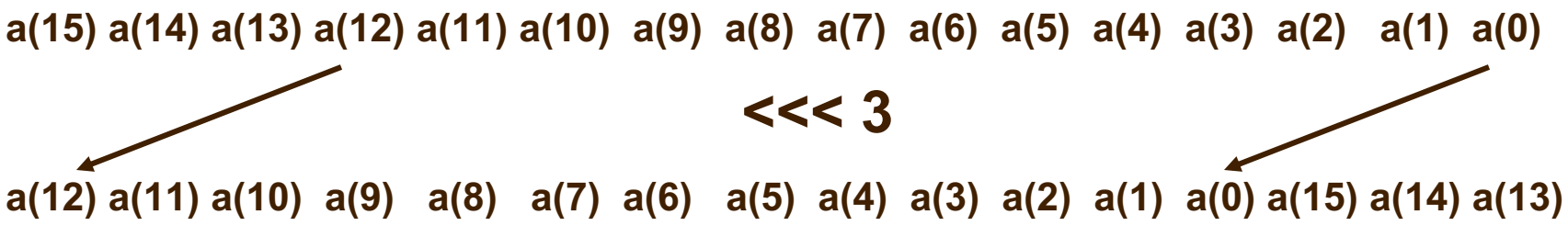
VHDL code for a 16-bit 2-to-1 Multiplexer

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

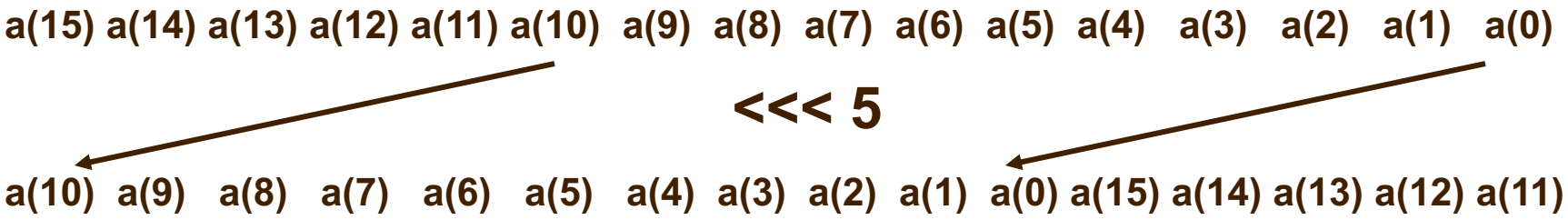
ENTITY mux2to1_16 IS
    PORT (    w0          : IN     STD_LOGIC_VECTOR(15 DOWNTO 0);
              w1          : IN     STD_LOGIC_VECTOR(15 DOWNTO 0);
              s           : IN     STD_LOGIC ;
              f           : OUT    STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END mux2to1_16 ;

ARCHITECTURE dataflow OF mux2to1_16 IS
BEGIN
    f <= w0 WHEN s = '0' ELSE w1 ;
END dataflow ;
```

Fixed rotation




```
y <= a(12 downto 0) & a(15 downto 13);
```




```
y <= a(10 downto 0) & a(15 downto 11);
```

Fixed rotation by L positions

a(15) a(14) a(13) a(12) a(11) a(10) a(9) a(8) a(7) a(6) a(5) a(4) a(3) a(2) a(1) a(0)

<<< L

a(……) a(……..) . . . . a(1) a(0) a(15) a(14) . . . . . a(……..) a(……..)

y <= a(……..) & a(……..);

Fixed rotation by L positions

a(15) a(14) a(13) a(12) a(11) a(10) a(9) a(8) a(7) a(6) a(5) a(4) a(3) a(2) a(1) a(0)

<<< L

a(15-L) a(15-L-1) . . . . . a(1) a(0) a(15) a(14) . . . . . a(15-L+2) a(15-L+1)

y <= a(15-L downto 0) & a(15 downto 15-L+1);

VHDL code for for a fixed 16-bit rotator

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

ENTITY fixed_rotator_left_16 IS
      GENERIC ( L : INTEGER := 1);
      PORT (     a              : IN       STD_LOGIC_VECTOR(15 DOWNTO 0);
                 y              : OUT      STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END fixed_rotator_left_16 ;

ARCHITECTURE dataflow OF fixed_rotator_left_16 IS
BEGIN
      y  <=  a(15-L downto 0) & a(15 downto 15-L+1);
END dataflow ;
```

Structural VHDL code for for a variable 16-bit rotator (1)

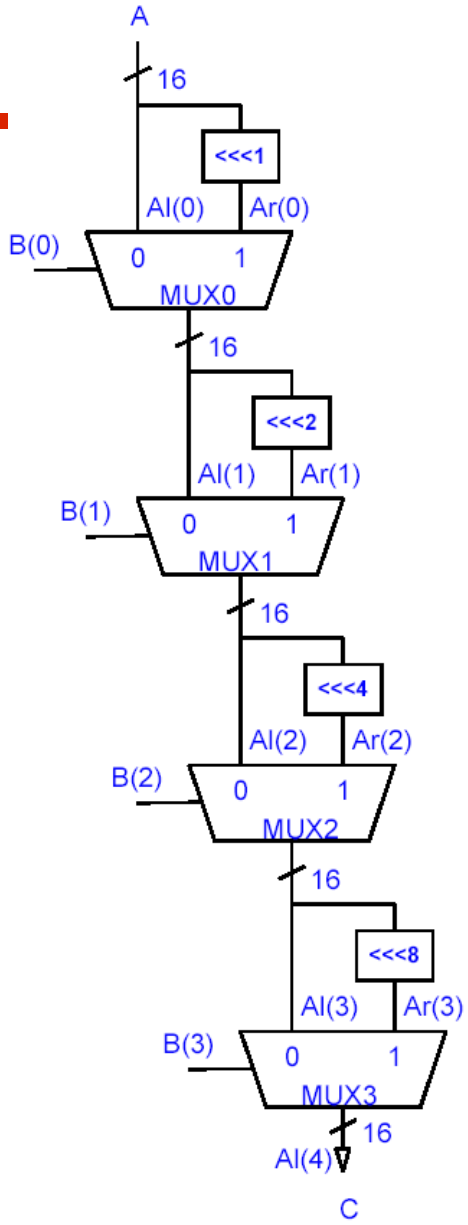
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY variable_rotator_16 is
        PORT(
                        A : IN STD_LOGIC_VECTOR(15 downto 0);
                        B : IN STD_LOGIC_VECTOR(3 downto 0);
                        C : OUT STD_LOGIC_VECTOR(15 downto 0)
             );
END variable_rotator_16;


