COE 561
Digital System Design & Synthesis
Introduction to VHDL

Dr. Aiman H. El-Maleh
Computer Engineering Department
King Fahd University of Petroleum & Minerals

Outline …

- Hardware description languages
- VHDL terms
- Design Entity
- Design Architecture
- VHDL model of full adder circuit
- VHDL model of 1's count circuit
- Other VHDL model examples
- Structural modeling of 4-bit comparator
- Design parameterization using Generic

2

… Outline

- **Test Bench example**
- **VHDL objects**
- **Variables vs. Signals**
- **Signal assignment & Signal attributes**
- **Subprograms, Packages, and Libraries**
- **Data types in VHDL**
- **Data flow modeling in VHDL**
- **Behavioral modeling in VHDL**

3

Hardware Description Languages

- **HDLs are used to describe the hardware for the purpose of modeling, simulation, testing, design, and documentation.**
  - Modeling: behavior, flow of data, structure
  - Simulation: verification and test
  - Design: synthesis
- **Two widely-used HDLs today**
  - **VHDL:** VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
  - **Verilog** (from Cadence, now IEEE standard)

4

Styles in VHDL

- **Behavioral**
  - High level, algorithmic, sequential execution
  - Hard to synthesize well
  - Easy to write and understand (like high-level language code)
- **Dataflow**
  - Medium level, register-to-register transfers, concurrent execution
  - Easy to synthesize well
  - Harder to write and understand (like assembly code)
- **Structural**
  - Low level, netlist, component instantiations and wiring
  - Trivial to synthesize
  - Hardest to write and understand (very detailed and low level)

5

VHDL Terms …

- **Entity:**
  - All designs are expressed in terms of entities
  - Basic building block in a design
- **Ports:**
  - Provide the mechanism for a device to communication with its environment
  - Define the names, types, directions, and possible default values for the signals in a component's interface
- **Architecture:**
  - All entities have an architectural description
  - Describes the behavior of the entity
  - A single entity can have multiple architectures (behavioral, structural, …etc)
- **Configuration:**
  - A configuration statement is used to bind a component instance to an entity-architecture pair.
  - Describes which behavior to use for each entity

6

… VHDL Terms …

- **Generic:**
  - A parameter that passes information to an entity
  - Example: for a gate-level model with rise and fall delay, values for the rise and fall delays passed as generics
- **Process:**
  - Basic unit of execution in VHDL
  - All operations in a VHDL description are broken into single or multiple processes
  - Statements inside a process are processed sequentially
- **Package:**
  - A collection of common declarations, constants, and/or subprograms to entities and architectures.

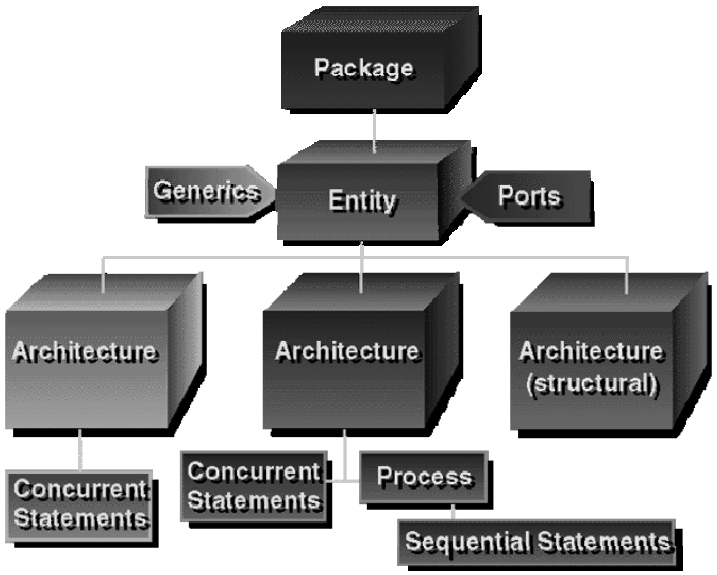
7

VHDL Terms …

- **Attribute:**
  - Data attached to VHDL objects or predefined data about VHDL objects
  - Examples:
    - maximum operation temperature of a device
    - Current drive capability of a buffer
- **VHDL is NOT Case-Sensitive**
  - Begin = begin = beGiN
- **Semicolon " ; " terminates declarations or statements.**
- After a double minus sign (--) the rest of the line is treated as a comment

