ECE 545
Lecture 6

Behavioral Modeling of Sequential-Circuit Building Blocks

Mixing Design Styles

Modeling of Circuits with a Regular Structure

Required reading

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

  Chapter 5.1, VHDL Process

  Chapter 8, Sequential Circuit Design:
  Principle (except subchapter 8.6)

  Slides for Chapter 8, available at
  http://academic.csuohio.edu/chu_p/rtl/rtl_hardware.html

Required reading

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

Chapter 14.5 For generate statement

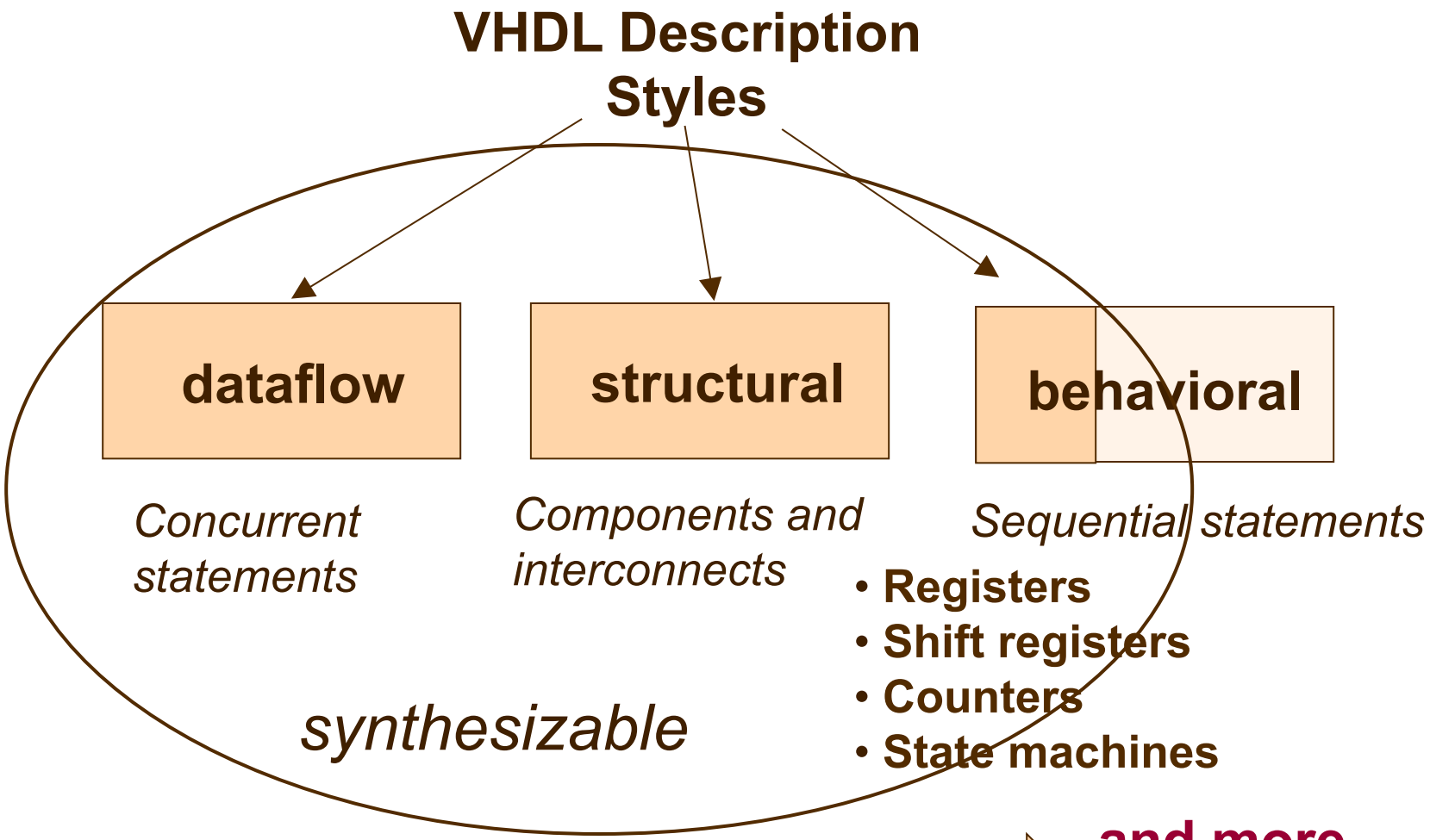
Chapter 14.6 Conditional generate statement

Chapter 9.1, Poor design practices and their remedies



Behavioral Design Style: Registers & Counters

VHDL Description Styles

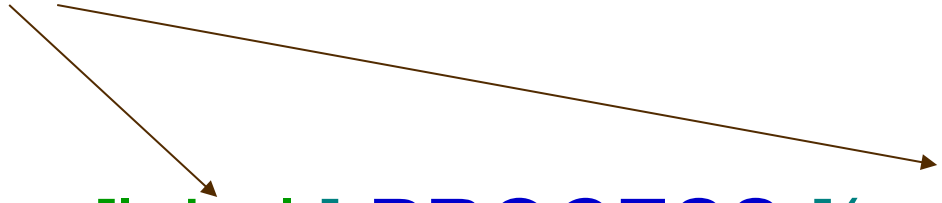


Processes in VHDL

- Processes Describe Sequential Behavior
- Processes in VHDL Are Very Powerful Statements
  - Allow to define an arbitrary behavior that may be difficult to represent by a real circuit
  - Not every process can be synthesized
- Use Processes with Caution in the Code to Be Synthesized
- Use Processes Freely in Testbenches

Anatomy of a Process

OPTIONAL



[label:] PROCESS [(*sensitivity list*)]
    [d*eclaration part*]
BEGIN
    statement part
END PROCESS [label];

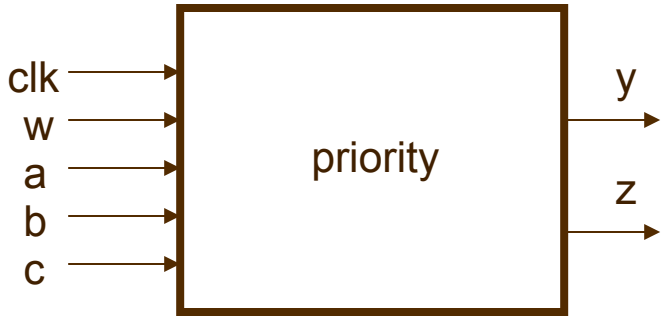
PROCESS with a SENSITIVITY LIST

- List of signals to which the process is sensitive.
- Whenever there is an event on any of the signals in the sensitivity list, the process fires.
- Every time the process fires, it will run in its entirety.
- **WAIT statements are NOT ALLOWED in a processes with SENSITIVITY LIST.**

label: process (*sensitivity list*)
    declaration part
begin
    statement part
end process;

Component Equivalent of a Process

```
priority: PROCESS (clk)
BEGIN
        IF w(3) = '1' THEN
                y <= "11" ;
        ELSIF w(2) = '1' THEN
                y <= "10" ;
        ELSIF w(1) = c THEN
                y <= a and b;
        ELSE
                z <= "00" ;
        END IF ;
END PROCESS ;
```



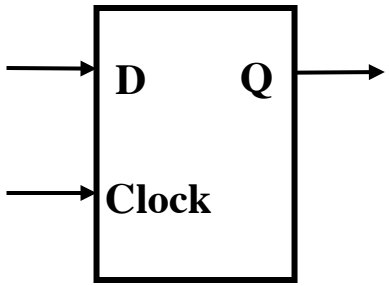
- All signals which appear on the sensitivity list are inputs e.g. *clk*
- All signals which appear on the left of signal assignment statement (<=) are outputs e.g. *y, z*
- Note that not all inputs need to be included on the sensitivity list
- All signals which appear on the right of signal assignment statement (<=) or in logic expressions are inputs e.g. *w, a, b, c*


High Performance
Low Power
CoolClock
XILINX
CoolRunner-II

Registers

D latch

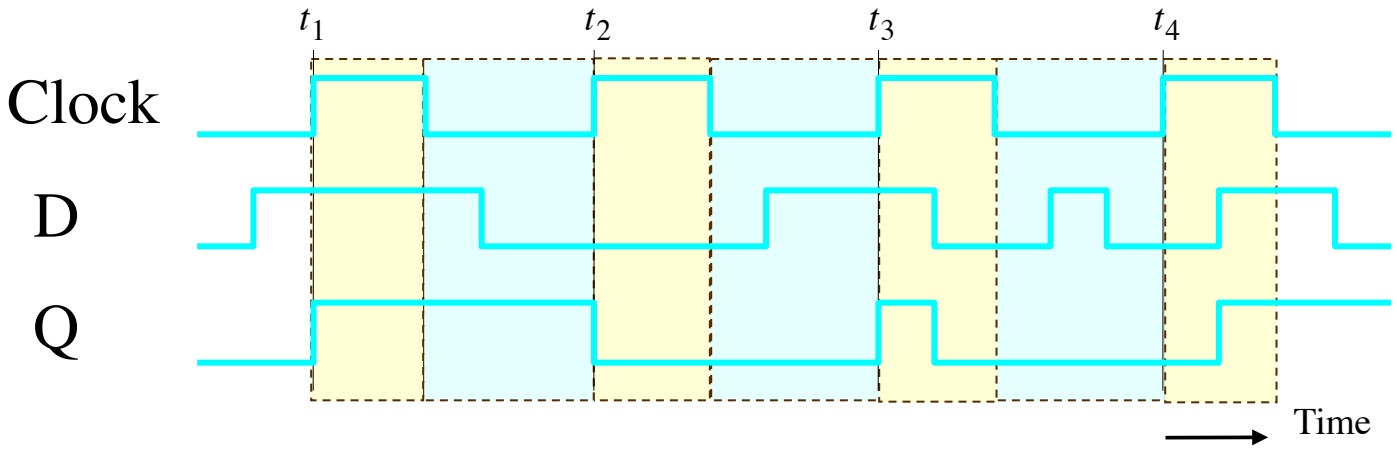
Graphical symbol



Truth table

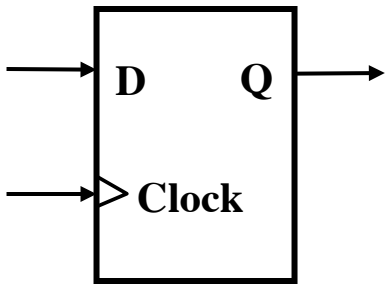
Clock	D	Q(*t*+1)
0	–	Q(*t*)
1	0	0
1	1	1

Timing diagram



D flip-flop

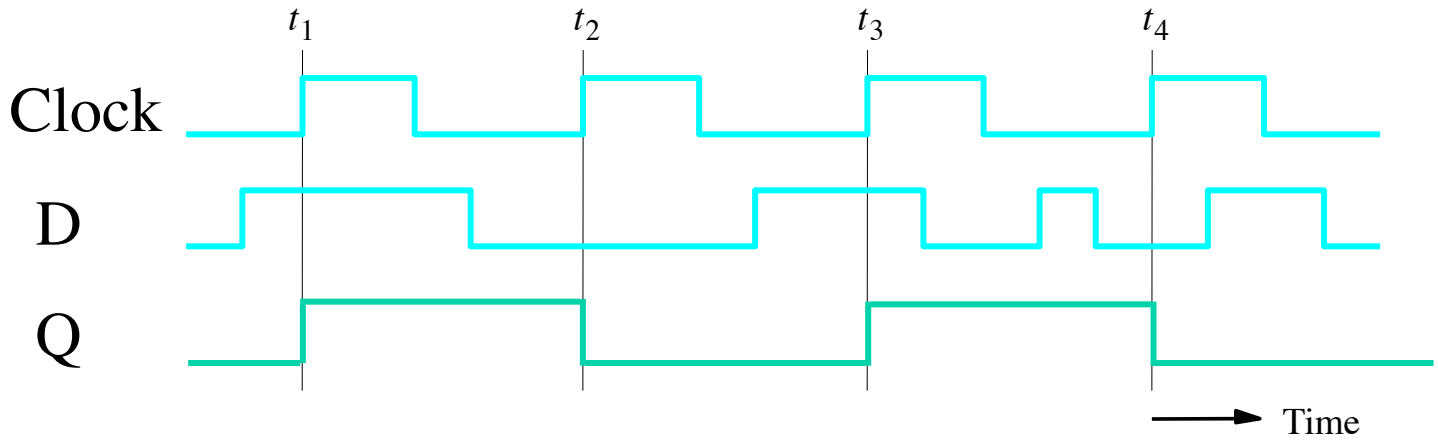
Graphical symbol



Truth table

Clk	D	Q(*t*+1)
↑	0	0
↑	1	1
0	–	Q(*t*)
1	–	Q(*t*)

Timing diagram

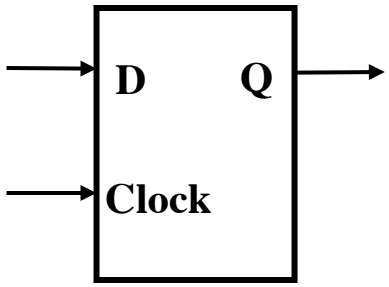


D latch

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

ENTITY latch IS
        PORT (    D, Clock  : IN        STD_LOGIC ;
                  Q         : OUT       STD_LOGIC) ;
END latch ;

ARCHITECTURE behavioral OF latch IS
BEGIN
        PROCESS ( D, Clock )
        BEGIN
                IF Clock = '1' THEN
                        Q <= D ;
                END IF ;
        END PROCESS ;
END behavioral;
```

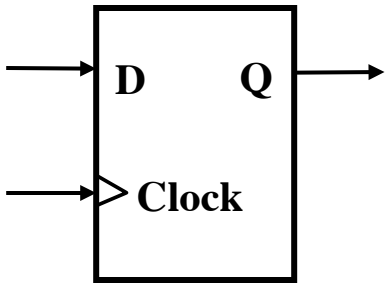


D flip-flop

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

ENTITY flipflop IS
      PORT (   D, Clock  : IN      STD_LOGIC ;
               Q         : OUT   STD_LOGIC) ;
END flipflop ;

ARCHITECTURE behavioral2 OF flipflop IS
BEGIN
      PROCESS ( Clock )
      BEGIN
           IF rising_edge(Clock) THEN
                Q <= D ;
           END IF ;
      END PROCESS ;
END behavioral2;
```

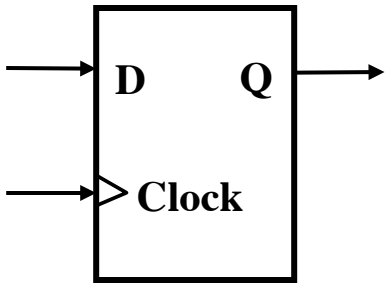


D flip-flop

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

ENTITY flipflop IS
        PORT (    D, Clock  : IN      STD_LOGIC ;
                  Q         : OUT     STD_LOGIC) ;
END flipflop ;

ARCHITECTURE behavioral OF flipflop IS
BEGIN
        PROCESS ( Clock )
        BEGIN
            IF Clock'EVENT AND Clock = '1' THEN
                Q <= D ;
            END IF ;
        END PROCESS ;
END behavioral ;
```

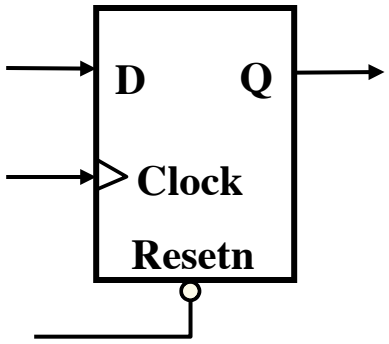


D flip-flop with asynchronous reset

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

ENTITY flipflop_ar IS
      PORT (   D, Resetn, Clock    : IN       STD_LOGIC ;
               Q                   : OUT      STD_LOGIC) ;
END flipflop_ar ;

ARCHITECTURE behavioral OF flipflop_ar IS
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn = '0' THEN
                  Q <= '0' ;
            ELSIF rising_edge(Clock) THEN
                  Q <= D ;
            END IF ;
      END PROCESS ;
END behavioral ;
```