ARCHITECTURE structural OF variable_rotator_16 IS

TYPE array16 IS ARRAY (0 to 4) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL Al : array16;
SIGNAL Ar : array16;
```

Block diagram



Structural VHDL code for for a variable 16-bit rotator (2)

```
BEGIN
        Al(0) <= A;

        G:
        FOR i IN 0 TO 3 GENERATE
            ROT_I: ENTITY work.fixed_rotator_left_16(dataflow)
                        GENERIC MAP (L => ……)
                        PORT MAP ( a => …….. ,
                                   y => ……..);
             MUX_I:  ENTITY work.mux2to1_16(dataflow)
                        PORT MAP (w0 => ……..,
                                  w1 => ……..,
                                  s  => ……..,
                                  f  => ……);
        END GENERATE;

        C <= Al(4);

END structural;
```

Structural VHDL code for for a variable 16-bit rotator (2)

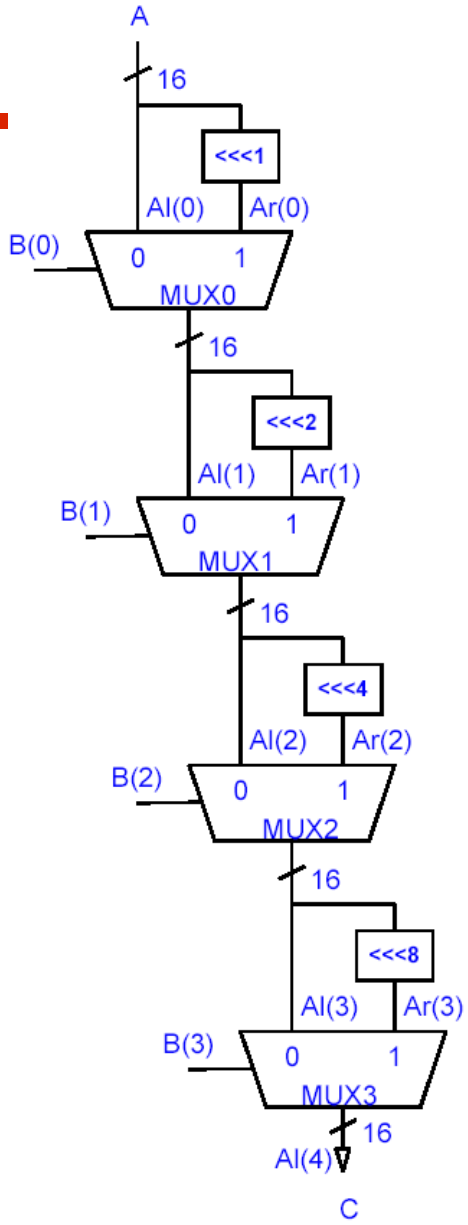
```
BEGIN
        Al(0) <= A;

        G:
        FOR i IN 0 TO 3 GENERATE
            ROT_I: ENTITY work.fixed_rotator_left_16(dataflow)
                    GENERIC MAP (L => 2** i)
                    PORT MAP ( a => Al(i) ,
                                    y => Ar(i));
             MUX_I:  ENTITY work.mux2to1_16(dataflow)
                    PORT MAP (w0 => Al(i),
                                   w1 => Ar(i),
                                    s   => B(i),
                                    f   => Al(i+1));
        END GENERATE;

        C <= Al(4);

END structural;
```

Block diagram



Dataflow VHDL code for for a variable 16-bit rotator (3)

```
BEGIN
    Al(0) <= A;

    G:
    FOR i IN 0 TO 3 GENERATE
        Ar(i) <= ……..;
        Al(i+1) <= ……..;
    END GENERATE;

    C <= Al(4);

END dataflow;
```

Dataflow VHDL code for for a variable 16-bit rotator (3)

```
BEGIN
        Al(0) <= A;

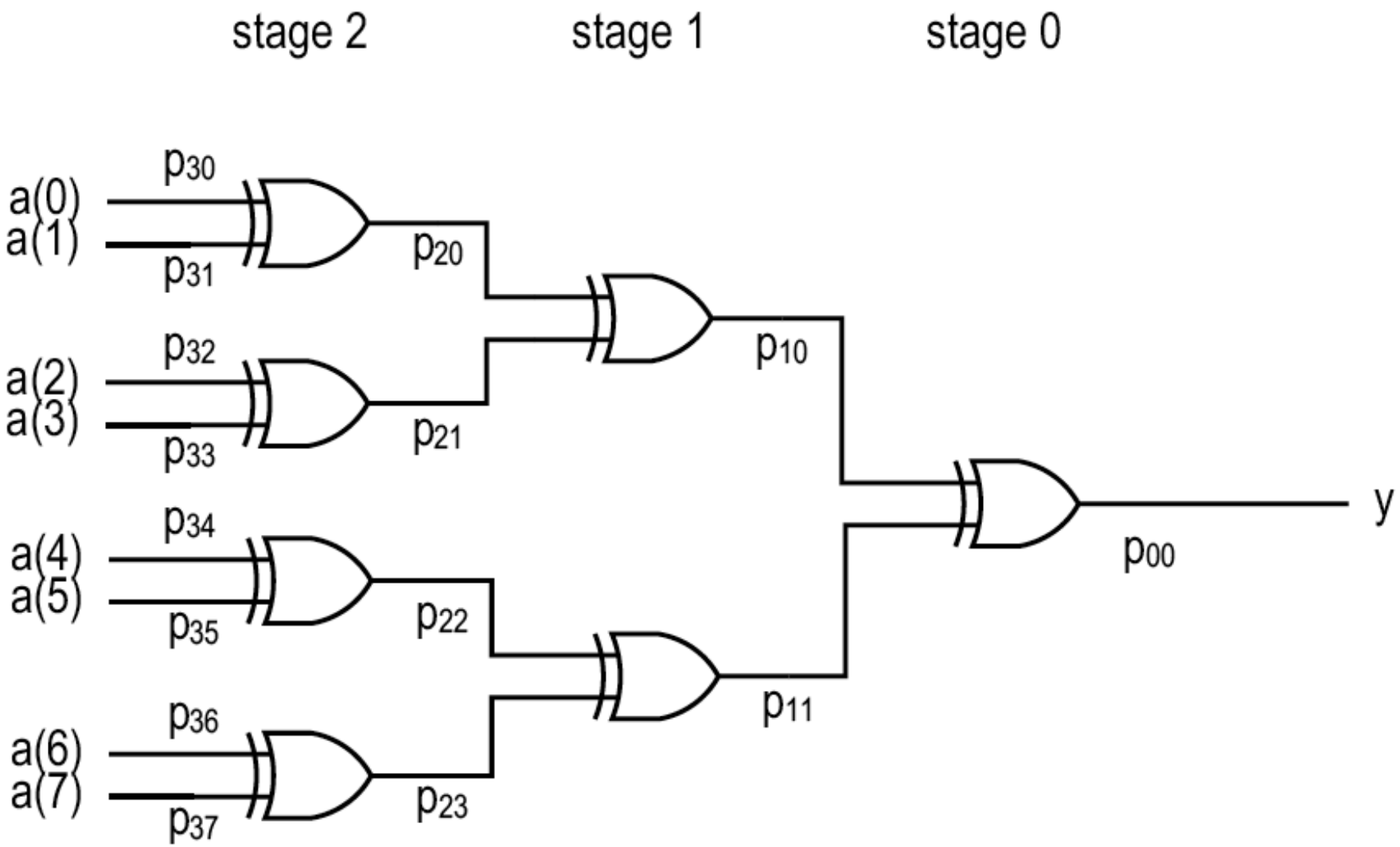
        G:
        FOR i IN 0 TO 3 GENERATE
                Ar(i) <= Al(i)(15-2**i downto 0) & Al(i)(15 downto 15-2**i+1);
                Al(i+1) <= Al(i) when B(i)='0' else Ar(i);
        END GENERATE;