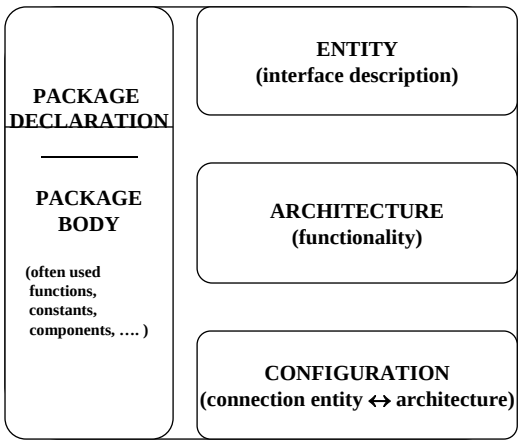
8

VHDL Models …



9

… VHDL Models



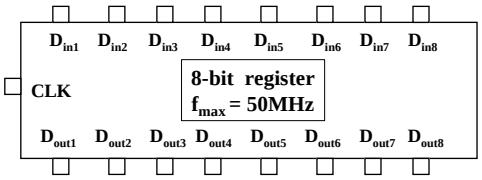
10

Design Entity …

- **In VHDL, the name of the system is the same as the name of its entity.**
- **Entity comprises two parts:**
  - *parameters* of the system as seen from outside such as bus-width of a processor or max clock frequency
  - *connections* which are transferring information to and from the system (system's inputs and outputs)
- **All parameters are declared as generics and are passed on to the body of the system**
- **Connections, which carry data to and from the system, are called *ports*. They form the second part of the entity.**

11

Illustration of an Entity



entity Eight_bit_register is

parameters

CLK one-bit input

connections

end [entity] [Eight_bit_register]

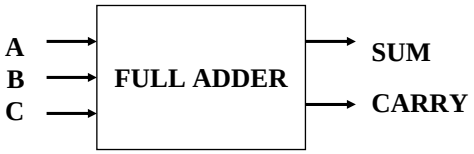
12

Entity Examples …

- Entity FULLADDER is

  -- Interface description of FULLADDER

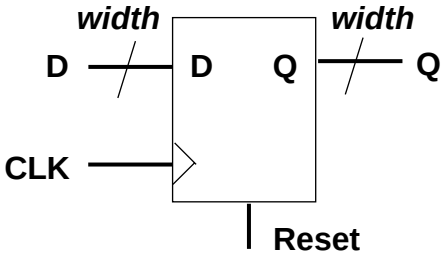
  port (A, B, C: in bit;
  SUM, CARRY: out bit);
  end FULLADDER;