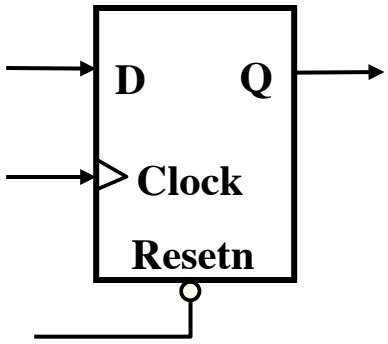


D flip-flop with synchronous reset

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY flipflop_sr IS
        PORT (      D, Resetn, Clock        : IN        STD_LOGIC ;
                    Q                       : OUT       STD_LOGIC) ;
END flipflop_sr ;

ARCHITECTURE behavioral OF flipflop_sr IS
BEGIN
        PROCESS(Clock)
        BEGIN
                IF rising_edge(Clock) THEN
                        IF Resetn = '0' THEN
                                Q <= '0' ;
                        ELSE
                                Q <= D ;
                        END IF ;
                END IF;
        END PROCESS ;

END behavioral ;
```



Asychronous vs. Synchronous

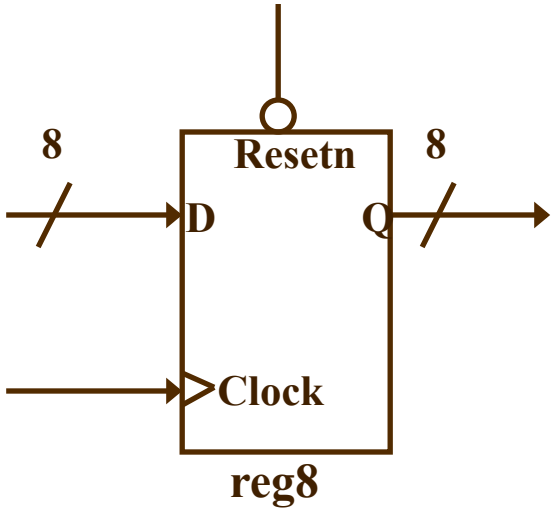
- In the IF loop, asynchronous items are
  - **Before** the rising_edge(Clock) statement
- In the IF loop, synchronous items are
  - **After** the rising_edge(Clock) statement

8-bit register with asynchronous reset

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

ENTITY reg8 IS
      PORT (   D                  : IN     STD_LOGIC_VECTOR(7 DOWNTO 0) ;
               Resetn, Clock      : IN     STD_LOGIC ;
               Q                  : OUT    STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END reg8 ;

ARCHITECTURE behavioral OF reg8 IS
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn = '0' THEN
                  Q <= "00000000" ;
            ELSIF rising_edge(Clock) THEN
                  Q <= D ;
            END IF ;
      END PROCESS ;
END behavioral ;`
```

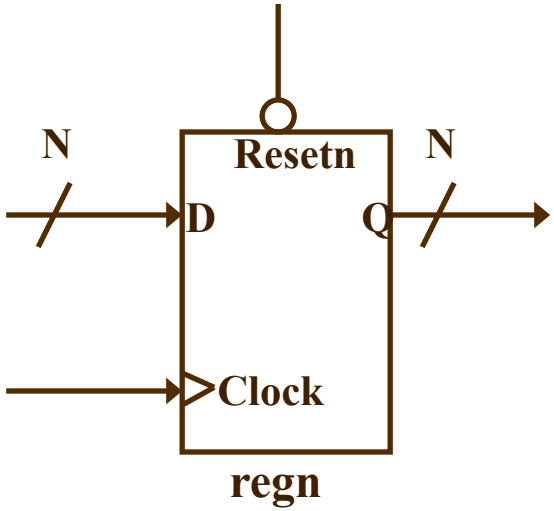


N-bit register with asynchronous reset

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

ENTITY regn IS
        GENERIC ( N : INTEGER := 16 ) ;
        PORT (    D              : IN     STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                  Resetn, Clock  : IN     STD_LOGIC ;
                  Q              : OUT    STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE behavioral OF regn IS
BEGIN
        PROCESS ( Resetn, Clock )
        BEGIN
                IF Resetn = '0' THEN
                        Q <= (OTHERS => '0') ;
                ELSIF rising_edge(Clock) THEN
                        Q <= D ;
                END IF ;
        END PROCESS ;
END behavioral ;
```



A word on generics

- Generics are typically **integer** values
  - In this class, the entity inputs and outputs should be std_logic or std_logic_vector
  - But the generics can be **integer**
- Generics are given a default value
  - **GENERIC (N : INTEGER := 16)** ;
  - This value can be overwritten when entity is instantiated as a component
- Generics are very useful when instantiating an often-used component
  - Need a 32-bit register in one place, and 16-bit register in another
  - Can use the same generic code, just configure them differently

Use of OTHERS

OTHERS stand for any index value that has not been previously mentioned.

Q <= “00000001” can be written as Q <= (0 => ‘1’, OTHERS => ‘0’)

Q <= “10000001” can be written as Q <= (7 => ‘1’, 0 => ‘1’, OTHERS => ‘0’)

or Q <= (7 | 0 => ‘1’, OTHERS => ‘0’)

Q <= “00011110” can be written as Q <= (4 downto 1=> ‘1’, OTHERS => ‘0’)

Component Instantiation in VHDL-93

```
U1:  ENTITY work.regn(behavioral)
                GENERIC MAP (N => 4)
                PORT MAP (D => z ,
                                Resetn => reset ,
                                Clock => clk,
                                Q => t );
```

Component Instantiation in VHDL-87

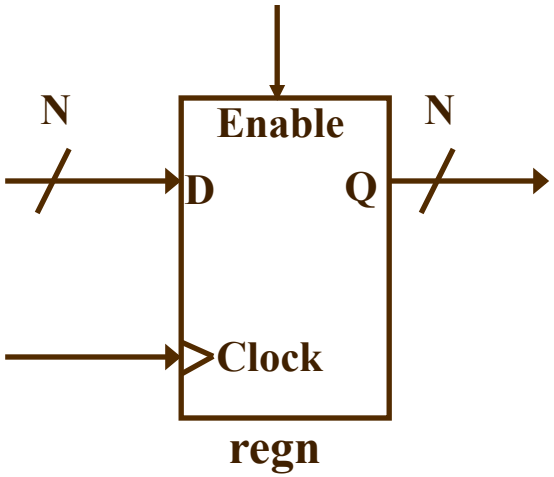
```
U1: regn        GENERIC MAP (N => 4)
                PORT MAP (D => z ,
                          Resetn => reset ,
                          Clock => clk,
                          Q => t );
```

N-bit register with enable

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

ENTITY regne IS
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (      D                   : IN        STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                    Enable, Clock       : IN        STD_LOGIC ;
                    Q                   : OUT       STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regne ;

ARCHITECTURE behavioral OF regne IS
BEGIN
        PROCESS (Clock)
        BEGIN
                IF rising_edge(Clock) THEN
                        IF Enable = '1' THEN
                                Q <= D ;
                        END IF ;
                END IF;
        END PROCESS ;
END behavioral ;
```



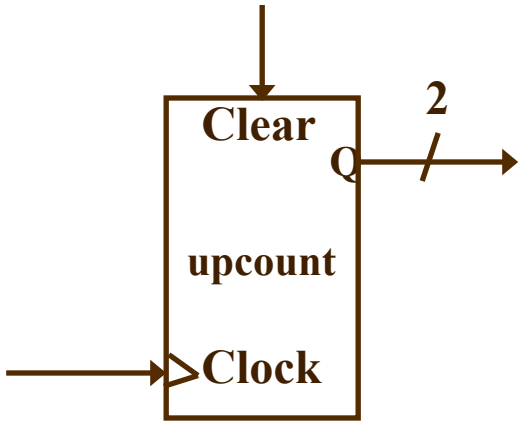


Counters

2-bit up-counter with synchronous reset

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
        PORT (      Clear, Clock      : IN        STD_LOGIC ;
                    Q                 : OUT       STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END upcount ;

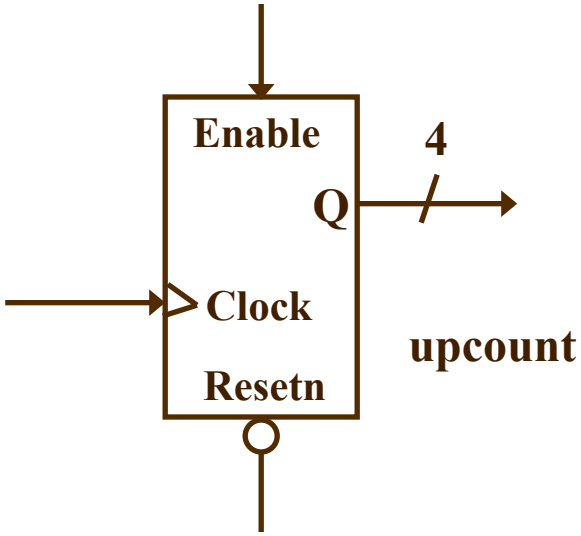
ARCHITECTURE behavioral OF upcount IS
   SIGNAL Count : std_logic_vector(1 DOWNTO 0);
BEGIN
        upcount: PROCESS ( Clock )
        BEGIN
                IF rising_edge(Clock) THEN
                        IF Clear = '1' THEN
                                Count <= "00" ;
                        ELSE
                                Count <= Count + 1 ;
                        END IF ;
                END IF;
        END PROCESS;
       Q <= Count;
END behavioral;
```



4-bit up-counter with asynchronous reset (1)

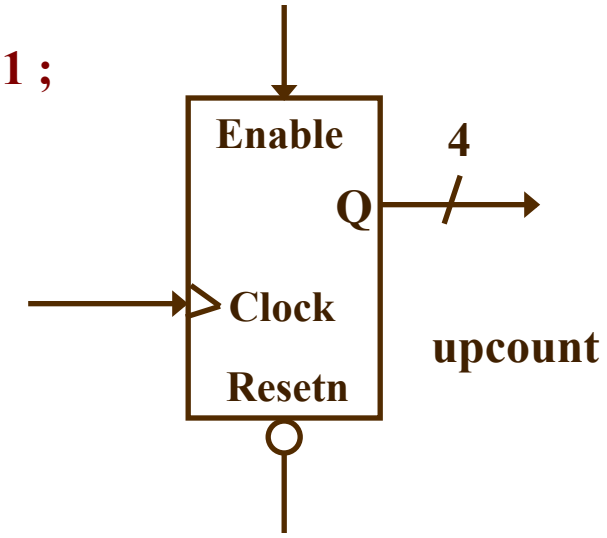
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY upcount_ar IS
      PORT (   Clock, Resetn, Enable   : IN       STD_LOGIC ;
               Q                       : OUT  STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount_ar ;
```



4-bit up-counter with asynchronous reset (2)

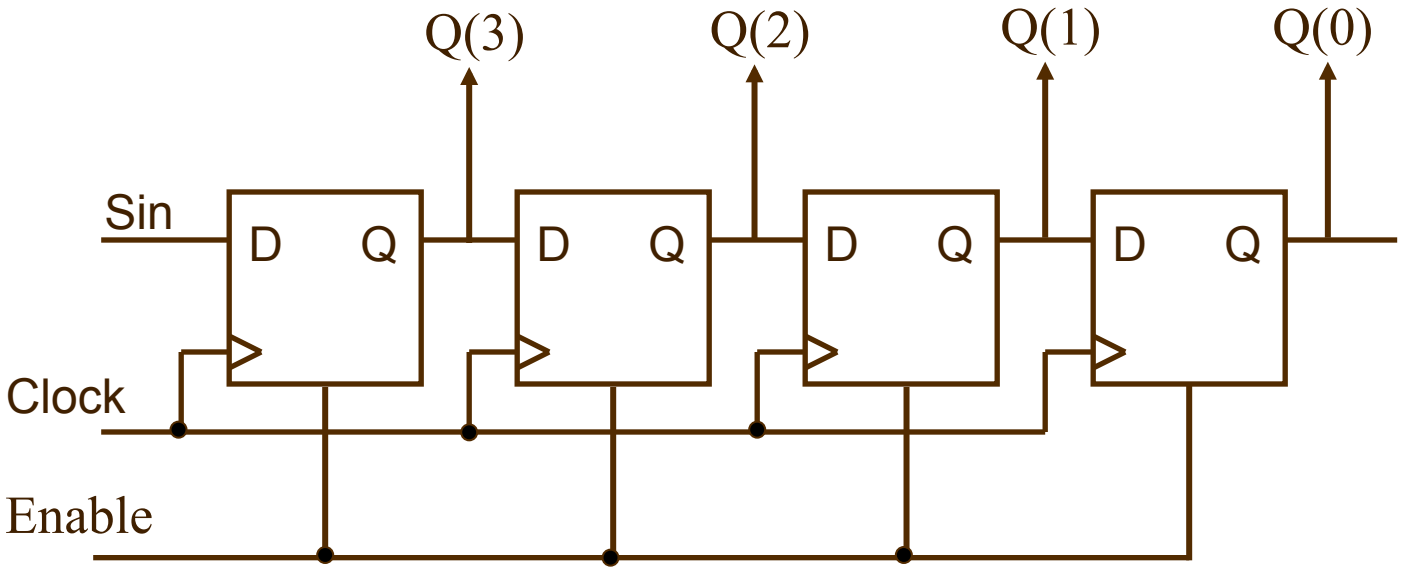
```
ARCHITECTURE behavioral OF upcount _ar IS
        SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
        PROCESS ( Clock, Resetn )
        BEGIN
                IF Resetn = '0' THEN
                        Count <= "0000" ;
                ELSIF rising_edge(Clock) THEN
                        IF Enable = '1' THEN
                                Count <= Count + 1 ;
                        END IF ;
                END IF ;
        END PROCESS ;
        Q <= Count ;
END behavioral ;
```