        C <= Al(4);

END dataflow;
```

Example 6
XOR Tree

XOR Tree


stage 2
stage 1
stage 0
a(0)
a(1)
a(2)
a(3)
a(4)
a(5)
a(6)
a(7)
p30
p31
p32
p33
p34
p35
p36
p37
p20
p21
p22
p23
p10
p11
p00
y

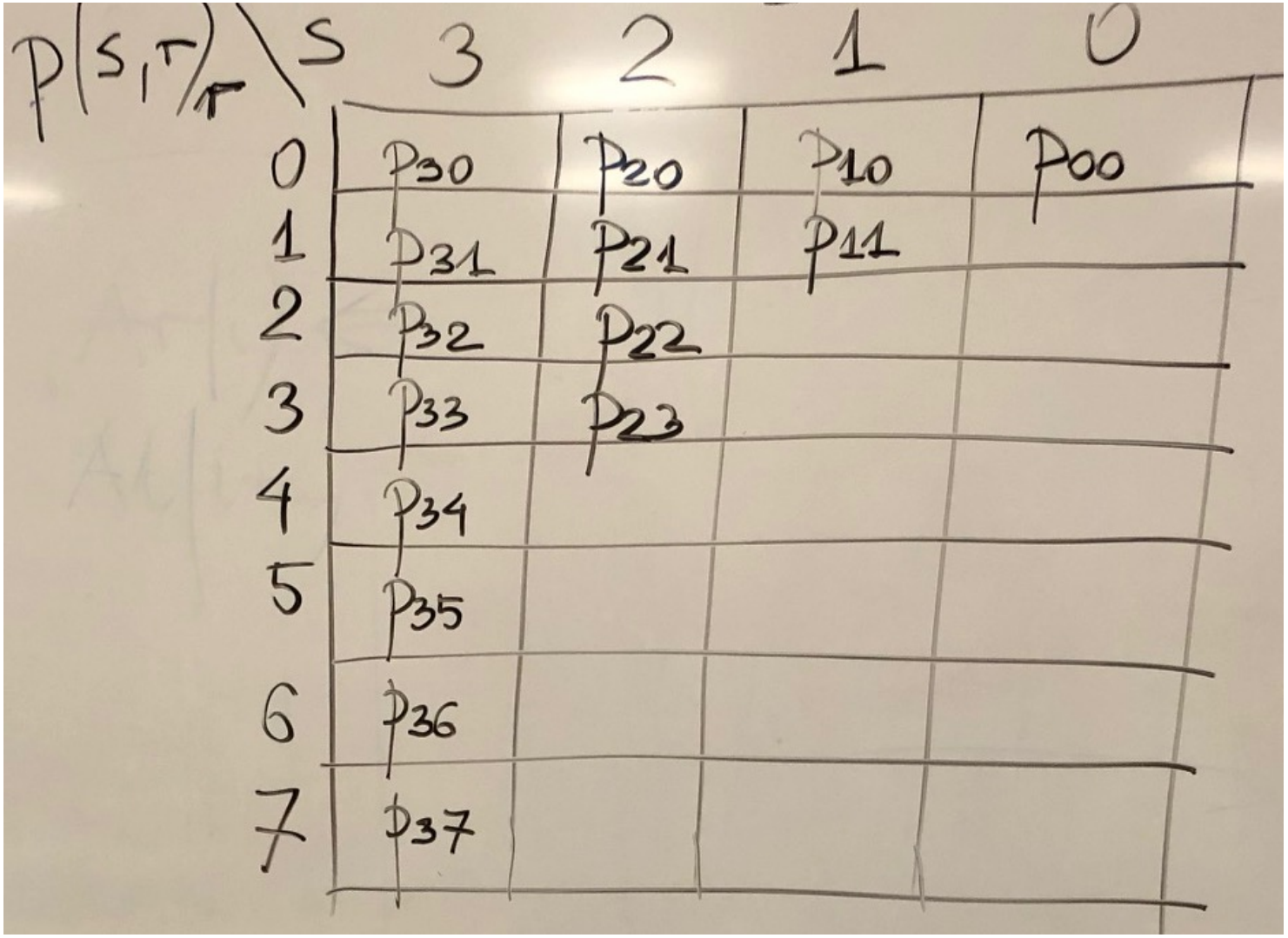
XOR Tree (1)

```
library ieee;
use ieee.std_logic_1164.all;
use work.util_pkg.all;

