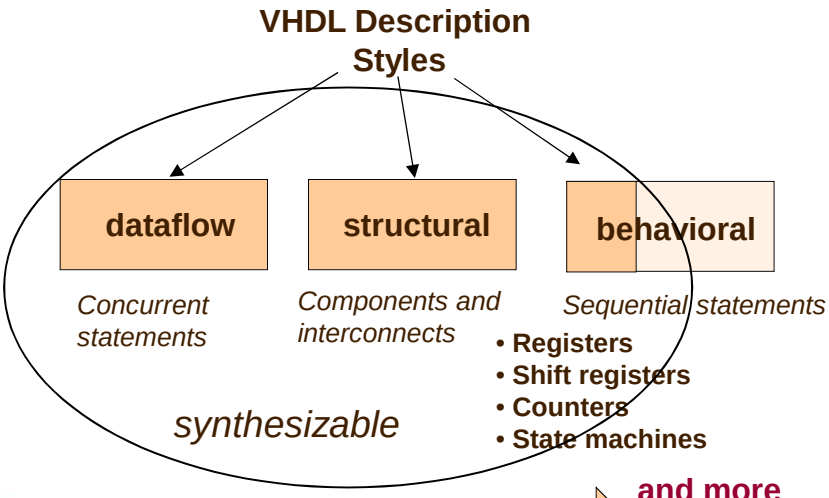
Behavioral Design Style: Registers & Counters

1

VHDL Description Styles



2

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

3

Anatomy of a Process

OPTIONAL

```
[label:] PROCESS [(sensitivity list)]
    [declaration part]
BEGIN
    statement part
END PROCESS [label];
```

4

PROCESS with a SENSITIVITY LIST

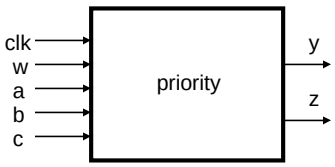
- List of signals to which the process is sensitive.
- Whenever there is an event on any of the signals in the sensitivity list, the process fires.
- Every time the process fires, it will run in its entirety.
- **WAIT statements are NOT ALLOWED in a processes with SENSITIVITY LIST.**

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

5

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 ;
```



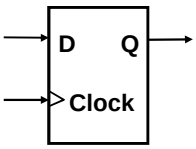
- All signals which appear on the sensitivity list are inputs e.g. *clk*
- All signals which appear on the left of signal assignment statement (<=) are outputs e.g. *y, z*
- Note that not all inputs need to be included on the sensitivity list
- All signals which appear on the right of signal assignment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

6

Registers

D flip-flop

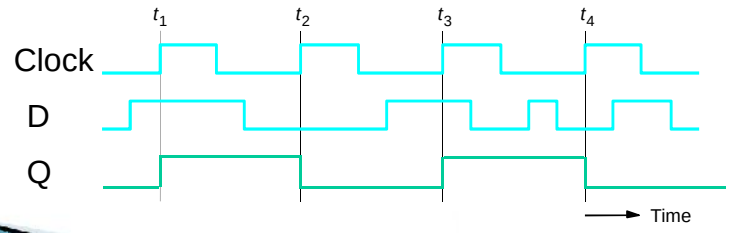
Graphical symbol



Truth table

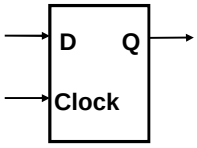
Clk	D	Q(*t*+1)
↑	0	0
↑	1	1
0	–	Q(*t*)
1	–	Q(*t*)

Timing diagram



D latch

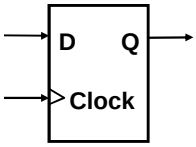
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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;
```



9

D flip-flop

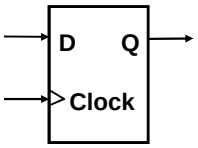
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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;
```



10

D flip-flop

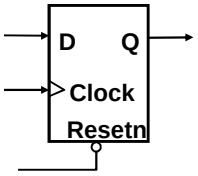
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```



11

D flip-flop with asynchronous reset

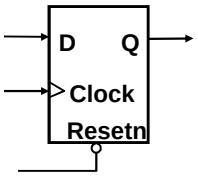
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```



12

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 ;
```