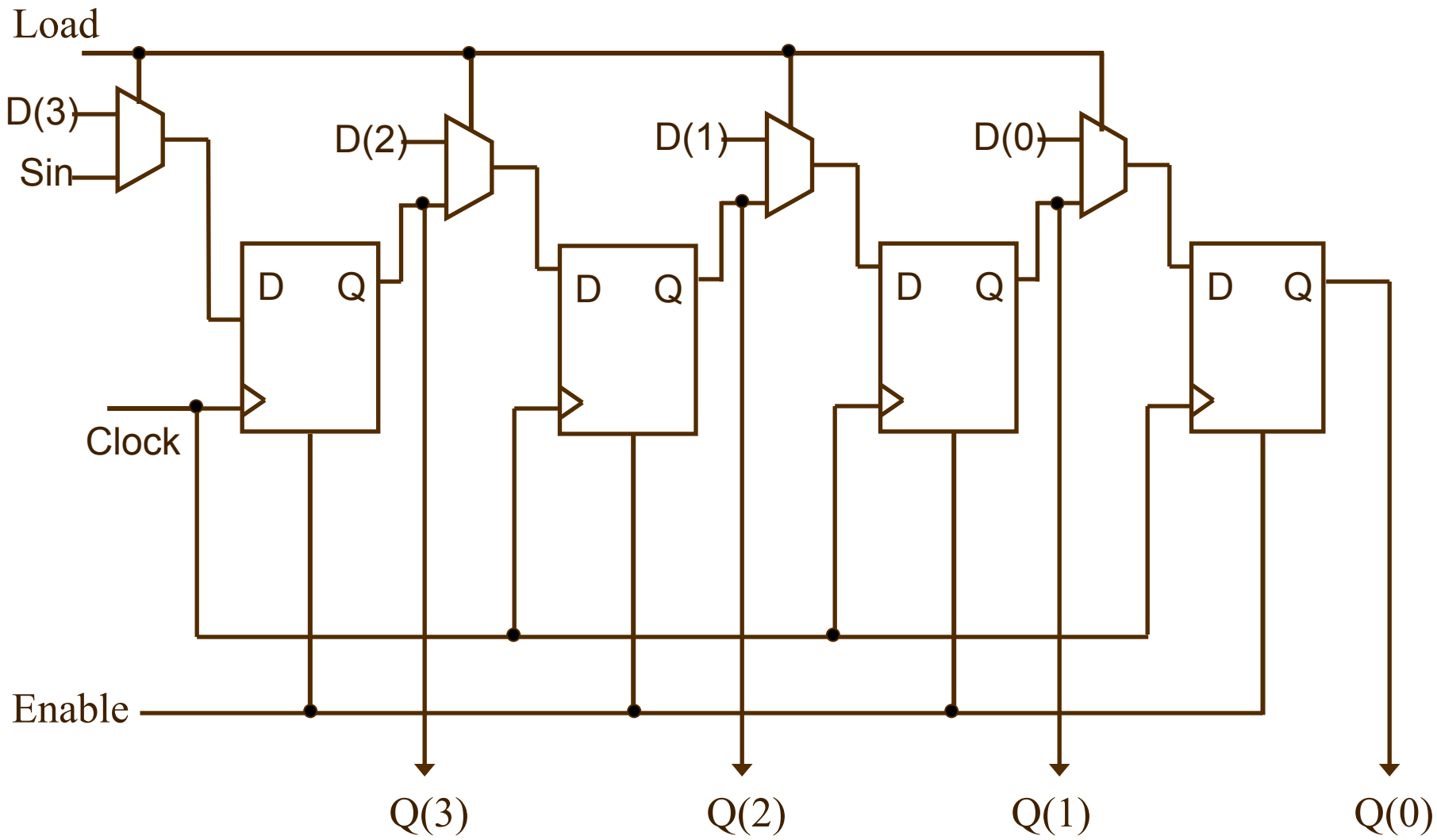


Shift Registers

Shift register – internal structure



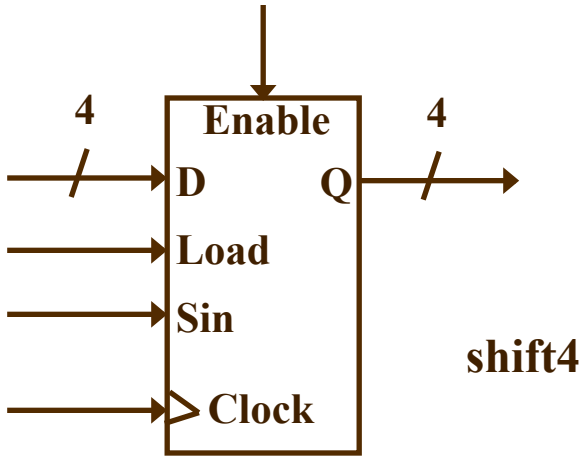
Shift Register With Parallel Load



4-bit shift register with parallel load (1)

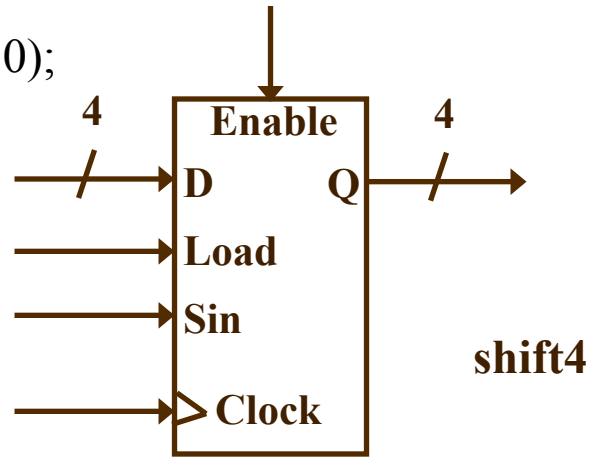
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY shift4 IS
      PORT (   D        : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               Enable   : IN     STD_LOGIC ;
               Load     : IN     STD_LOGIC ;
               Sin      : IN     STD_LOGIC ;
               Clock    : IN     STD_LOGIC ;
               Q        : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;
```



4-bit shift register with parallel load (2)

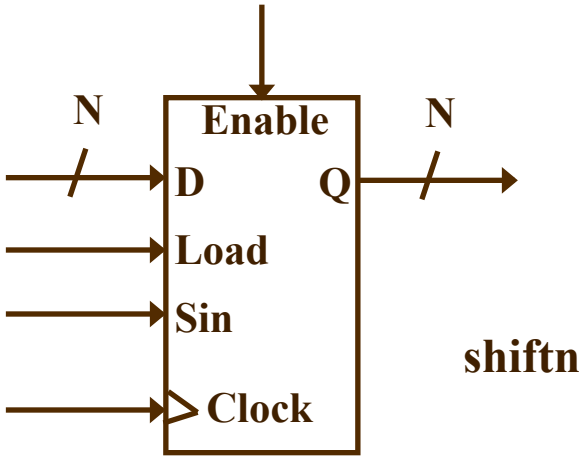
```
ARCHITECTURE behavioral OF shift4 IS
        SIGNAL Qt : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
        PROCESS (Clock)
        BEGIN
                IF rising_edge(Clock) THEN
                        IF Enable = '1' THEN
                                IF Load = '1' THEN
                                        Qt <= D ;
                                ELSE
                                        Qt <= Sin & Qt(3 downto 1);
                                END IF ;
                        END IF;
                END IF ;
        END PROCESS ;
        Q <= Qt;
END behavioral ;
```



N-bit shift register with parallel load (1)

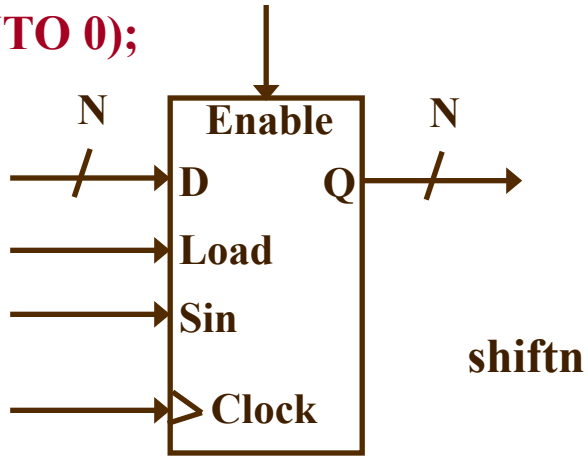
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

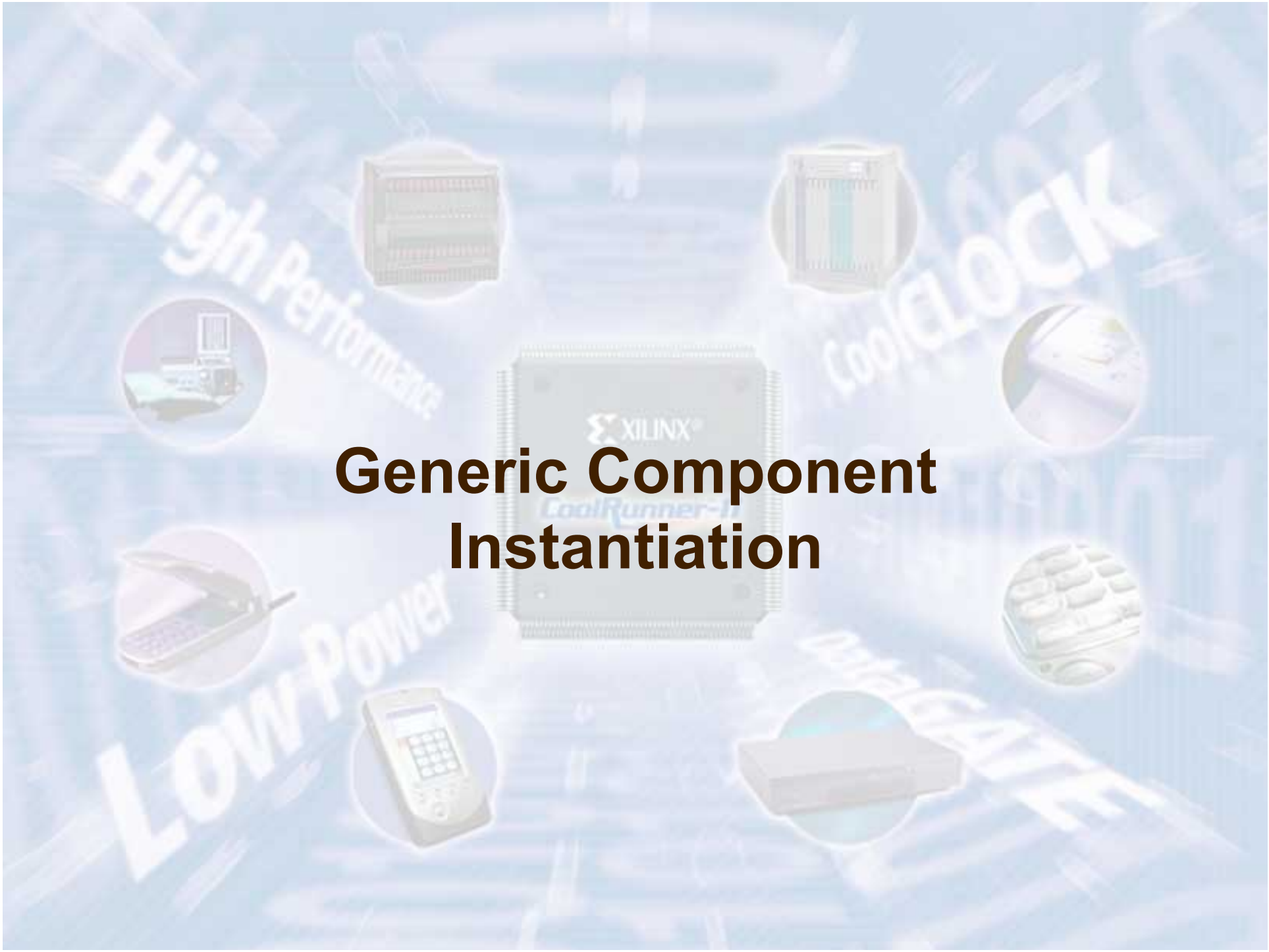
ENTITY shiftn IS
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (  D : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                Enable  : IN    STD_LOGIC ;
                Load    : IN    STD_LOGIC ;
                Sin     : IN    STD_LOGIC ;
                Clock   : IN    STD_LOGIC ;
                Q       : OUT   STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END shiftn ;
```



N-bit shift register with parallel load (2)

```
ARCHITECTURE behavioral OF shiftn IS
        SIGNAL Qt: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
BEGIN
        PROCESS (Clock)
        BEGIN
                IF rising_edge(Clock) THEN
                        IF Enable = '1' THEN
                                IF Load = '1' THEN
                                        Qt <= D ;
                                ELSE
                                        Qt <= Sin & Qt(N-1 downto 1);
                                END IF ;
                        END IF;
                END IF ;
        END PROCESS ;
        Q <= Qt;
END behavior al;
```





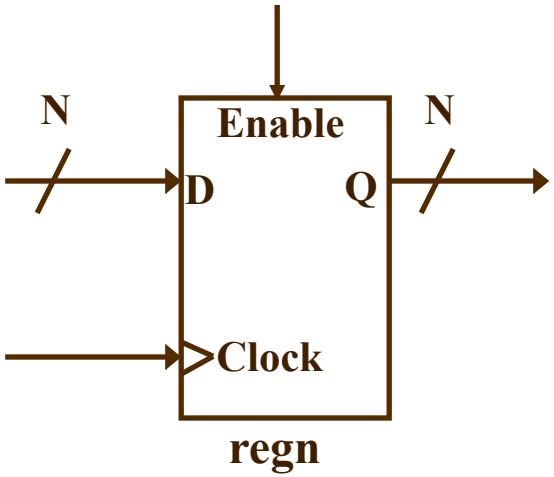
Generic Component Instantiation

N-bit register with enable

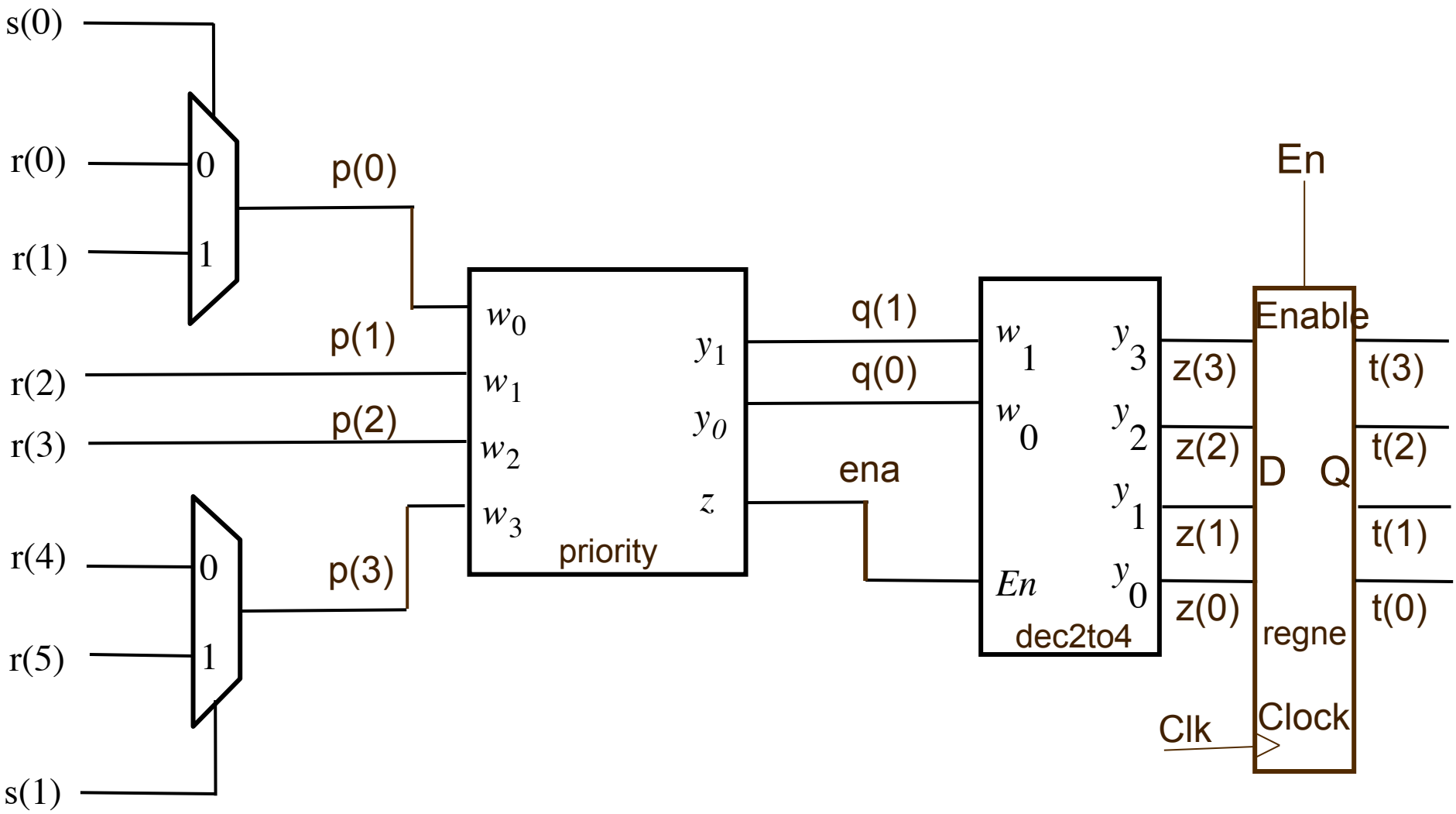
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY regn IS
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (      D                   : IN        STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                    Enable, Clock       : IN        STD_LOGIC ;
                    Q                   : OUT       STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE Behavior OF regn IS
BEGIN
        PROCESS (Clock)
        BEGIN
                IF (Clock'EVENT AND Clock = '1' ) THEN
                        IF Enable = '1' THEN
                                Q <= D ;
                        END IF ;
                END IF;
        END PROCESS ;
END Behavior ;
```



Circuit built of medium scale components



Structural description – example (1)
VHDL-93

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

ENTITY priority_resolver IS
    PORT (r        : IN     STD_LOGIC_VECTOR(5 DOWNTO 0) ;
          s        : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
          clk      : IN     STD_LOGIC;
          en       : IN     STD_LOGIC;
          t        : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END priority_resolver;

ARCHITECTURE structural OF priority_resolver IS

SIGNAL  p : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL  q : STD_LOGIC_VECTOR (1  DOWNTO 0) ;
SIGNAL  z : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL  ena : STD_LOGIC ;
```