13

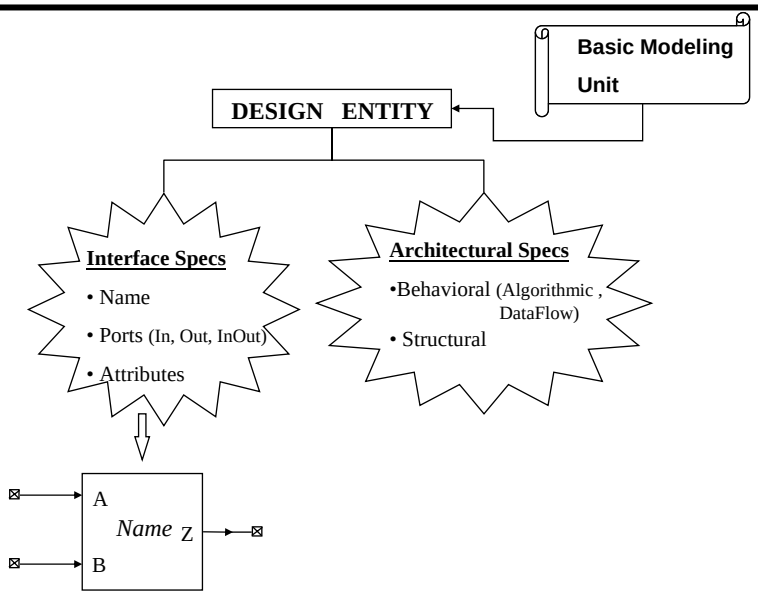
… Entity Examples

- **Entity Register is**
  -- parameter: width of the register
  generic (width: integer);
  --input and output signals
  port (CLK, Reset: in bit;
  D: in bit_vector(1 to width);
  Q: out bit_vector(1 to width));
  end Register;



14

… Design Entity



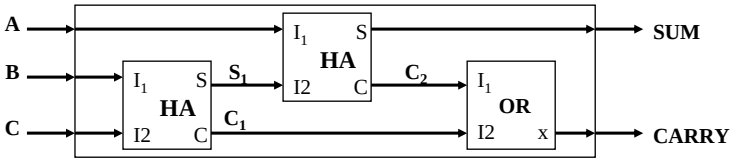
Architecture Examples: Behavioral Description

- **Entity FULLADDER is**
  port (A, B, C: in bit;
  SUM, CARRY: out bit);
  end FULLADDER;

- **Architecture CONCURRENT of FULLADDER is**
  begin
  SUM <= A xor B xor C after 5 ns;
  CARRY <= (A and B) or (B and C) or (A and C) after 3 ns;
  end CONCURRENT;

Architecture Examples: Structural Description …

- architecture STRUCTURAL of FULLADDER is
  signal S1, C1, C2 : bit;
  component HA
    port (I1, I2 : in bit; S, C : out bit);
  end component;
  component OR
    port (I1, I2 : in bit; X : out bit);
  end component;
  begin
  INST_HA1 : HA port map (I1 => B, I2 => C, S => S1, C => C1);
  INST_HA2 : HA port map (I1 => A, I2 => S1, S => SUM, C => C2);
  INST_OR : OR port map (I1 => C2, I2 => C1, X => CARRY);
  end STRUCTURAL;



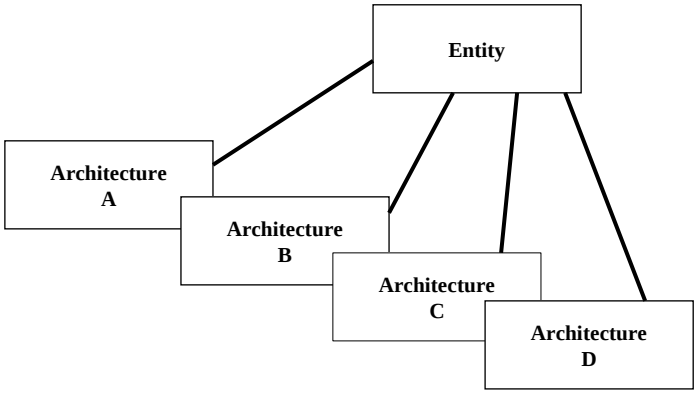
… Architecture Examples: Structural Description

```
Entity HA is
PORT (I1, I2 : in bit; S, C : out bit);
end HA ;
Architecture behavior of HA is
begin
   S <= I1 xor I2;
   C <= I1 and I2;
end behavior;
```

```
Entity OR is
PORT (I1, I2 : in bit; X : out bit);
end OR ;
Architecture behavior of OR is
begin
   X <= I1 or I2;
end behavior;
```

One Entity Many Descriptions

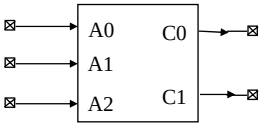
- **A system (an *entity*) can be specified with different *architectures***