13

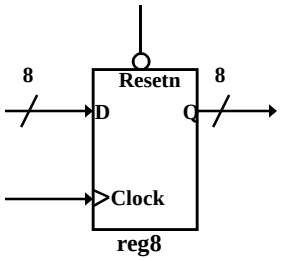
Asychronous vs. Synchronous

- 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

14

8-bit register with asynchronous reset

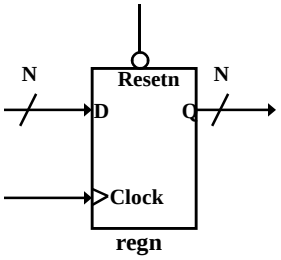
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```



15

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 ;
```



16

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

17

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')

18

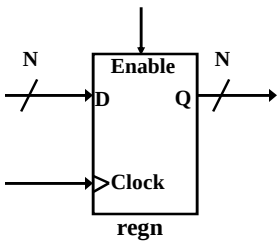
Component Instantiation in VHDL-93

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

19

N-bit register with enable

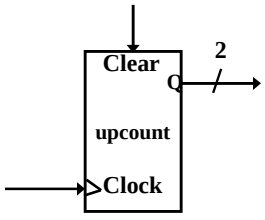
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```



20

2-bit up-counter with synchronous reset

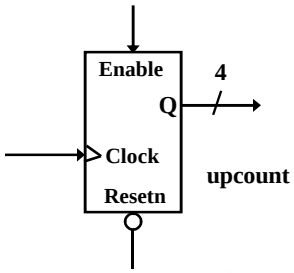
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
        PORT (      Clear, Clock      : IN          STD_LOGIC ;
                    Q                 : OUT     STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END upcount ;

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;
```



22

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

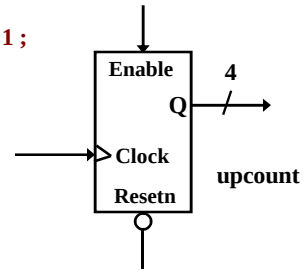
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY upcount_ar IS
      PORT (   Clock, Resetn, Enable                    : IN
               Q                  : OUT                 STD_LOGIC_VECTOR (3 I
END upcount_ar ;
```



23

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 ;
```