Structural description – example (2) VHDL-93

```
BEGIN

    u1:  ENTITY work.mux2to1(dataflow)
                    PORT MAP (w0 => r(0) ,
                                    w1 => r(1),
                                    s => s(0),
                                    f => p(0));
    p(1) <= r(2);
    p(2) <= r(3);

    u2:  ENTITY work.mux2to1(dataflow)
                    PORT MAP (w0 => r(4) ,
                                    w1 => r(5),
                                    s => s(1),
                                    f => p(3));

    u3: ENTITY work.priority(dataflow)
                   PORT MAP (w => p,
                                   y => q,
                                   z => ena);
```

Structural description – example (3) VHDL-93

```
u4: ENTITY work.dec2to4 (dataflow)
                PORT MAP (w => q,
                          En => ena,
                           y => z);

u5: ENTITY work.regne(behavioral)
                GENERIC MAP (N => 4)
                 PORT MAP (D => z ,
                           Enable => En ,
                           Clock => Clk,
                           Q => t );
END structural;
```

Structural description – example (1)
VHDL-87

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

ENTITY priority_resolver IS
    PORT (r        : IN      STD_LOGIC_VECTOR(5 DOWNTO 0) ;
          s        : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
          clk      : IN      STD_LOGIC;
          en       : IN      STD_LOGIC;
          t        : OUT     STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END priority_resolver;

ARCHITECTURE structural OF priority_resolver IS

SIGNAL  p : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL  q : STD_LOGIC_VECTOR (1  DOWNTO 0) ;
SIGNAL  z : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL  ena : STD_LOGIC ;
```

Structural description – example (2)
VHDL-87

```
COMPONENT mux2to1
  PORT (w0, w1, s        : IN     STD_LOGIC ;
        f                : OUT    STD_LOGIC ) ;
END COMPONENT ;

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

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 ;
```

Structural description – example (3)
VHDL-87

```
COMPONENT regn
   GENERIC ( N : INTEGER := 8 ) ;
   PORT (          D     : IN          STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                   Enable, Clock     : IN                STD_LOGIC ;
                   Q     : OUT         STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END COMPONENT ;
```

Structural description – example (4)
VHDL-87

```
BEGIN

    u1: mux2to1 PORT MAP (w0 => r(0) ,
                          w1 => r(1),
                          s => s(0),
                          f => p(0));
    p(1) <= r(2);
    p(2) <= r(3);

    u2: mux2to1 PORT MAP (w0 => r(4) ,
                          w1 => r(5),
                          s => s(1),
                          f => p(3));

    u3: priority PORT MAP (w => p,
                           y => q,
                           z => ena);

    u4: dec2to4 PORT MAP (w => q,
                          En => ena,
                          y => z);
```

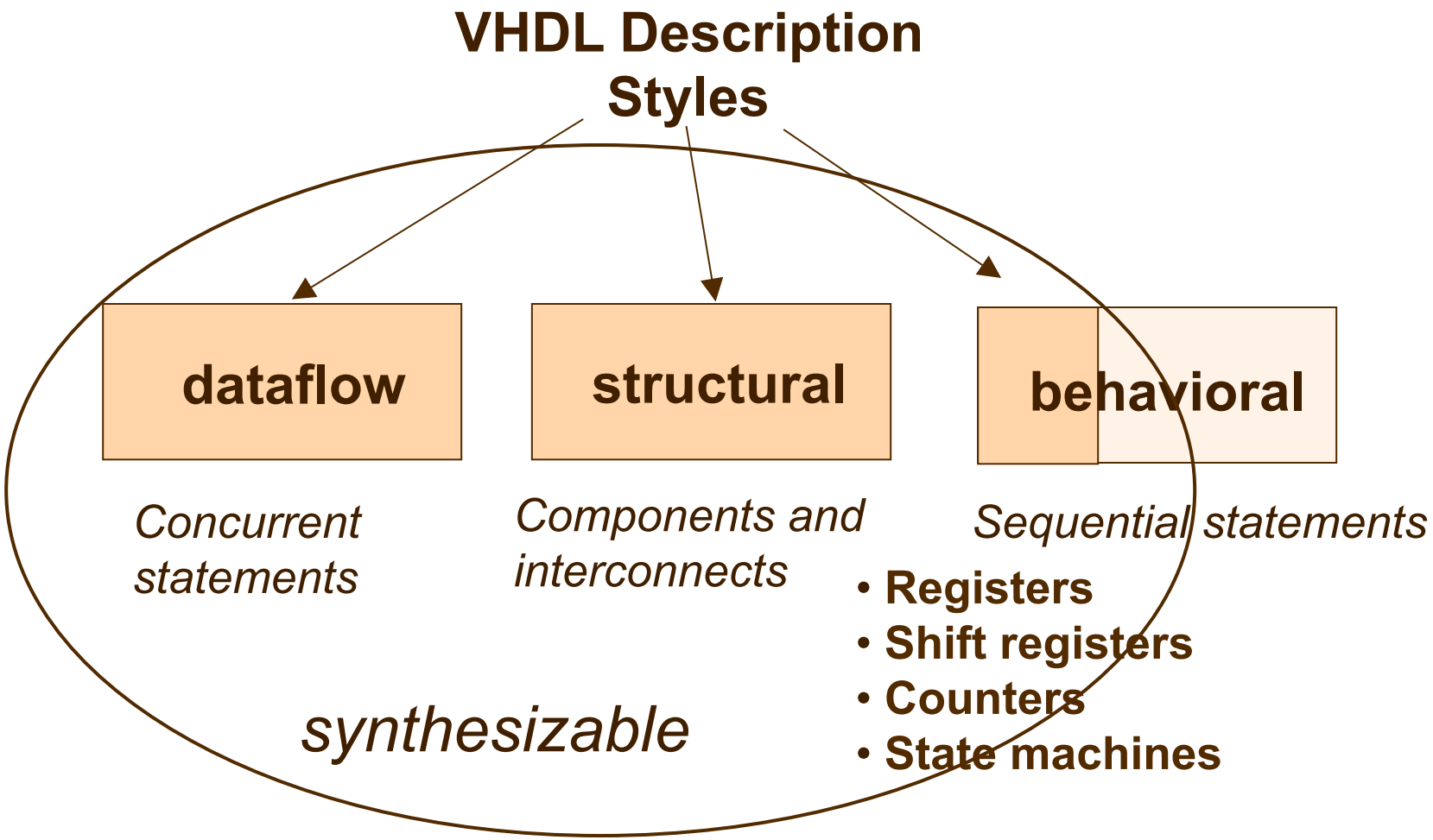
Structural description – example (5)
VHDL-87

```
u5: regne       GENERIC MAP (N => 4)
                PORT MAP (D => z ,
                          Enable => En ,
                          Clock => Clk,
                          Q => t );
END structural;
```



Mixing Description Styles Inside of an Architecture

VHDL Description Styles



Mixed Style Modeling

architecture ARCHITECTURE_NAME **of** ENTITY_NAME **is**

- Here you can declare signals, constants, functions, procedures…
- Component declarations

begin

Concurrent statements:

- Concurrent simple signal assignment
- Conditional signal assignment
- Selected signal assignment
- Generate statement
- Component instantiation statement
- Process statement
  - **inside process you can use only sequential statements**

end ARCHITECTURE_NAME;

Concurrent Statements

PRNG Example (1)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.prng_pkg.all;

ENTITY PRNG IS
    PORT( Coeff          : in  std_logic_vector(4 downto 0);
          Load_Coeff     : in  std_logic;
          Seed           : in  std_logic_vector(4 downto 0);
          Init_Run       : in  std_logic;
          Clk            : in  std_logic;
          Current_State  : out std_logic_vector(4 downto 0));
END PRNG;

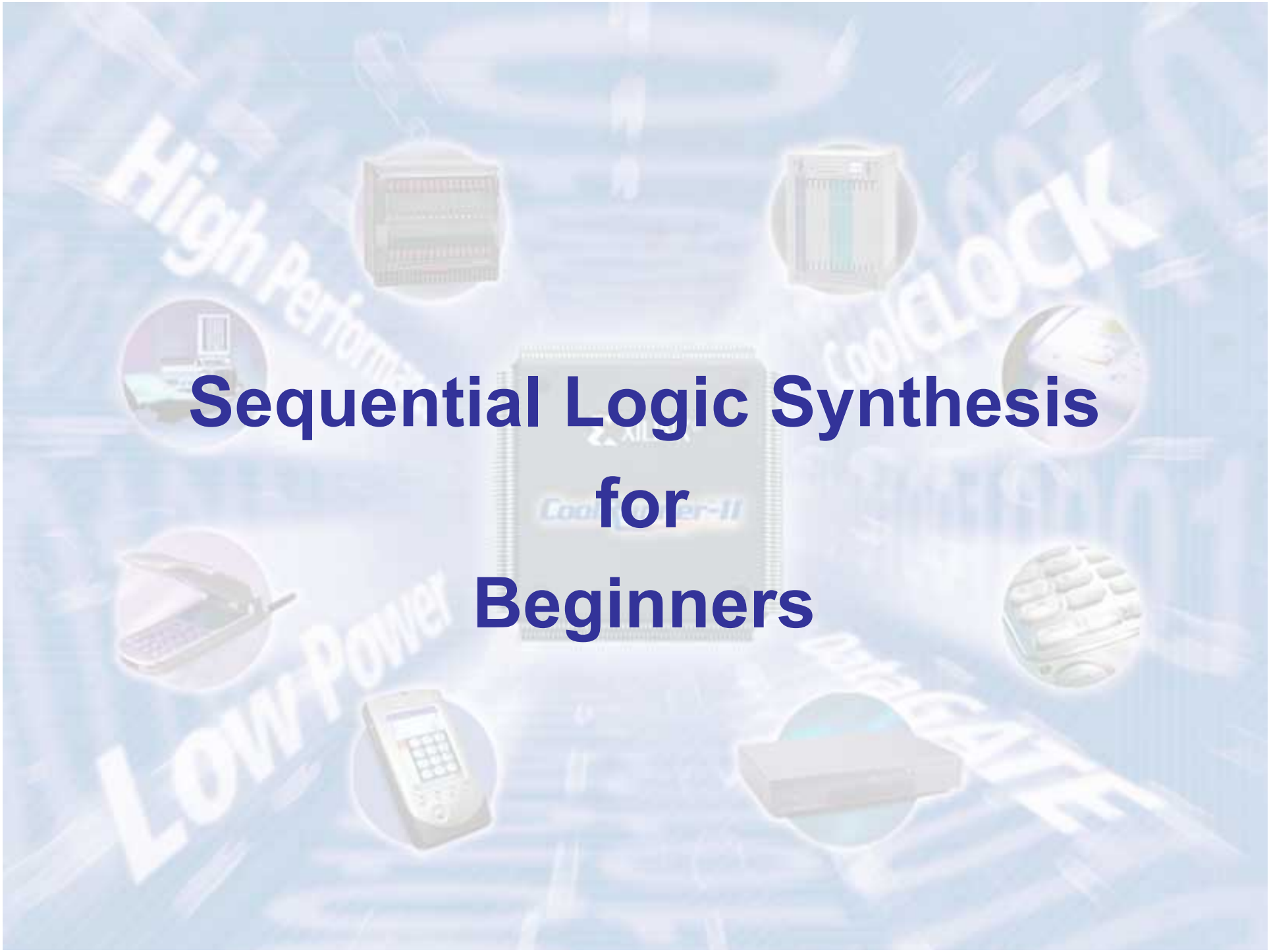
ARCHITECTURE mixed OF PRNG is
    signal Ands          : std_logic_vector(4 downto 0);
    signal Sin           : std_logic;
    signal Coeff_Q       : std_logic_vector(4 downto 0);
    signal Shift5_Q      : std_logic_vector(4 downto 0);
```

PRNG Example (2)

```
BEGIN
    -- Data Flow
    Sin <= Ands(0) XOR Ands(1) XOR Ands(2) XOR Ands(3) XOR Ands(4);
    Current_State <= Shift5_Q;
    Ands <= Coeff_Q AND Shift5_Q;

    -- Behavioral
    Coeff_Reg: PROCESS(Clk)
    BEGIN
        IF rising_edge(Clk)  THEN
            IF Load_Coeff = '1'   THEN
                Coeff_Q <= Coeff;
            END IF;
        END IF;
    END PROCESS;

    -- Structural
    Shift5_Reg : ENTITY work.Shift5(behavioral) PORT MAP ( D     => Seed,
                                                           Load  => Init_Run,
                                                           Sin   => Sin,
                                                           Clock => Clk,
                                                           Q     => Shift5_Q);
END mixed;
```



Sequential Logic Synthesis for Beginners

For Beginners

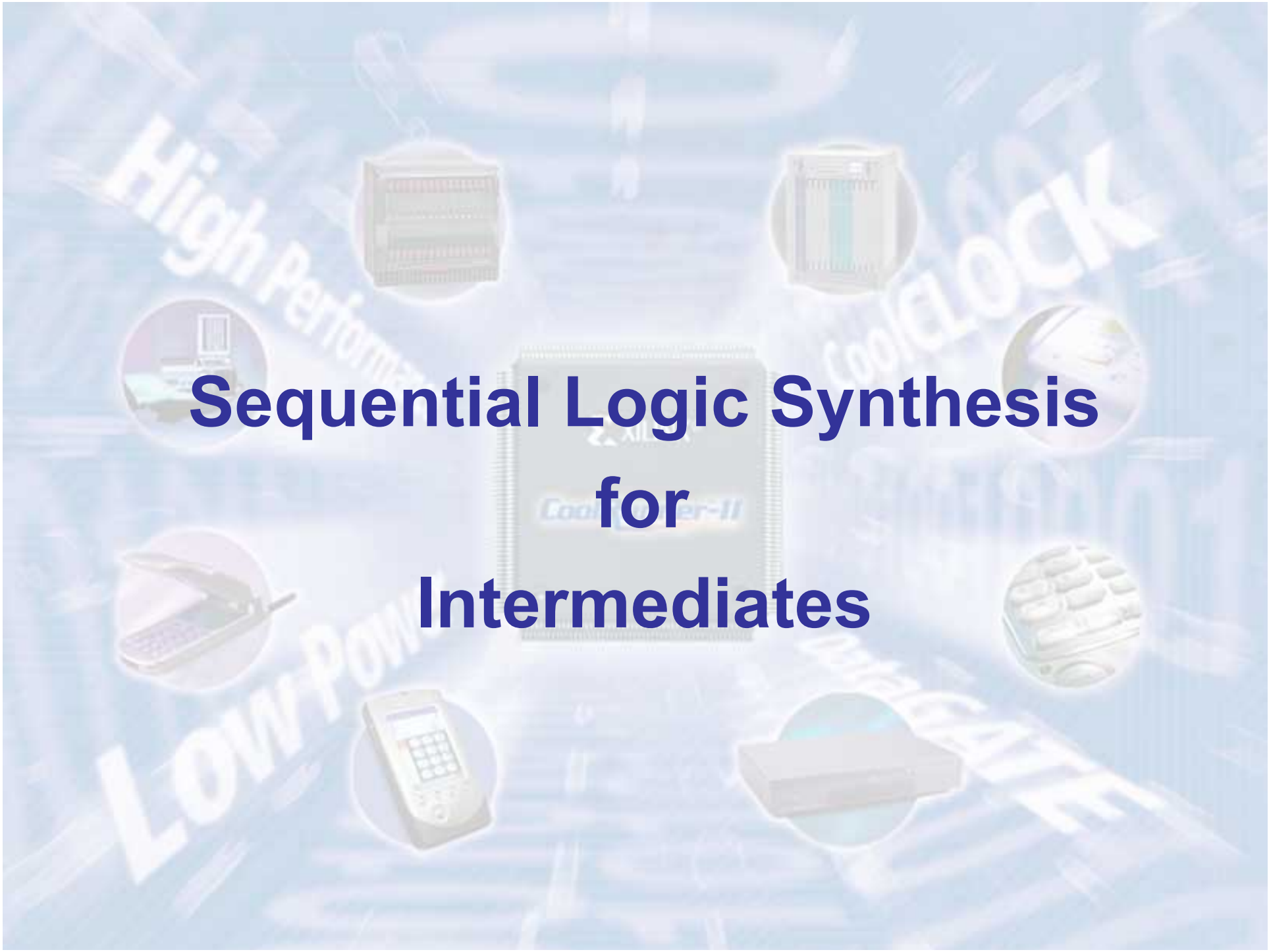
Use processes with very simple structure only to describe

- registers
- shift registers
- counters
- state machines.