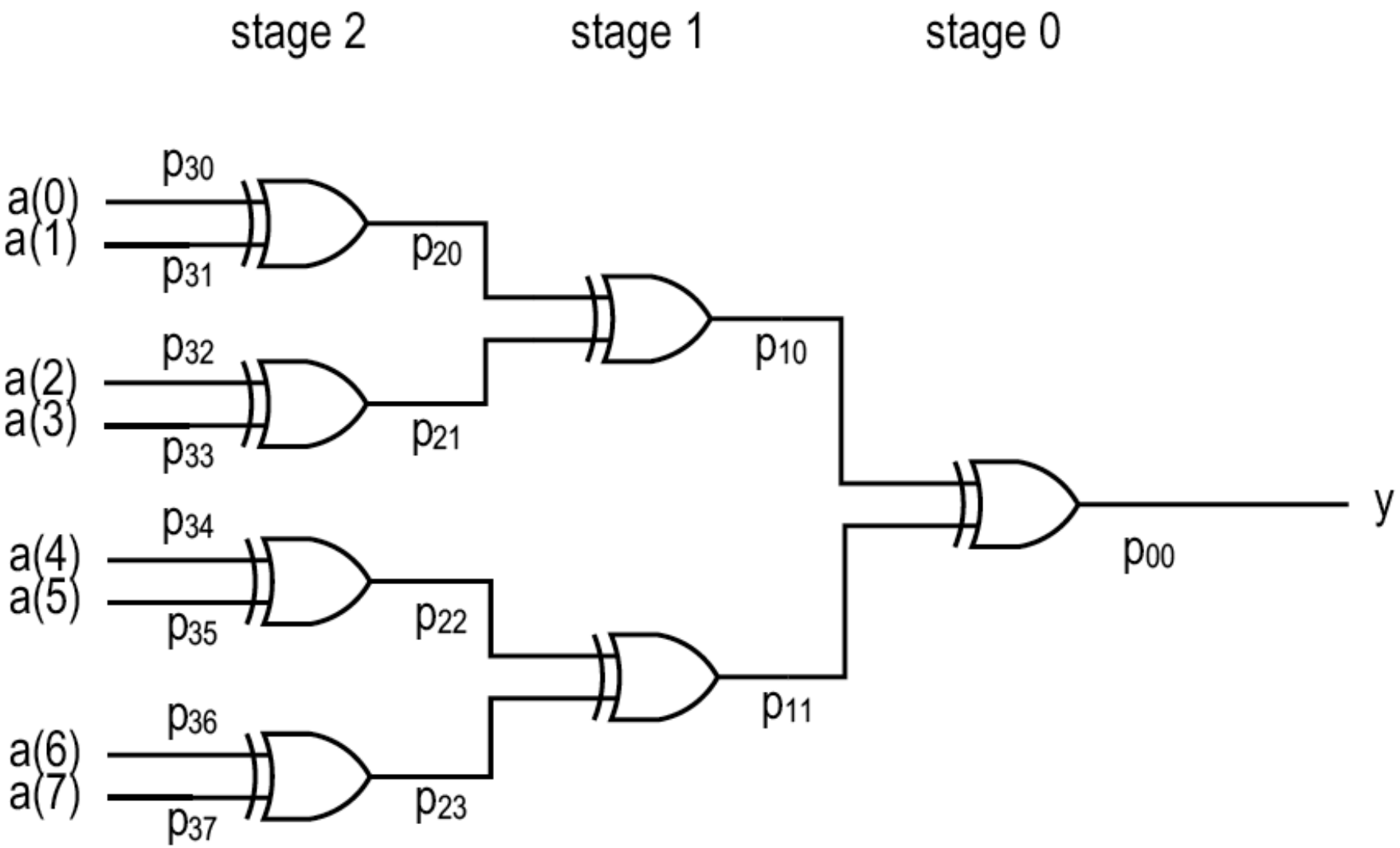
entity reduced_xor is
   generic(WIDTH: natural:=8);
   port(
      a: in std_logic_vector(WIDTH-1 downto 0);
      y: out std_logic
   );
end reduced_xor;

architecture gen_tree_arch of reduced_xor is
   constant STAGE: natural:= log2c(WIDTH);
   signal p:  std_logic_2d(STAGE downto 0, WIDTH-1 downto 0);
```

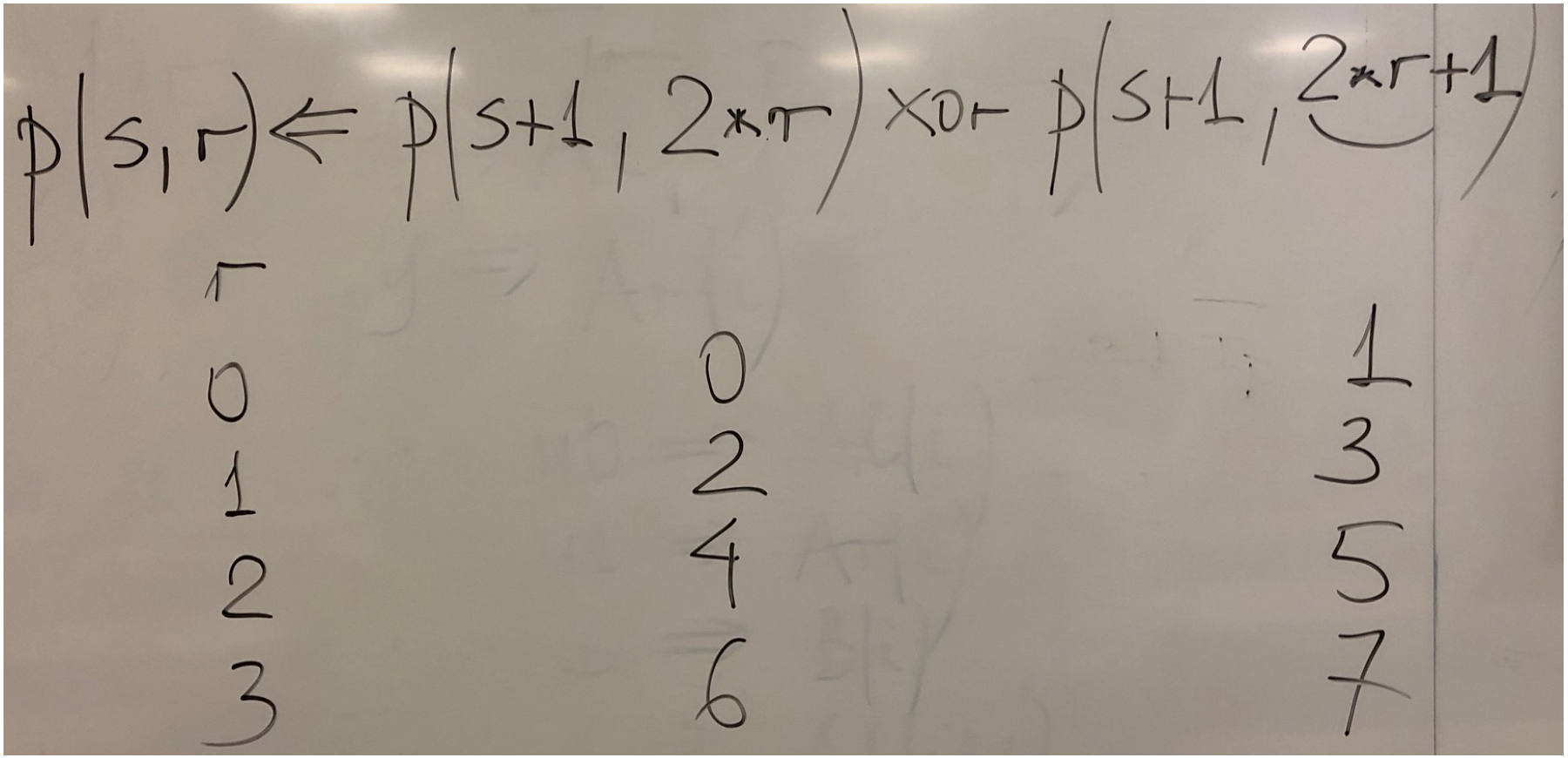
XOR Tree Internal Structure



XOR Tree


stage 2
stage 1
stage 0
a(0)
a(1)
a(2)
a(3)
a(4)
a(5)
a(6)
a(7)
p30
p31
p32
p33
p34
p35
p36
p37
p20
p21
p22
p23
p10
p11
p00
y

XOR Tree – Main Instruction



XOR Tree (2)