19

Example: Ones Count Circuit

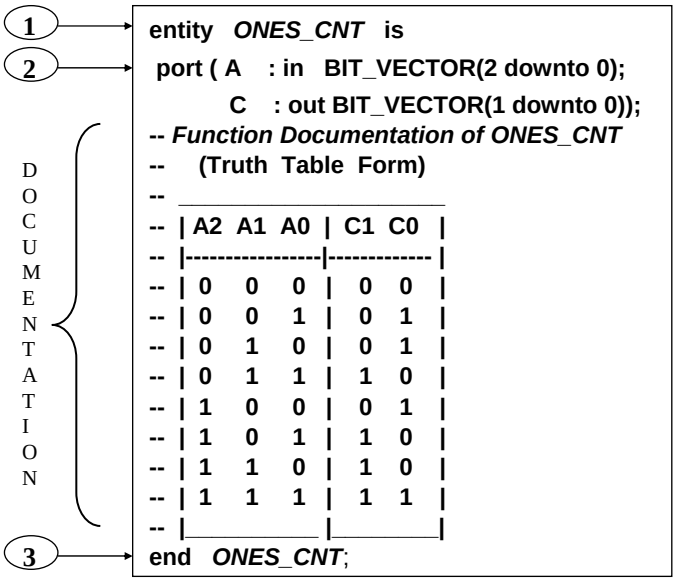
- **Value of C1 C0 = No. of ones in the inputs A2, A1, and A0**
  - C1 is the Majority Function (=1 iff two or more inputs =1)
  - C0 is a 3-Bit Odd-Parity Function (OPAR3))
  - C1 = A1 A0 + A2 A0 + A2 A1
  - C0 = A2 A1' A0' + A2' A1 A0' + A2' A1' A0 + A2 A1 A0



20

10

Ones Count Circuit Interface Specification



21

Ones Count Circuit Architectural Body: Behavioral (Truth Table)

```
Architecture Truth_Table of ONES_CNT is

begin
   Process(A)    -- Sensitivity List Contains only Vector A
    begin

        CASE A  is
                WHEN "000" =>     C <= "00";
                WHEN "001" =>     C <= "01";
                WHEN "010" =>     C <= "01";
                WHEN "011" =>     C <= "10";
                WHEN "100" =>     C <= "01";
                WHEN "101" =>     C <= "10";
                WHEN "110" =>     C <= "10";
                WHEN "111" =>     C <= "11";
        end CASE;
    end process;

end Truth_Table;
```

22

11

Ones Count Circuit Architectural Body: Behavioral (Algorithmic)

```
Architecture  Algorithmic  of  ONES_CNT  is
begin

    Process(A)    -- Sensitivity List Contains only Vector A
      Variable  num: INTEGER range 0 to 3;

    begin
        num :=0;
        For i   in   0 to 2 Loop
                IF A(i) = '1' then
                        num := num+1;
                end if;
        end Loop;
--
--      Transfer "num"  Variable Value to a SIGNAL
--
        CASE num is
                WHEN 0 =>       C <= "00";
                WHEN 1 =>       C <= "01";
                WHEN 2 =>       C <= "10";
                WHEN 3 =>       C <= "11";
        end CASE;
    end process;

end Algorithmic;
```

23

Ones Count Circuit Architectural Body: Data Flow

- **C1 = A1 A0 + A2 A0 + A2 A1**
- **C0 = A2 A1' A0' + A2' A1 A0' + A2' A1' A0 + A2 A1 A0**

```
Architecture  Dataflow  of  ONES_CNT  is
begin

   C(1) <=(A(1) and A(0)) or (A(2) and A(0))
            or (A(2) and A(1));

   C(0) <= (A(2) and not A(1) and not A(0))
            or (not A(2) and A(1) and not A(0))
            or (not A(2) and not A(1) and A(0))
            or (A(2) and A(1) and A(0));

end Dataflow;
```