Use examples discussed in class as a template.

Create **generic** entities for registers, shift registers, and counters, and instantiate the corresponding components in a higher level circuit using GENERIC MAP PORT MAP.

Supplement sequential components with combinational logic described using concurrent statements.



Sequential Logic Synthesis for Intermediates

For Intermmediates

1. Use Processes with IF and CASE statements only. Do not use LOOPS or VARIABLES.
2. Sensitivity list of the PROCESS should include **only** signals that can by themsleves change the outputs of the sequential circuit (typically, clock and asynchronous set or reset)
3. Do not use PROCESSes without sensitivity list

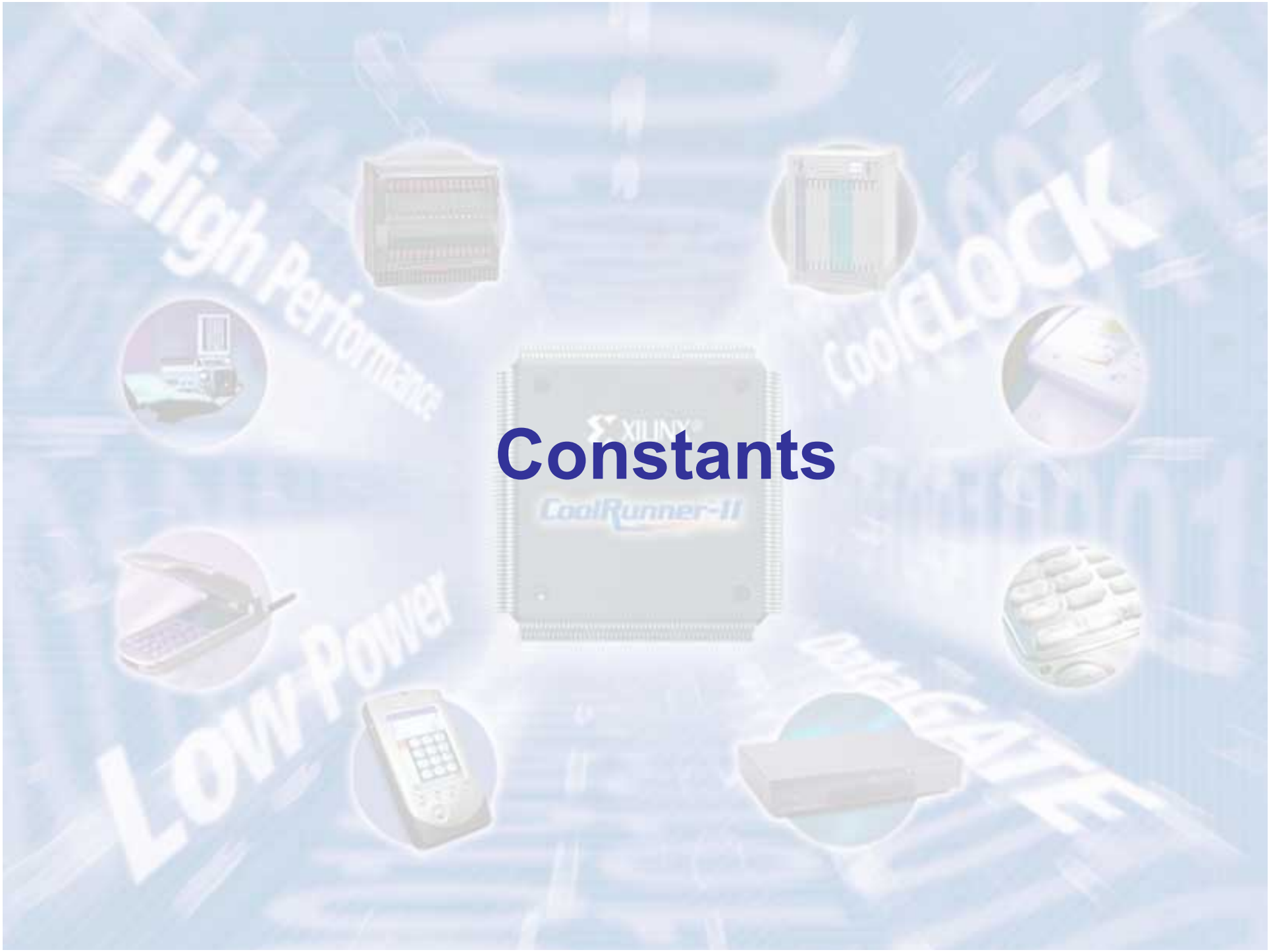
   (they can be synthesizable, but make simulation inefficient)

For Intermmediates (2)

Given a single signal, the assignments to this signal should only be made within a single process block in order to avoid possible conflicts in assigning values to this signal.

```
Process 1: PROCESS (a, b)
BEGIN
   y <= a AND b;
END PROCESS;

Process 2: PROCESS (a, b)
BEGIN
   y <= a OR b;
END PROCESS;
```



Constants

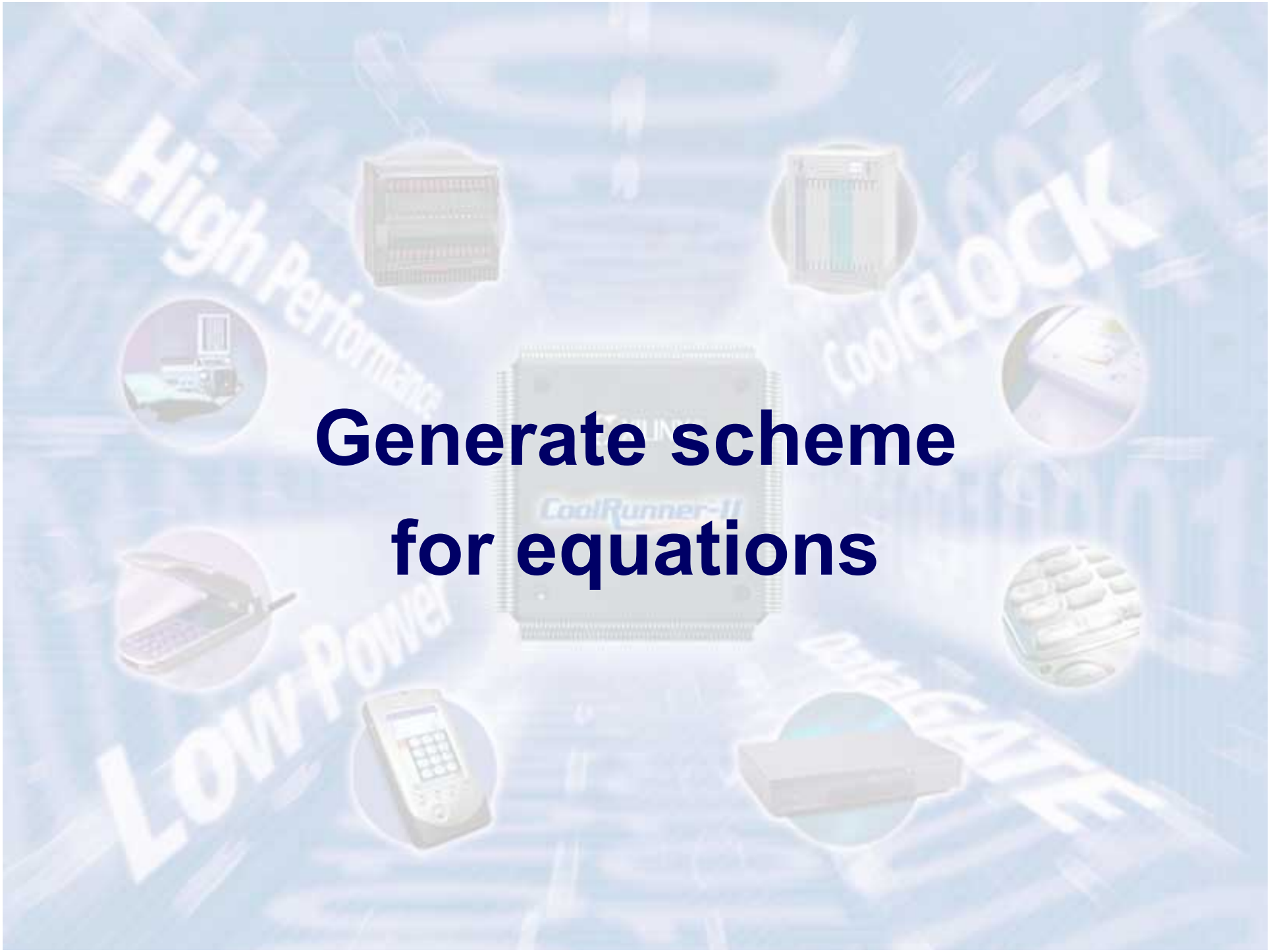
Constants

Syntax:

CONSTANT name : type := value;

Examples:

```
CONSTANT init_value : STD_LOGIC_VECTOR(3 downto 0) := "0100";
CONSTANT ANDA_EXT : STD_LOGIC_VECTOR(7 downto 0) := X"B4";
CONSTANT counter_width : INTEGER := 16;
CONSTANT buffer_address : INTEGER := 16#FFFE#;
CONSTANT clk_period : TIME := 20 ns;
CONSTANT strobe_period : TIME := 333.333 ms;
```



Generate scheme for equations

Dataflow VHDL

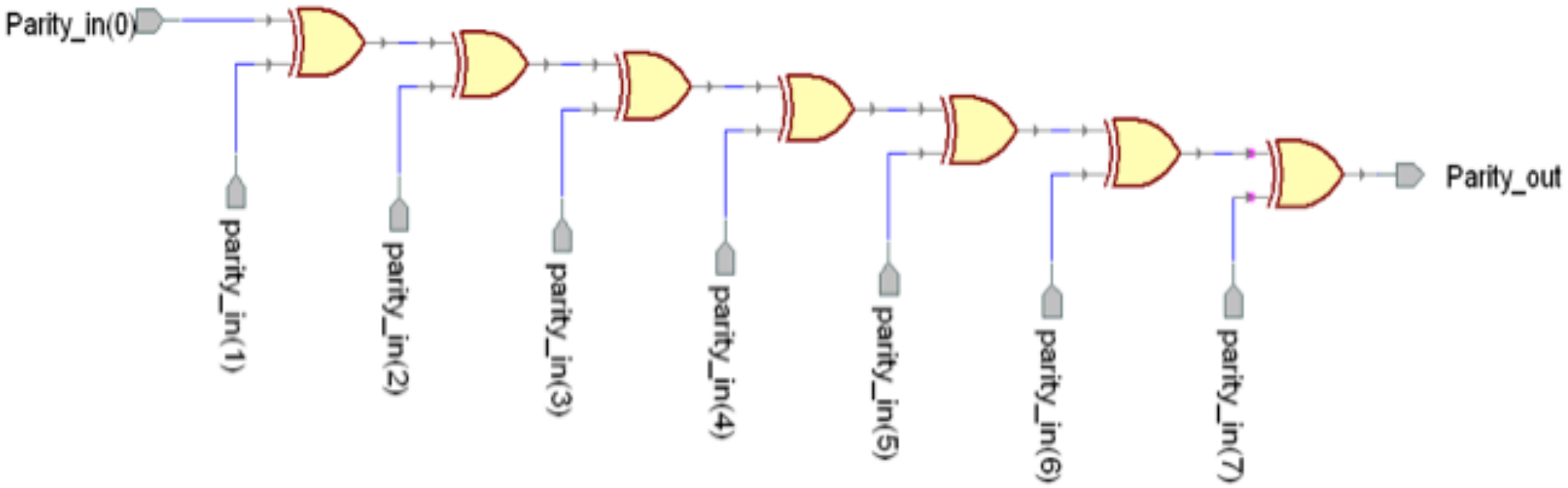
Major instructions

Concurrent statements

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

PARITY Example

PARITY: Block Diagram

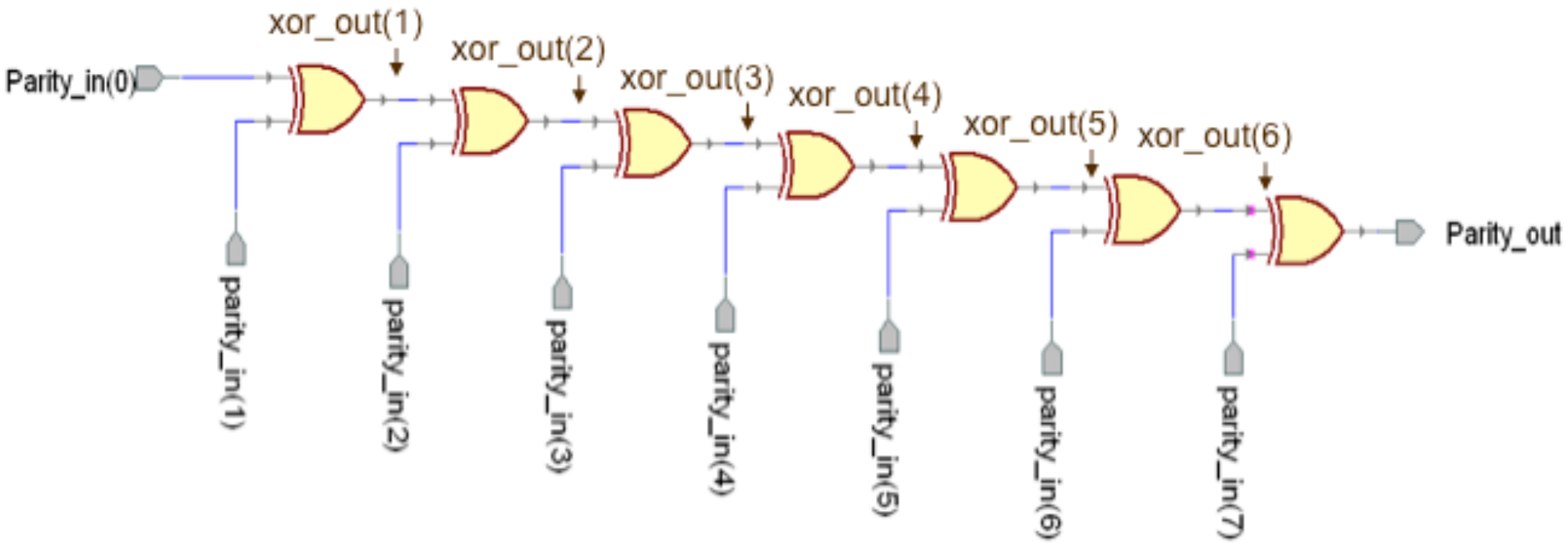


PARITY: Entity Declaration

```
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;
```