```
begin
   -- rename input signal
   in_gen:
   for i in 0 to (WIDTH-1) generate
      p(STAGE, i) <= a(i);
   end generate;
   -- replicated structure
   stage_gen:
   for s in (STAGE-1) downto 0 generate
      row_gen:
      for r in 0 to (2**s-1) generate
         p(s, r) <= p(s+1, 2*r) xor p(s+1, 2*r+1);
      end generate;
   end generate;
   -- rename output signal
   y <= p(0, 0);
end gen_tree_arch;
```

util_pkg (1)

```
library ieee;
use ieee.std_logic_1164.all;

package util_pkg is

   type std_logic_2d is
      array(integer range <>, integer range <>) of std_logic;
   function log2c (n: integer) return integer;

end util_pkg ;
```

Array Data Type

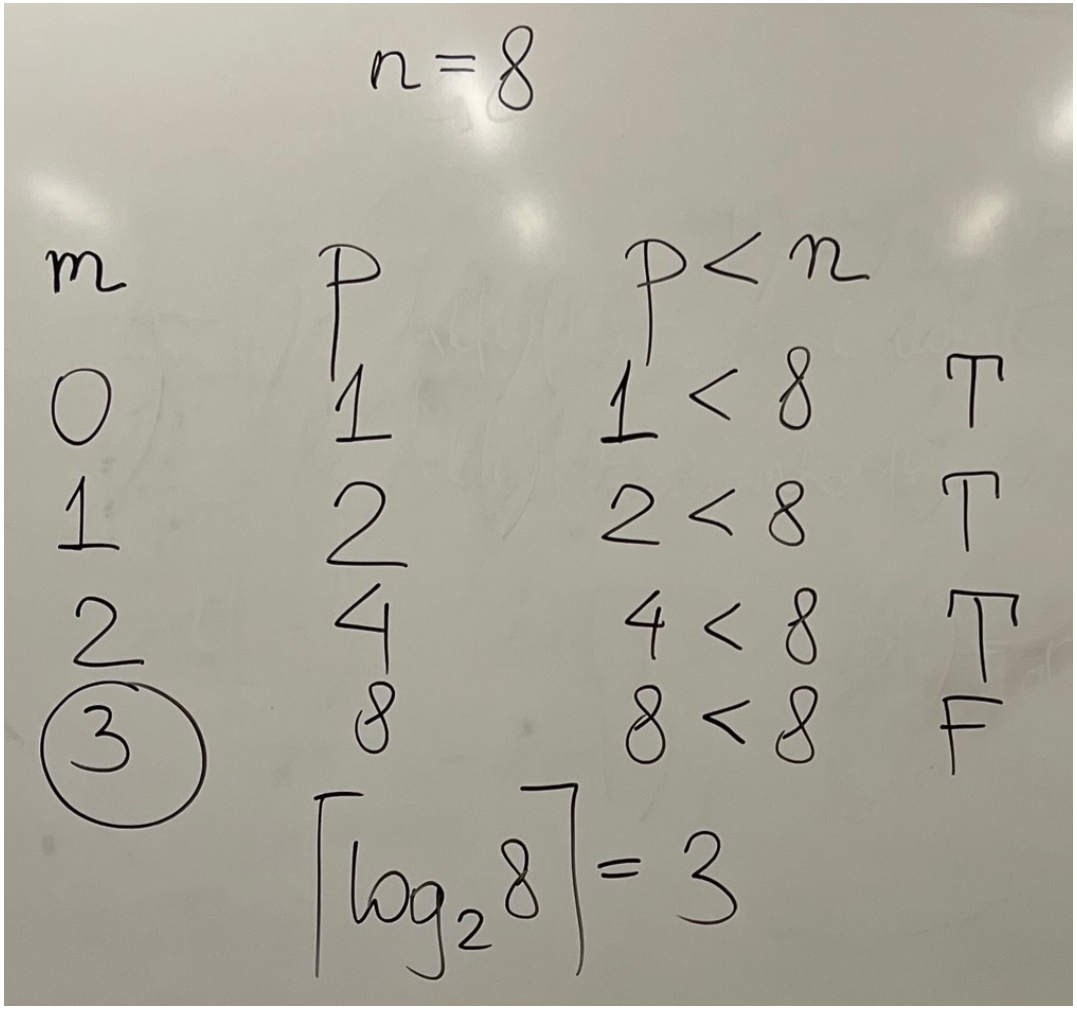
Array Type