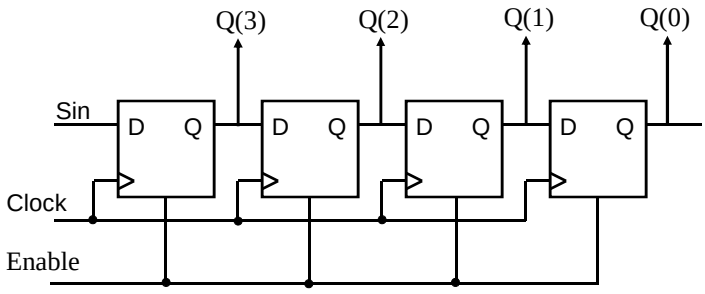


24

Shift Registers

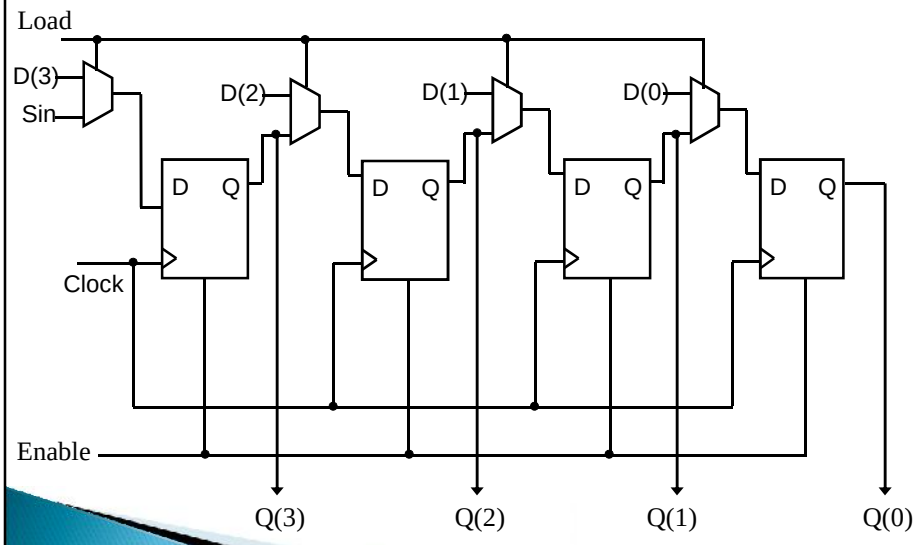
25

Shift register – internal structure



26

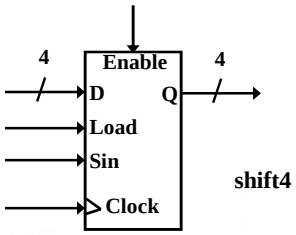
Shift Register With Parallel Load



27

4-bit shift register with parallel load (1)

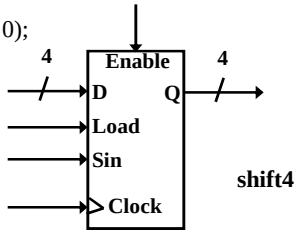
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```



28

4-bit shift register with parallel load (2)

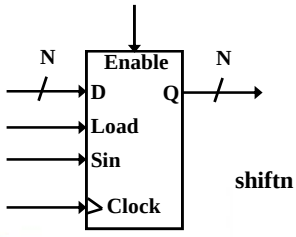
```
ARCHITECTURE behavioral OF shift4 IS
        SIGNAL Qt : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
        PROCESS (Clock)
        BEGIN
                IF rising_edge(Clock) THEN
                        IF Enable = '1' THEN
                                IF Load = '1' THEN
                          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



29

N-bit shift register with parallel load (1)

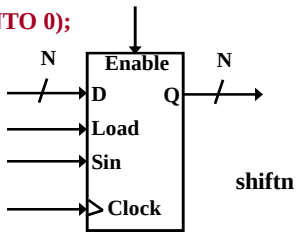
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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 ;
```