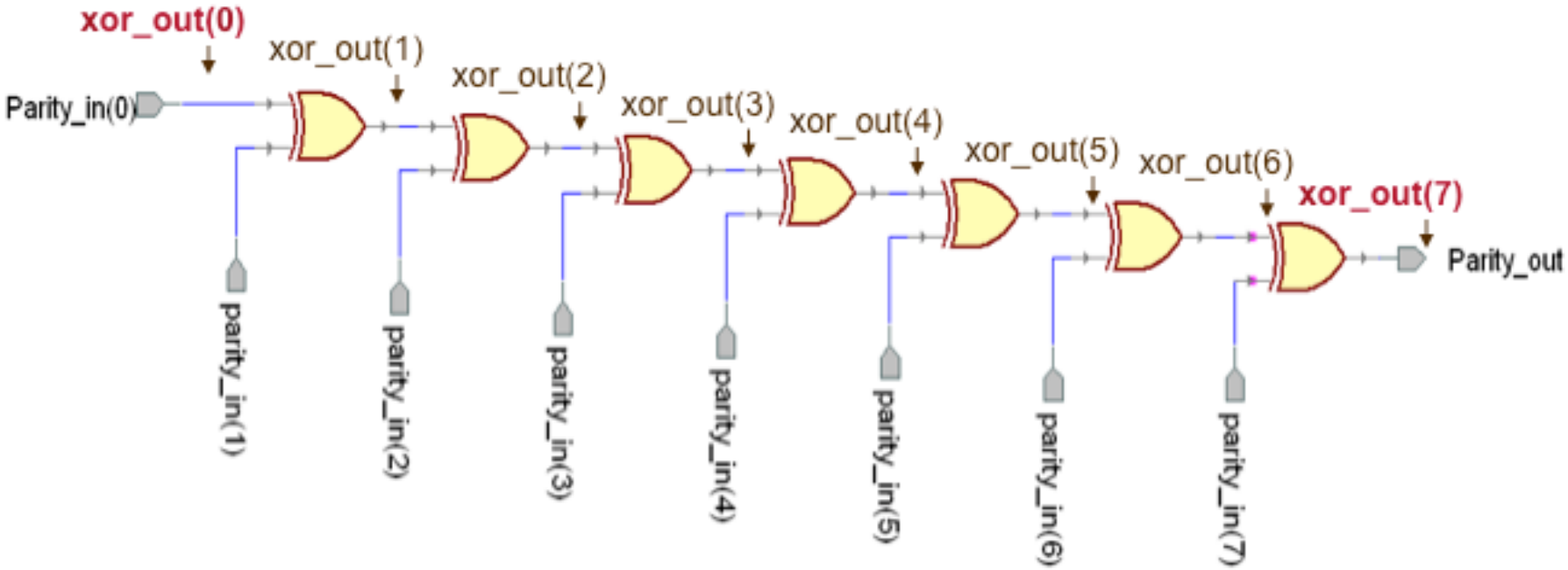
PARITY: Architecture (2)

```
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

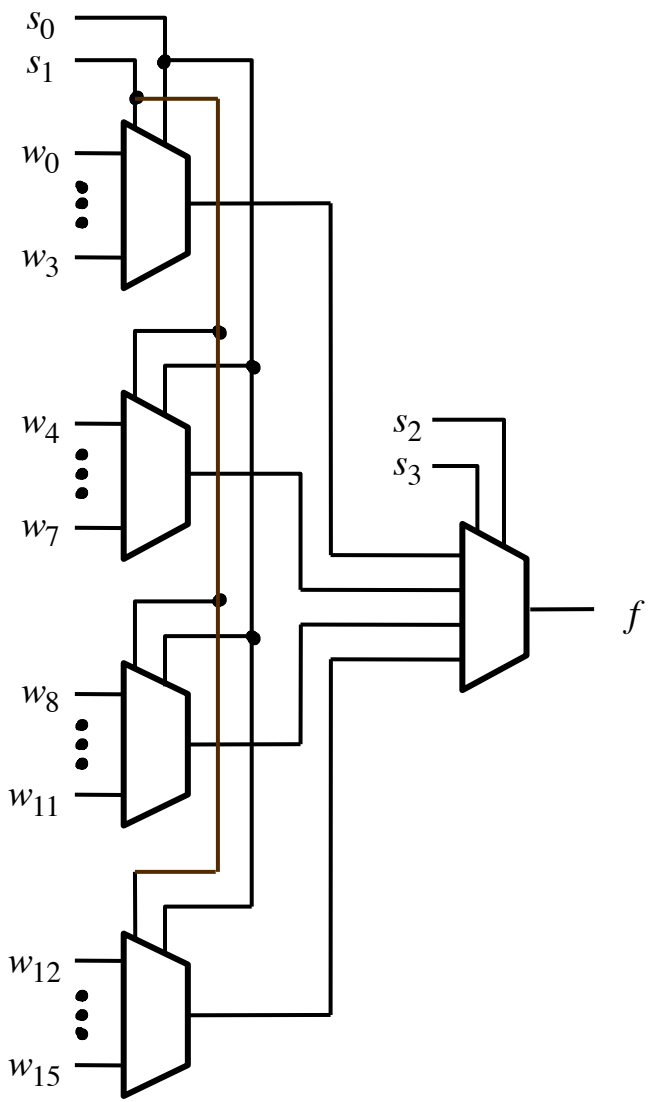
Structural VHDL

Major instructions

- component instantiation (port map)
- component instantiation with generic (generic map, port map)
- **generate scheme for component instantiations (for-generate)**

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 ;
```

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

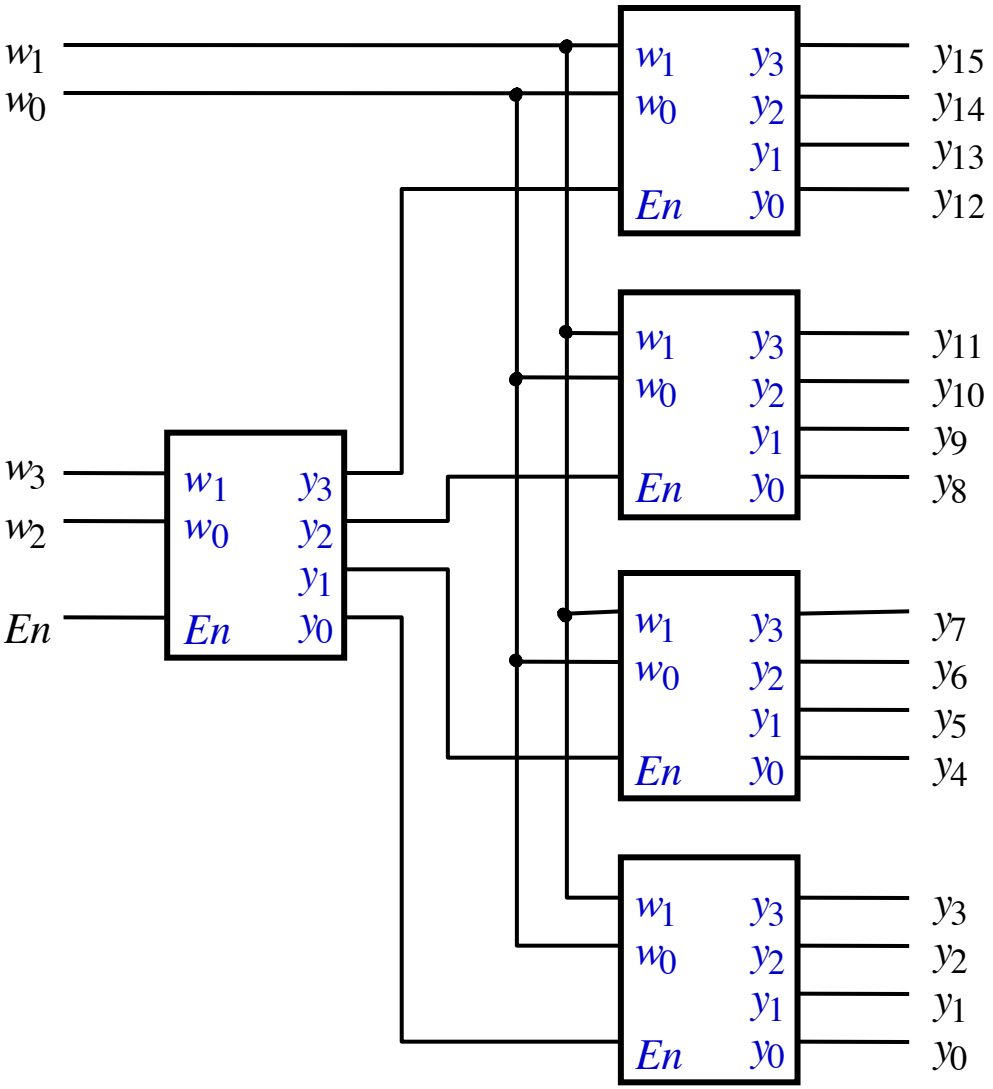
    COMPONENT mux4to1
        PORT (    w0, w1, w2, w3      : IN     STD_LOGIC ;
                  s                   : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                  f                   : OUT    STD_LOGIC ) ;
    END COMPONENT ;

    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
w_1 y_3
w_0 y_2
y_1
En 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 ;
```

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_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)  );
    Dec_left: dec2to4 PORT MAP ( w(3 DOWNTO 2), En, m ) ;
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
    G1: FOR i IN 0 TO 3 GENERATE
        Dec_ri: dec2to4 PORT MAP ( ……., ……., ……..);
    END GENERATE ;
    Dec_left: dec2to4 PORT MAP ( w(3 DOWNTO 2), En, m ) ;
END Structure ;
```

VHDL code for Example 2 (2)

```
ARCHITECTURE Structure OF dec4to16 IS

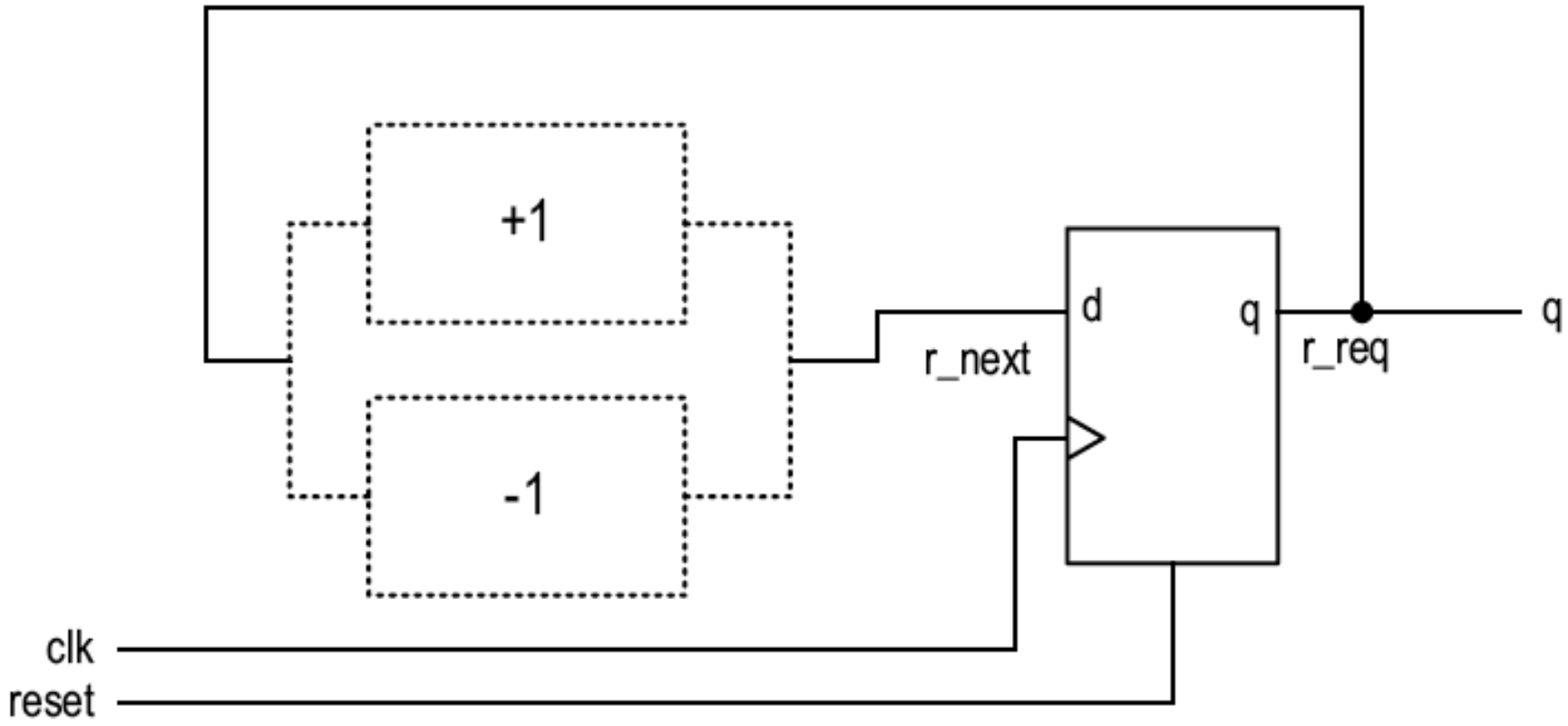
    COMPONENT dec2to4
        PORT (   w        : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                 En       : IN     STD_LOGIC ;
                 y        : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
    END COMPONENT ;

    SIGNAL m : STD_LOGIC_VECTOR(3 DOWNTO 0) ;

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

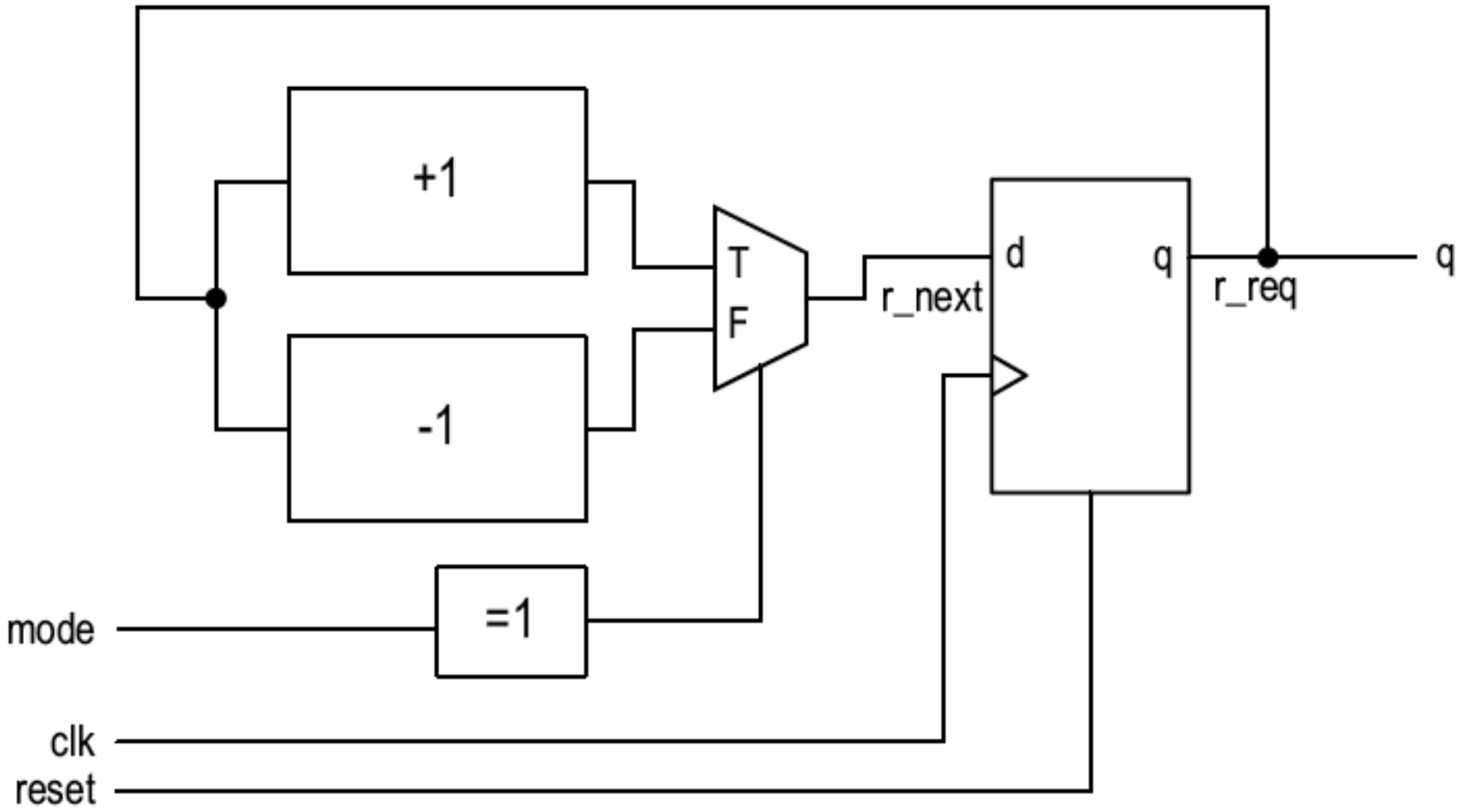
   dec_gen:  --decrementor
   if UP/=1 generate
      r_next <= r_reg – 1;
   end generate;

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

end mixed;
```