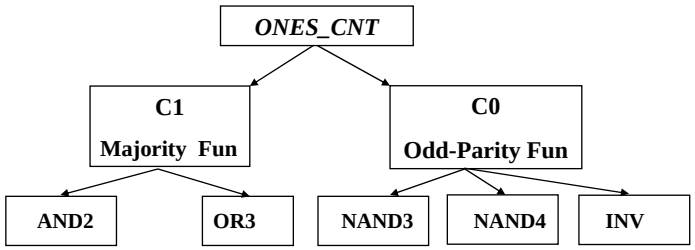
24

12

Ones Count Circuit Architectural Body: Structural …

- C1 = A1 A0 + A2 A0 + A2 A1 = MAJ3(A)
- C0 = A2 A1' A0' + A2' A1 A0' + A2' A1' A0 + A2 A1 A0
  = OPAR3(A)

Structural Design Hierarchy



25

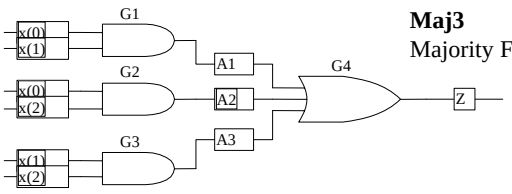
Ones Count Circuit Architectural Body: Structural …

- Entity MAJ3 is
  PORT(X: in BIT_Vector(2 downto 0);
  Z: out BIT);
  end MAJ3;

- Entity OPAR3 is
  PORT(X: in BIT_Vector(2 downto 0);
  Z: out BIT);
  end OPAR3;

26

VHDL Structural Description of Majority Function …



Maj3
Majority Function

Architecture Structural **of** *MAJ3* **is**
Component AND2
PORT(I1, I2: **in BIT**; O: **out BIT**);
end Component ;
Component OR3
PORT(I1, I2, I3: **in BIT**; O: **out BIT**);
end Component ;

Declare Components To be Instantiated

27

VHDL Structural Description of Majority Function

SIGNAL A1, A2, A3: BIT; ***Declare Maj3 Local Signals***
begin
-- *Instantiate Gates*
g1**: AND2 PORT MAP (X(0), X(1), A1);**
g2**: AND2 PORT MAP (X(0), X(2), A2);**
g3**: AND2 PORT MAP (X(1), X(2), A3);**
g4**: OR3 PORT MAP (A1, A2, A3, Z);**
end Structural;

Wiring of Maj3 Components

28

14

VHDL Structural Description of Odd Parity Function …

Architecture Structural **of** *OPAR3* **is**

Component INV

PORT(Ipt: in BIT; Opt: out BIT);

end Component ;

Component NAND3

PORT(I1, I2, I3: in BIT;

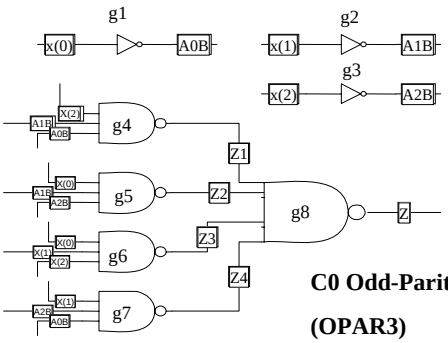
O: out BIT);

end Component ;

Component NAND4

PORT(I1, I2, I3, I4: in BIT;
O: out BIT);

end Component ;



C0 Odd-Parity (OPAR3)

VHDL Structural Description of Odd Parity Function

```
SIGNAL A0B, A1B, A2B, Z1, Z2, Z3, Z4: BIT;
begin
   g1: INV PORT MAP (X(0), A0B);
   g2: INV PORT MAP (X(1), A1B);
   g3: INV PORT MAP (X(2), A2B);
   g4: NAND3 PORT MAP (X(2), A1B, A0B, Z1);
   g5: NAND3 PORT MAP (X(0), A1B, A2B, Z2);
   g6: NAND3 PORT MAP (X(0), X(1), X(2), Z3);
   g7: NAND3 PORT MAP (X(1), A2B, A0B, Z4);
   g8: NAND4 PORT MAP (Z1, Z2, Z3, Z4, Z);
end Structural;
```