```
TYPE data_type_name IS ARRAY (range_1, range2, ...)
    OF element_data_type;
```

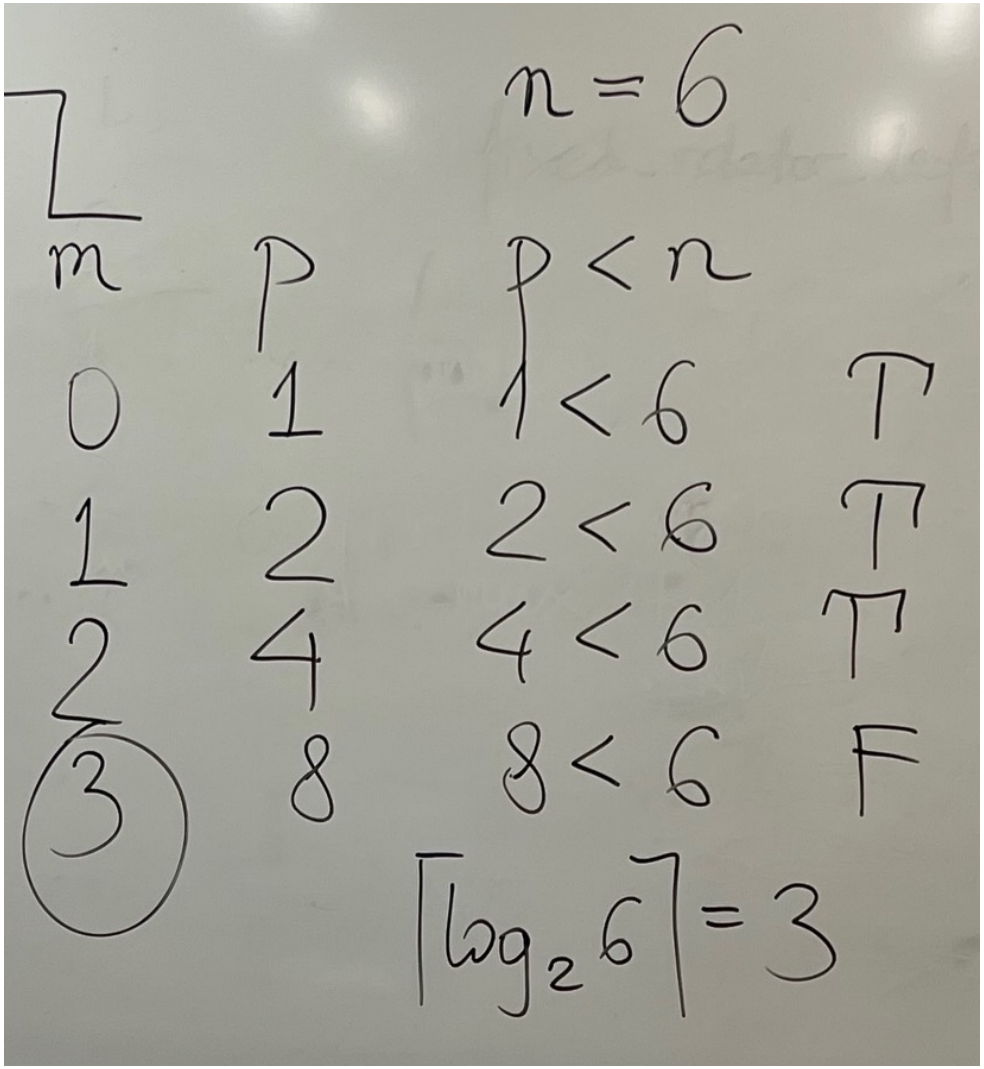
util_pkg (2) – log2c(n)

```
--package body
package body util_pkg is
   function log2c(n: integer) return integer is
      variable m, p: integer;
   begin
      m := 0;
      p := 1;
      while p < n loop
         m := m + 1;
         p := p * 2;
      end loop;
      return m;
   end log2c;
end util_pkg;
```

log2c(8)



log2c(6)



XOR Tree with Arbitrary Input Size (1)

```
begin

   -- rename input signal
   in_gen:
   for i in 0 to (WIDTH-1) generate
      p(STAGE,i) <= a(i);
   end generate;

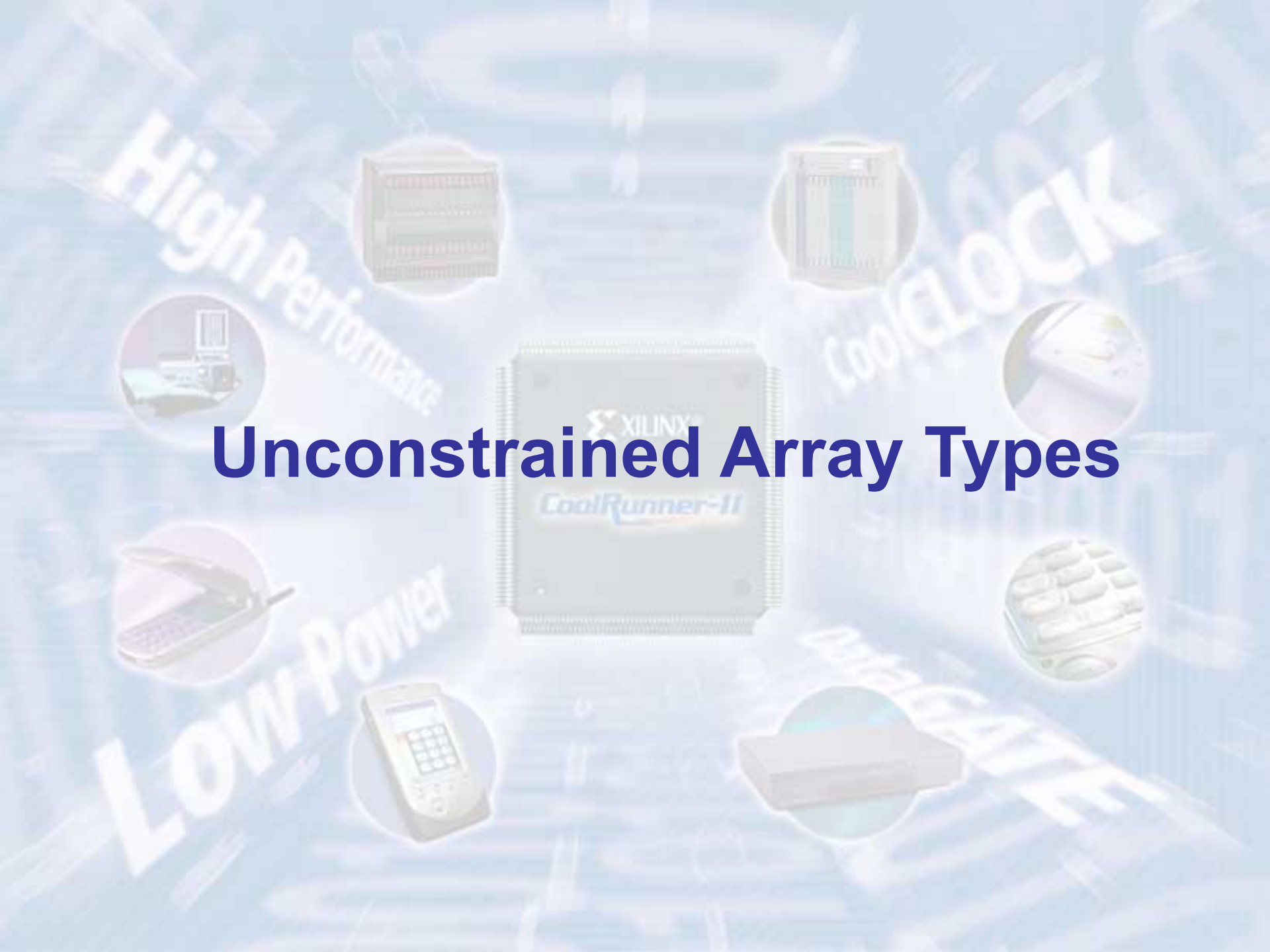
   -- padding 0's
   pad0_gen:
   if WIDTH < (2**STAGE) generate
      zero_gen:
      for i in WIDTH to (2**STAGE-1) generate
         p(STAGE,i) <= '0';
      end generate;
   end generate;
```

XOR Tree with Arbitrary Input Size (2)