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;

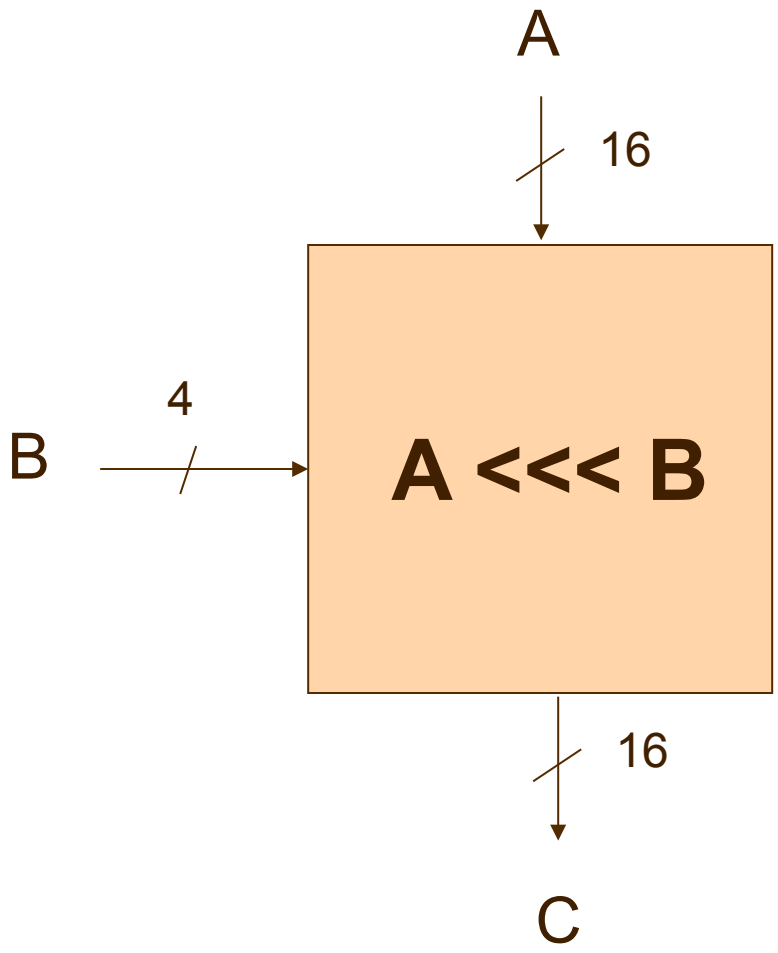
  -- output logic

  q <= std_logic_vector(r_reg);

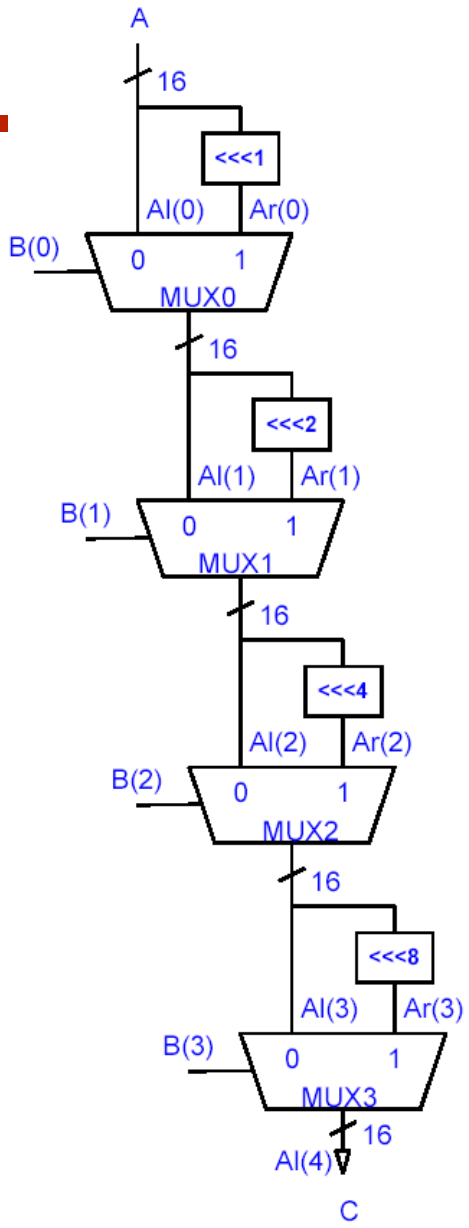
end arch;
```

Example 5
Variable Rotator

Example 3: Variable rotator - Interface



Block diagram





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;
```

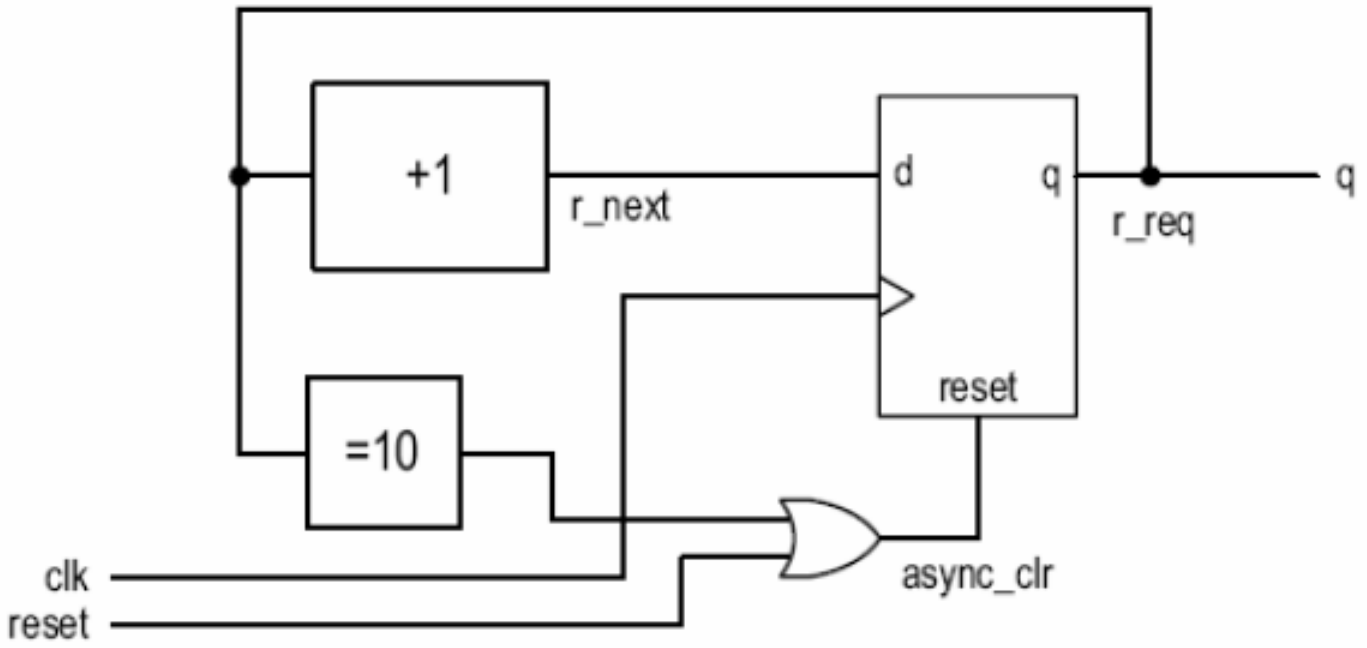


Poor Design Practices

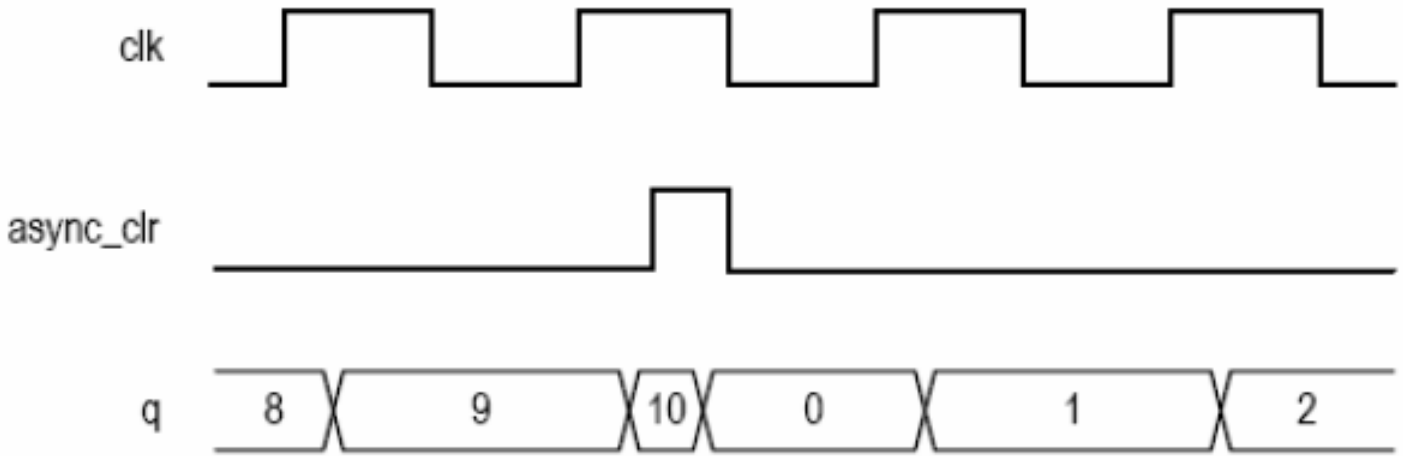
1. Poor design practice and remedy

- Synchronous design is the most important methodology
- Poor practice in the past (to save chips)
  - Misuse of asynchronous reset
  - Misuse of gated clock
  - Misuse of derived clock

Misuse of Asynchronous Reset

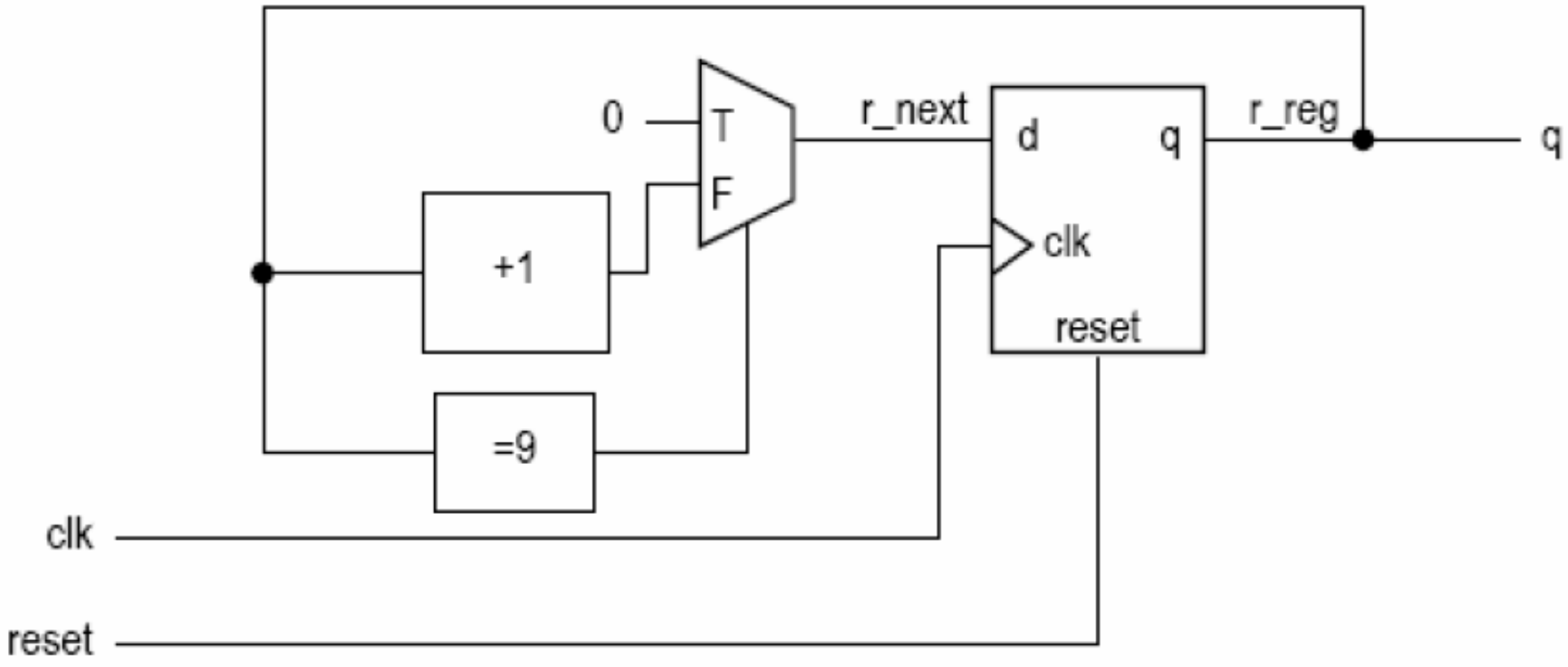


(a) Block diagram



(b) Timing diagram

- Problem
  - Glitches in transition 1001 (9) => 0000 (0)
  - Glitches in aync_clr can reset the counter
  - How about timing analysis? (maximal clock rate)
- Asynchronous reset should only be used for power-on initialization


0
T
F
r_next
d
q
r_reg
q
+1
clk
reset
=9
clk
reset

Decade (mod-10) counter

```
architecture two_seg_arch of mod10_counter is
   constant TEN: integer := 10;
   signal r_reg: unsigned(3 downto 0);
   signal r_next: unsigned(3 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;
   -- next-state logic
   r_next <= (others=>'0') when r_reg=(TEN-1) else
             r_reg + 1;
   -- output logic
   q <= std_logic_vector(r_reg);
end two_seg_arch;
```