30

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 PROCESS ;
        Q <= Qt;
END behavior al;
```



31

Sequential Logic Synthesis for Beginners

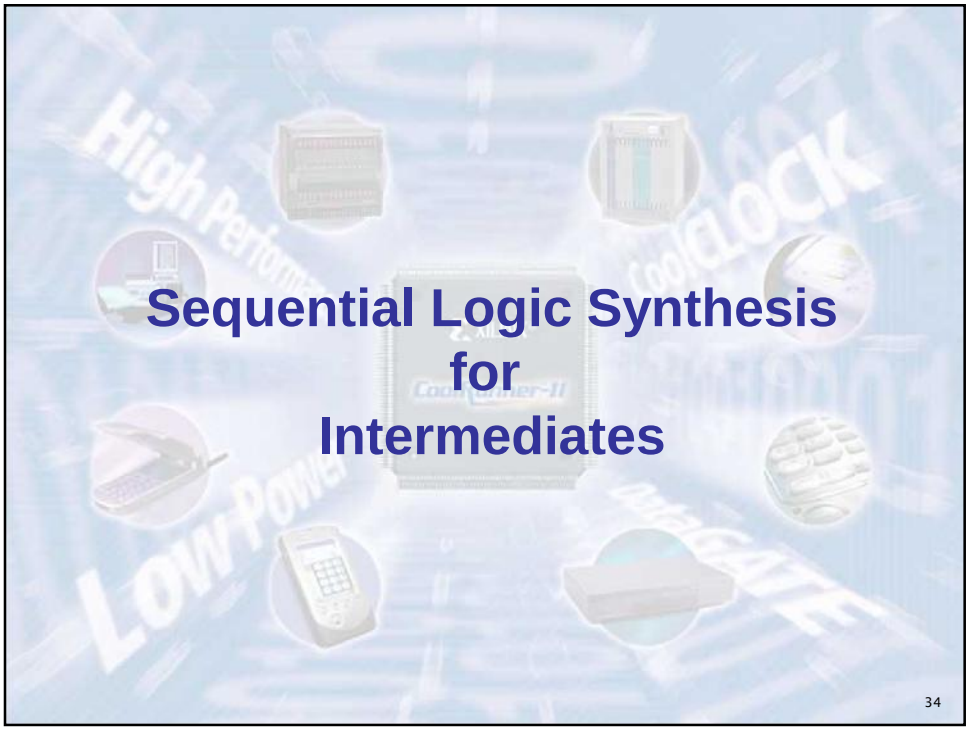
32

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.

33



34

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)

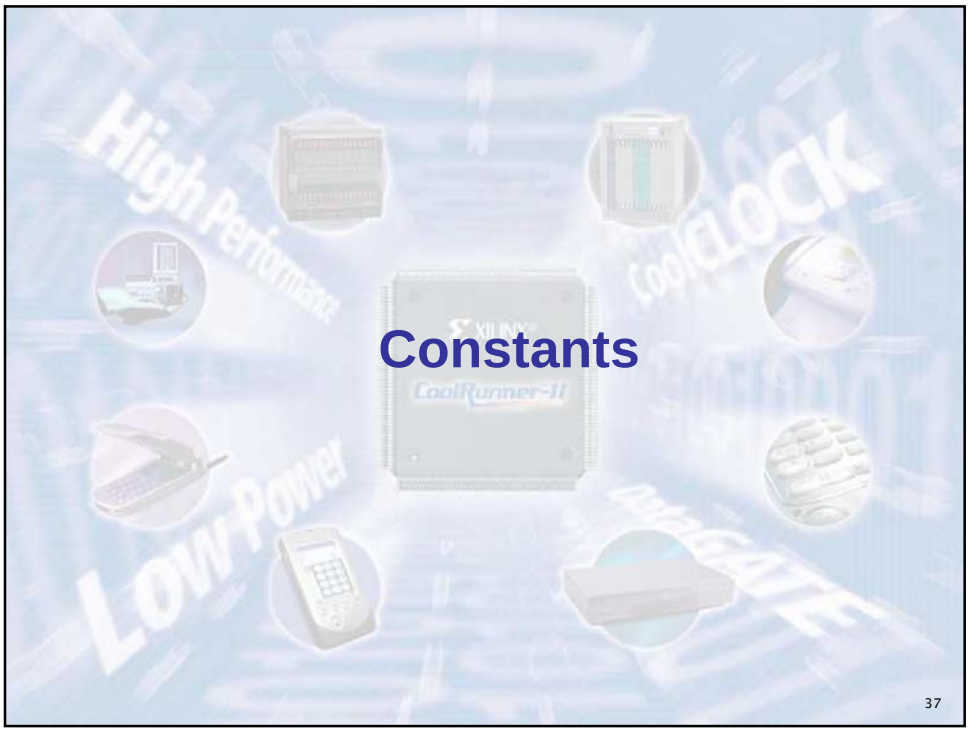
35

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;
```