```
-- replicated structure
  stage_gen:
  for s in (STAGE-1) downto 0 generate
     row_gen:
     for r in 0 to (2**s-1) generate
        p(s,r) <= p(s+1,2*r) xor p(s+1,2*r+1);
     end generate;
  end generate;

  -- rename output signal
  y <= p(0,0);

end gen_tree2_arch;
```



Unconstrained Array Types

Predefined Unconstrained Array Types

Predefined

bit_vector	array of bits
string	array of characters

Predefined Unconstrained Array Types

Defined in the std_logic_1164 package:

```
type std_logic_vector is array (natural range <>)
        of std_logic;
```

User-defined Unconstrained Array Types

```
type std_logic_2d is
    array(integer range <>, integer range <>) of std_logic;

signal s1: std_logic_2d(3 downto 0, 5 downto 0);
signal s2: std_logic_2d(15 downto 0, 31 downto 0);
signal s3: std_logic_2d(7 downto 0, 1 downto 0);
```

User-defined Unconstrained Array Type in a package

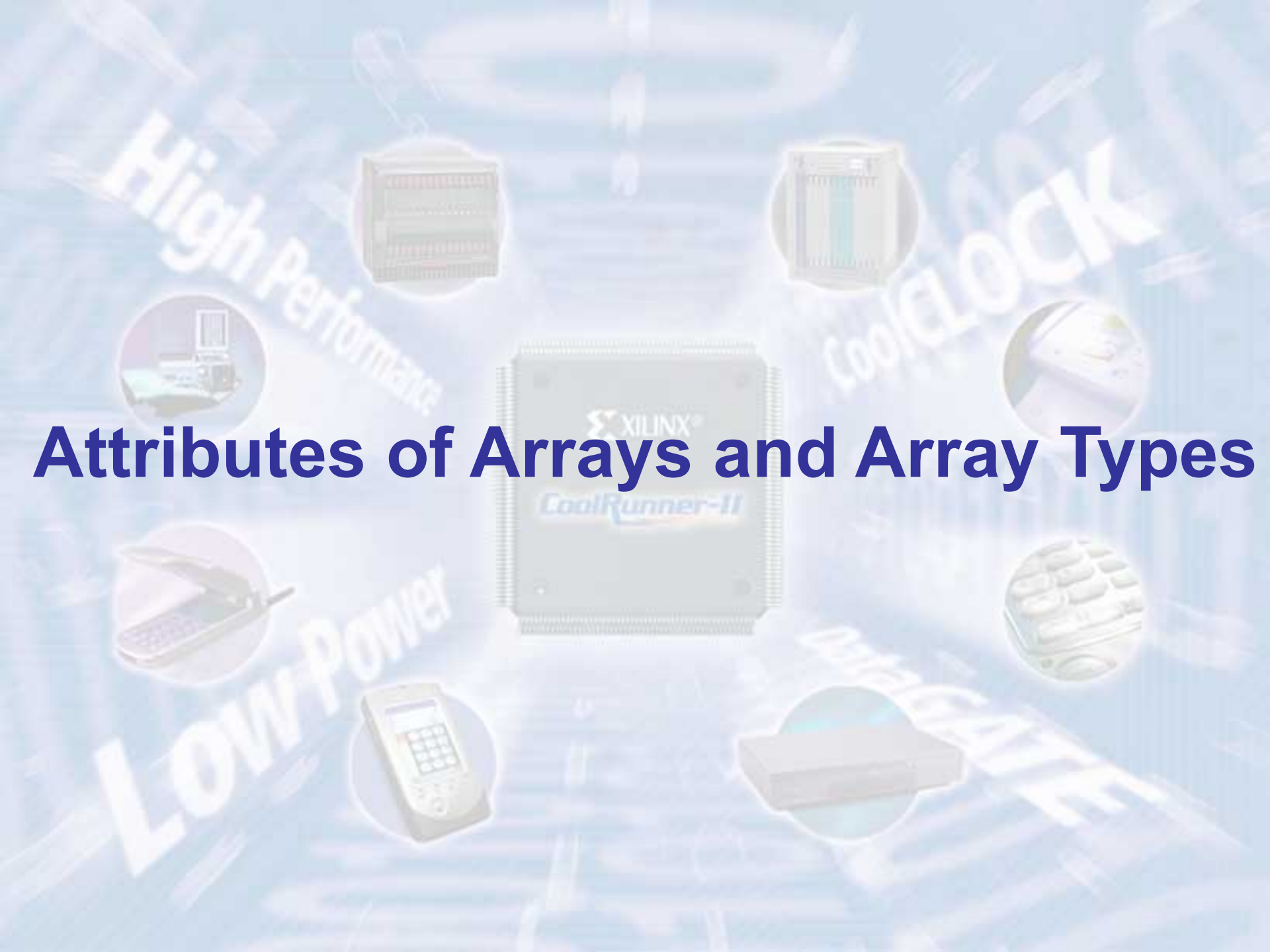
```
use work.util_pkg.all;

entity ….
  generic (
     ROW: natural;
     COL: natural)
  );
 port (
     p1: in std_logic_2d(ROW-1 downto 0, COL-1 downto 0);
     ……..
 )
architecture
    signal s1: std_logic_2d(ROW-1 downto 0, COL-1 downto 0);
```

Array-of-Arrays Data Type

```
constant  ROW : natural := 4;
constant COL : natural := 6;
type sram_row_by_col is array (ROW -1 downto 0) of
            std_logic_vector(COL – 1 downto 0);