Misuse of Gated Clock

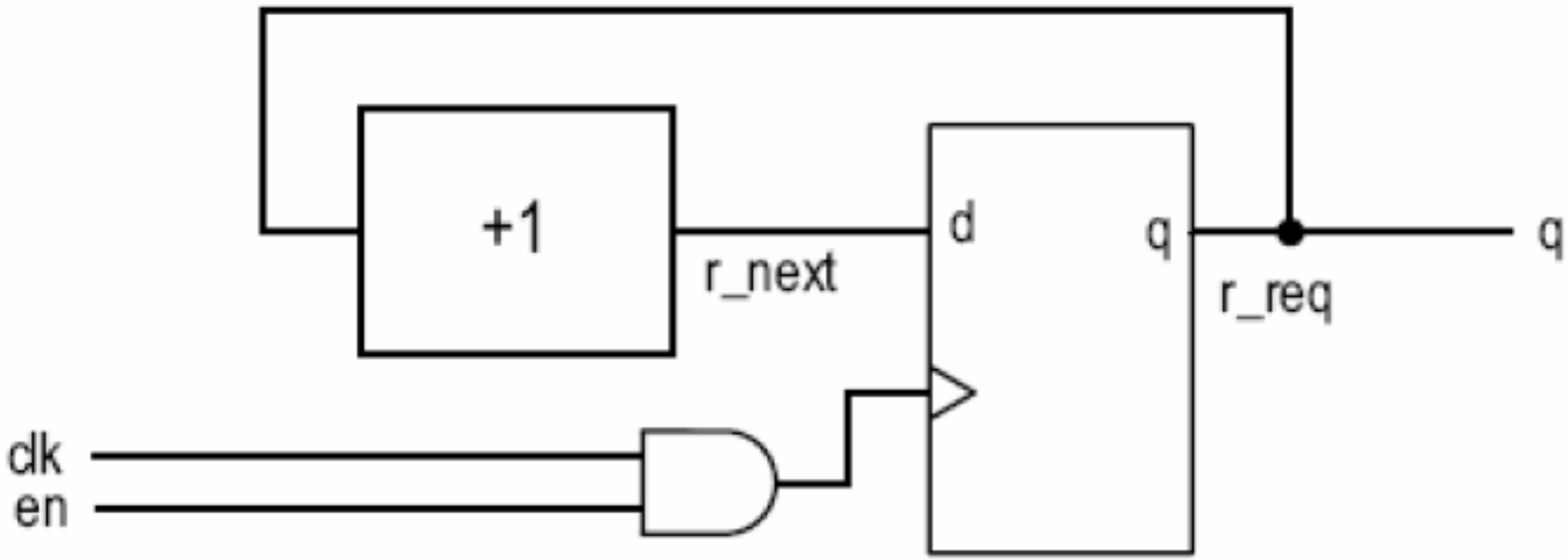
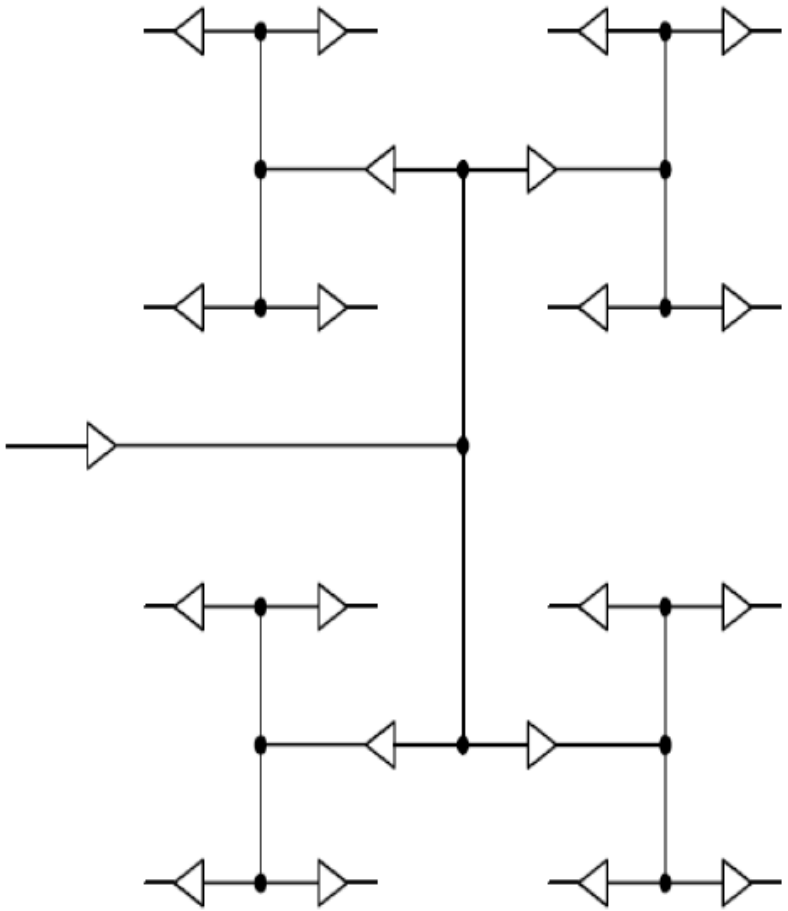


Figure 9.2 Disabling FF with gated clock

- Problem
  - Gated clock width can be narrow
  - Gated clock may pass glitches of en
  - Difficult to design the clock distribution network

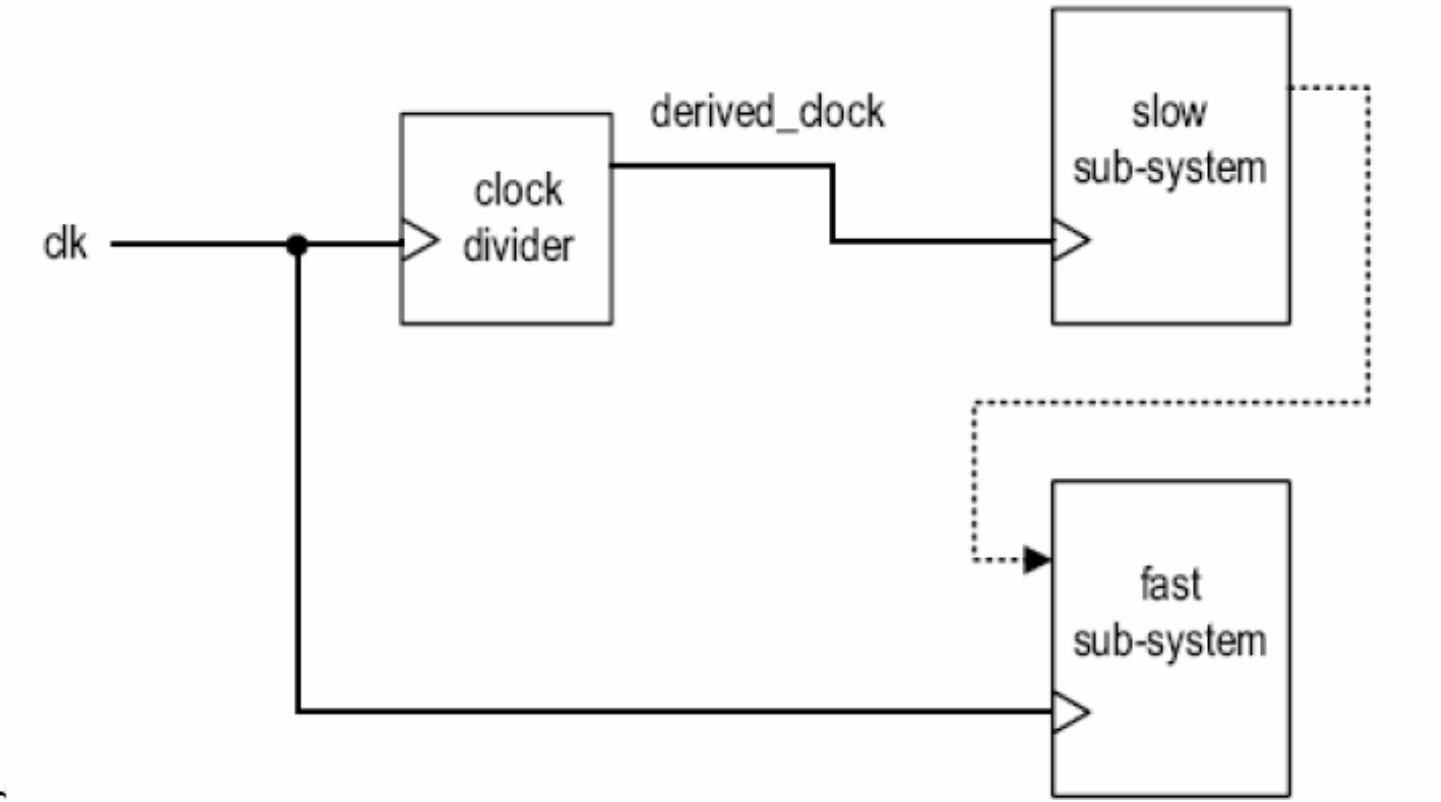
Dedicated Clock Tree Network – H Tree



Misuse of Derived Clock

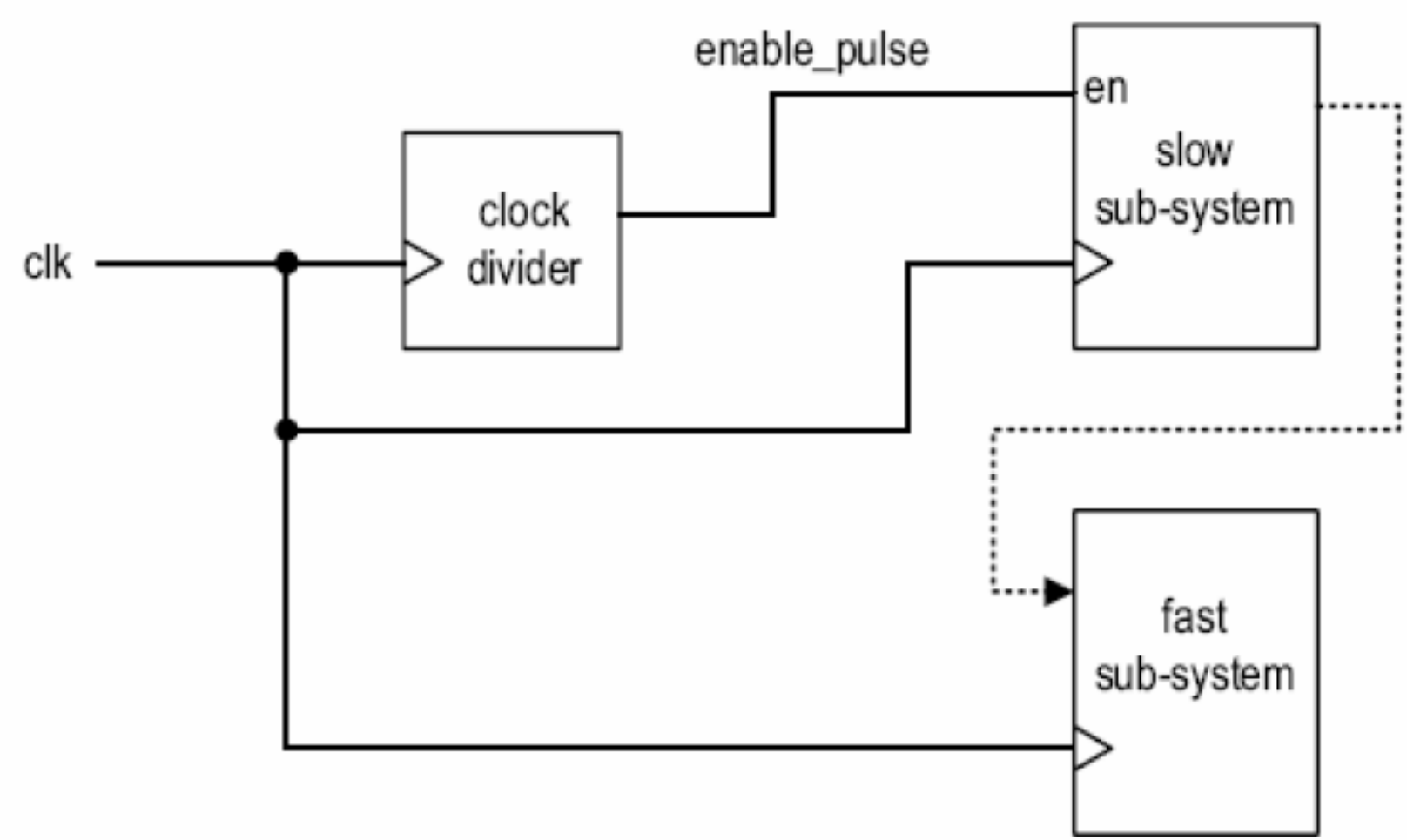
Misuse of derived clock

- Subsystems may run at different clock rate
- Poor design: use a derived slow clock for slow subsystem

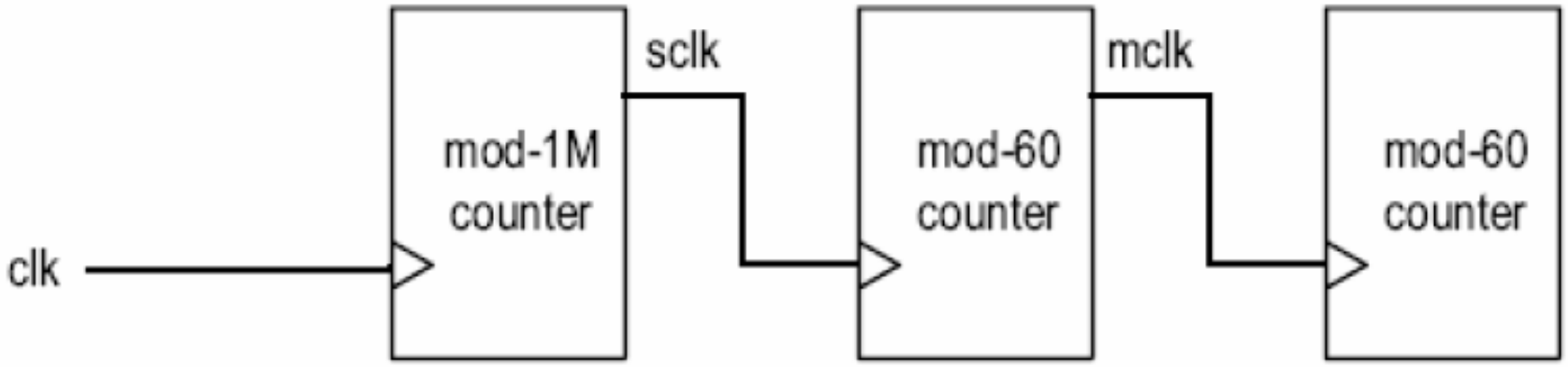


- Problem
  - Multiple clock distribution network
  - How about timing analysis? (maximal clock rate)

- Better use a synchronous one-clock enable pulse

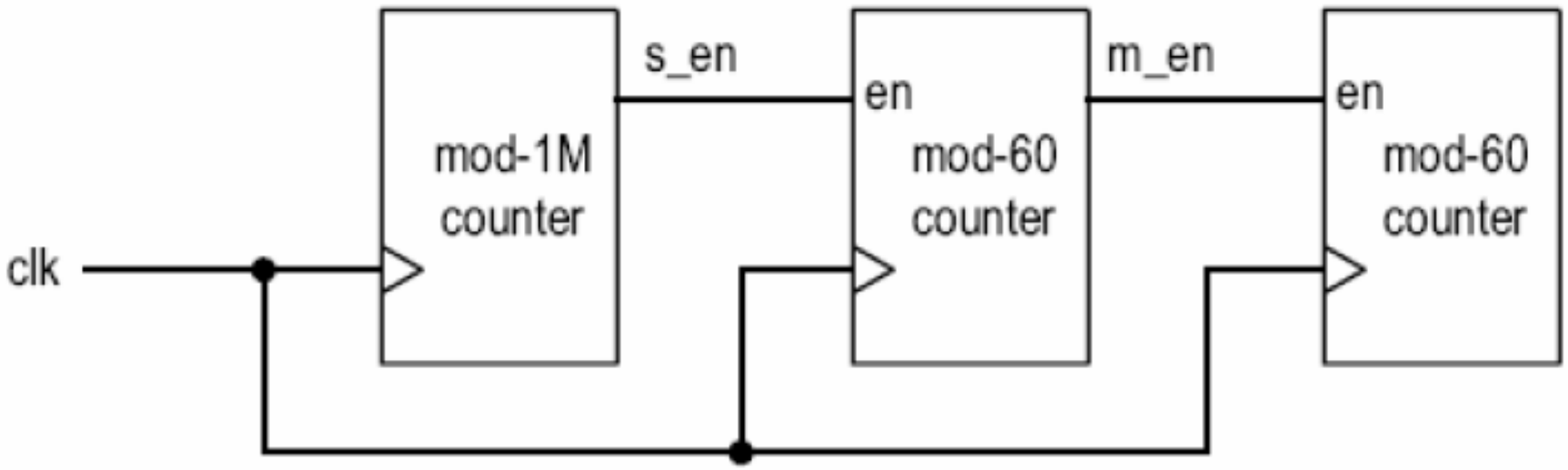


- E.g., second and minutes counter
  - Input: 1 MHz clock
  - Poor design:



(a) Design with derived clock

– Better design



(b) Design with a single synchronous clock