36



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;
```

38

Dataflow VHDL

Major instructions

Concurrent statements

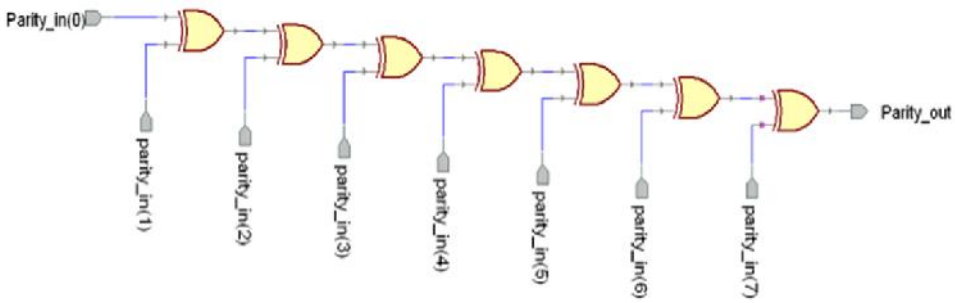
- concurrent signal assignment (⇐)
- conditional concurrent signal assignment (when-else)
- selected concurrent signal assignment (with-select-when)
- **generate scheme for equations (for-generate)**

40

PARITY Example

41

PARITY: Block Diagram



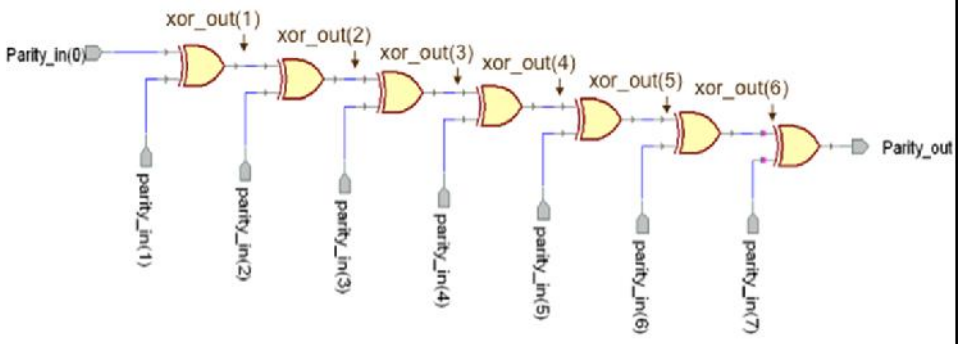
42

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;
```

43

PARITY: Block Diagram



44

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;
```

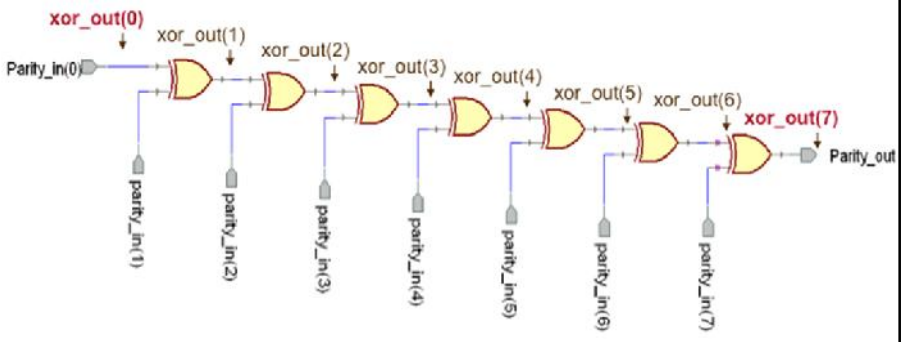
45

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;
```

46

PARITY: Block Diagram (2)



47

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;
```

48

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;
```

49

For Generate Statement

For - Generate

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

if - Generate

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

50

Structural VHDL

Major instructions

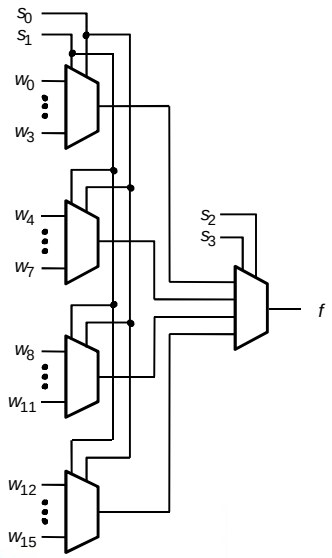
- component instantiation (port map)
- component instantiation with generic (generic map, port map)
- **generate scheme for component instantiations (for-generate)**

52

Example 1

53

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 ;
```

55

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 ;
```