……
signal t1: sram_row_by_col;
signal v1: std_logic_vector(COL-1 downto 0);
signal b1: std_logic;
……
t1 <= ("000101", "101001", others => (others => '0'));
b1 <= t1(3)(0);
v1 <= t1(2);
```



Attributes of Arrays and Array Types

Array Attributes

A'left(N)	left bound of index range of dimension N of A
A'right(N)	right bound of index range of dimension N of A
A'low(N)	lower bound of index range of dimension N of A
A'high(N)	upper bound of index range of dimension N of A
A'range(N)	index range of dimension N of A
A'reverse_range(N)	reversed index range of dimension N of A
A'length(N)	length of index range of dimension N of A
A'ascending(N)	true if index range of dimension N of A is an ascending range, false otherwise

Array Attributes - Examples

type A **is array** (1 **to** 4, 31 **downto** 0);

A'left(1) =

A'right(2) =

A'low(1) =

A'high(2) =

A'range(1) =

A'length(2) =

A'ascending(2) =

Array Attributes - Examples

type A **is array** (1 **to** 4, 31 **downto** 0);

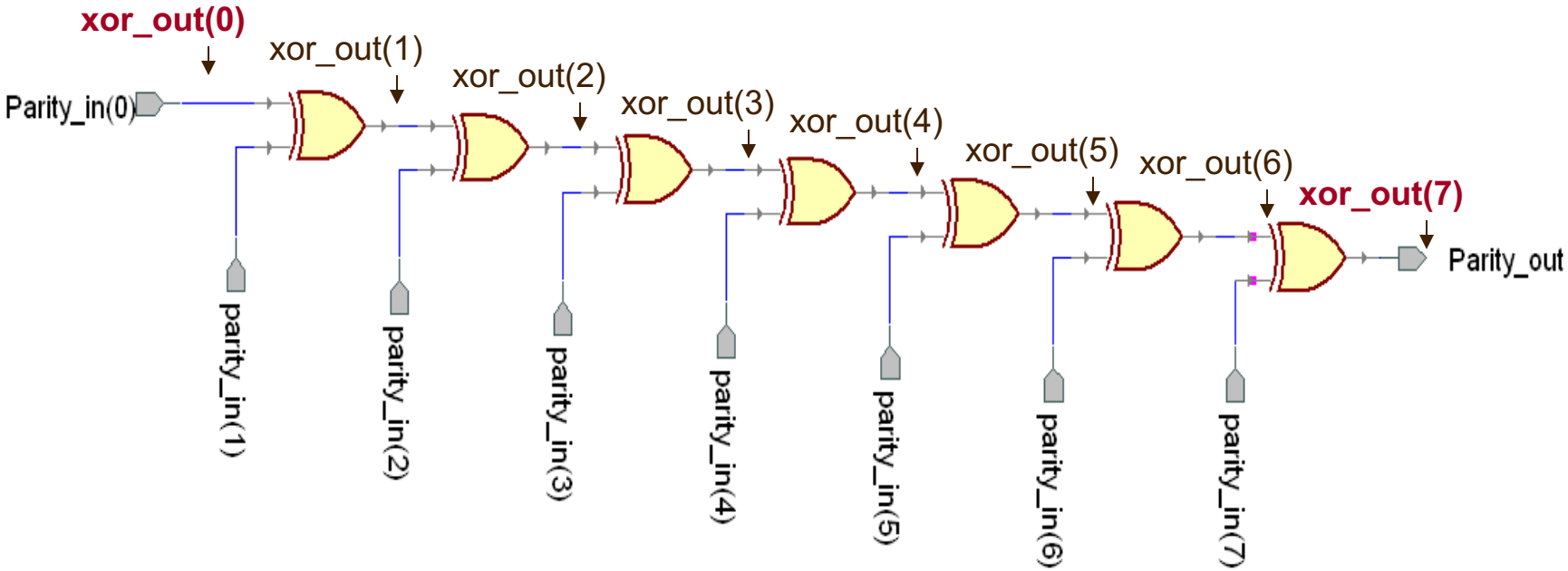
A'left(1)	= 1
A'right(2)	= 0
A'low(1)	= 1
A'high(2)	= 31
A'range(1)	= 1 **to** 4
A'length(2)	= 32
A'ascending(2)	= false

Unconstrained PARITY Generator (1)

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

ENTITY parity IS
     PORT(
          parity_in    : IN STD_LOGIC_VECTOR;
          parity_out   : OUT STD_LOGIC
        );
END parity;
```

PARITY: Block Diagram (2)



Unconstrained PARITY Generator (2)

```
ARCHITECTURE parity_dataflow OF parity IS
CONSTANT  width: natural := parity_in'length;
SIGNAL xor_out: STD_LOGIC_VECTOR (width-1 DOWNTO 0);

BEGIN

    xor_out(0) <= parity_in(0);

    G2: FOR i IN 1 TO width-1 GENERATE
            xor_out(i) <= xor_out(i-1) XOR parity_in(i);
    END GENERATE G2;

    parity_out <= xor_out(width-1);

END parity_dataflow;
```

Unconstrained PARITY Generator (3)

Will the previous code work for the following types of signal parity_in?

std_logic_vector(7 downto 0);
std_logic_vector(0 to 7);
std_logic_vector(15 downto 8);
std_logic_vector(8 to 15);

Unconstrained PARITY Generator (3)

```
ARCHITECTURE parity_dataflow OF parity IS
SIGNAL xor_out: STD_LOGIC_VECTOR (parity_in'range);

BEGIN

    xor_out(0) <= parity_in(parity_in'low);

    G2: FOR i IN parity_in'low+1 TO parity_in'high GENERATE
            xor_out(i) <= xor_out(i-1) XOR parity_in(i);
    END GENERATE G2;

    parity_out <= xor_out(parity_in'high);

END parity_dataflow;
```



Non-synthesizable VHDL

Delays

Delays are not synthesizable

Statements, such as

```
wait for  5 ns
a <= b after 10 ns
```

will not produce the required delay, and should not be used in the code intended for synthesis.

Initializations

Declarations of signals (and variables) with initialized values, such as

SIGNAL a : STD_LOGIC **:= '0';**

cannot be synthesized, and thus should be avoided.

If present, they will be ignored by the synthesis tools.

Use set and reset signals instead.

Dual-edge triggered register/counter (1)

In FPGAs register/counter can change only at either rising (default) or falling edge of the clock.

Dual-edge triggered clock is not synthesizable correctly, using either of the descriptions provided below.

Dual-edge triggered register/counter (2)

```
PROCESS (clk)
BEGIN
   IF (clk'EVENT AND clk='1' ) THEN
         counter <= counter + 1;
   ELSIF (clk'EVENT AND clk='0' ) THEN
         counter <= counter + 1;
   END IF;
END PROCESS;
```

Dual-edge triggered register/counter (3)

```
PROCESS (clk)
BEGIN
    IF (clk'EVENT) THEN
        counter <= counter + 1;
    END IF;
END PROCESS;
```

```
PROCESS (clk)
BEGIN
        counter <= counter + 1;
END PROCESS;
```