56

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

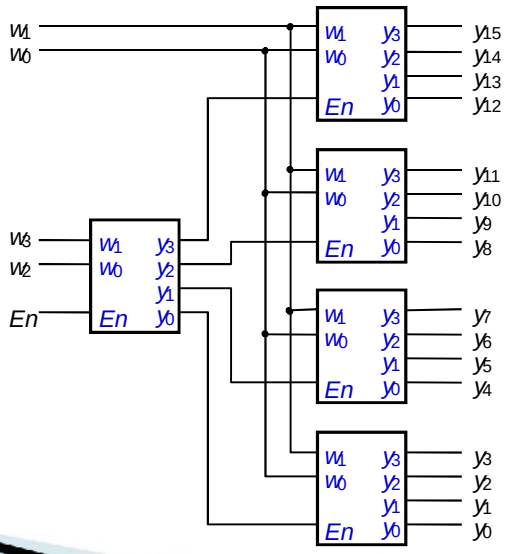
```
ARCHITECTURE Structure OF Example1 IS

    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 ;
```

59

Example 2

60

Example 2



61

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 ;
```

62

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 ;
```

63

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 ;
```

64

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 ;
```

65

VHDL code for Example 2 (2)

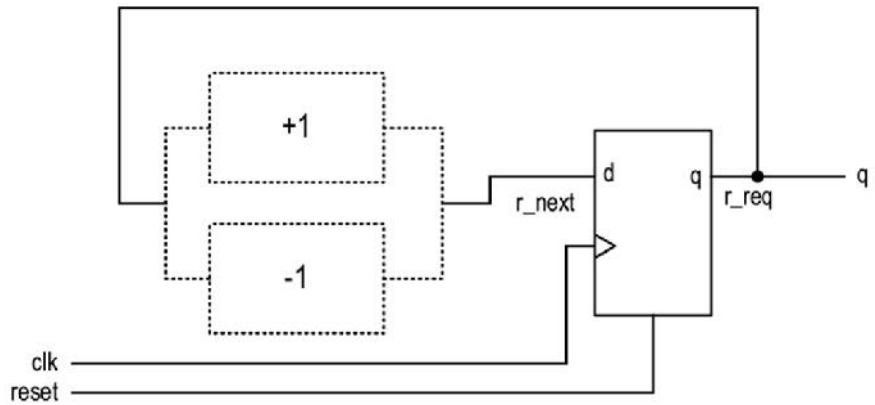
```
ARCHITECTURE Structure OF dec4to16 IS

   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 ;
```

66

Example 3
Up-or-down Free Running Counter

67

Up-or-down Free Running Counter



68

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;
```

69

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;
```

70

Up-or-down Free Running Counter (3)

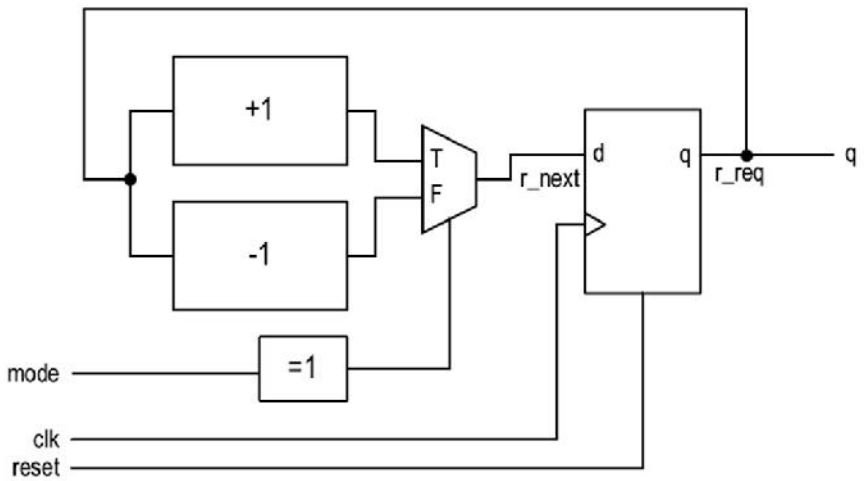
```
-- next-state logic
  inc_gen:  -- incrementor
  if UP=1 generate
    r_next <= r_reg + 1;
  end generate;

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

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

end mixed;
```

71

Example 4
Up-and-down Free Running Counter

72

Up-and-down Free Running Counter



73

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;
```

74

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;
```

75

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;
```

76