15

VHDL Top Structural Level of Ones Count Circuit

```
Architecture Structural of ONES_CNT is
Component MAJ3
   PORT( X: in BIT_Vector(2 downto 0); Z: out BIT);
END Component ;
Component OPAR3
   PORT( X: in BIT_Vector(2 downto 0); Z: out BIT);
END Component ;
begin
-- Instantiate Components
   c1: MAJ3 PORT MAP (A, C(1));
   c2: OPAR3  PORT MAP (A, C(0));
end Structural;
```

31

VHDL Behavioral Definition of Lower Level Components

```
Entity INV is
        PORT( Ipt: in BIT;
         Opt: out BIT);
end INV;
Architecture behavior of INV is
begin
        Opt <= not Ipt;
end behavior;
```

```
Entity NAND2 is
        PORT( I1, I2: in BIT;
         O: out BIT);
end NAND2;
Architecture behavior of NAND2 is
begin
        O <= not (I1 and I2);
end behavior;
```

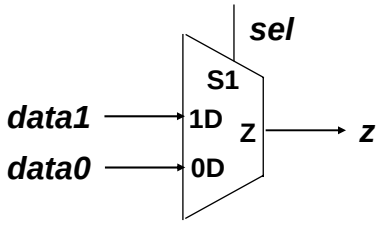
Other Lower Level Gates Are Defined Similarly

32

16

VHDL Model of 2x1 Multiplexer

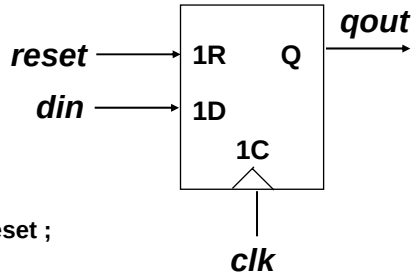
```
Entity mux2_1 IS
    Generic (dz_delay: TIME := 6 NS);
    PORT (sel, data1, data0: IN BIT; z: OUT BIT);
END mux2_1;

Architecture dataflow OF mux2_1 IS
Begin
    z <= data1 AFTER dz_delay WHEN sel='1' ELSE
            data0 AFTER dz_delay;
END dataflow;
```



33

VHDL Model of D-FF – Synchronous Reset

```
Entity DFF IS
    Generic (td_reset, td_in: TIME := 8 NS);
    PORT (reset, din, clk: IN BIT; qout: OUT BIT :='0');
END DFF;
Architecture behavioral OF DFF IS
Begin
    Process(clk)
    Begin
        IF (clk = '0' AND clk'Event ) Then
            IF reset = '1' Then
                qout <= '0'  AFTER    td_reset ;
            ELSE
                qout <= din  AFTER   td_in ;
            END IF;
        END IF;
    END process;
END behavioral ;
```

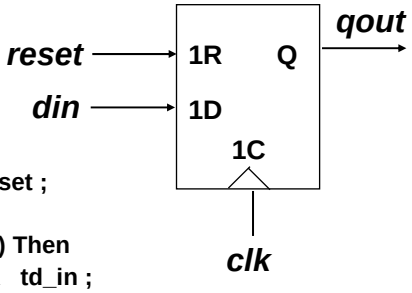


34

17

VHDL Model of D-FF – Asynchronous Reset

```
Entity DFF IS
    Generic (td_reset, td_in: TIME := 8 NS);
    PORT (reset, din, clk: IN BIT; qout: OUT BIT :='0');
END DFF;
Architecture behavioral OF DFF IS
Begin
    Process(clk, reset)
    Begin
            IF reset = '1' Then
                qout <= '0'  AFTER    td_reset ;
            ELSE
                IF (clk = '0' AND clk'Event ) Then
                        qout <= din  AFTER   td_in ;
                END IF;
            END IF;
     END process;
END behavioral ;
```



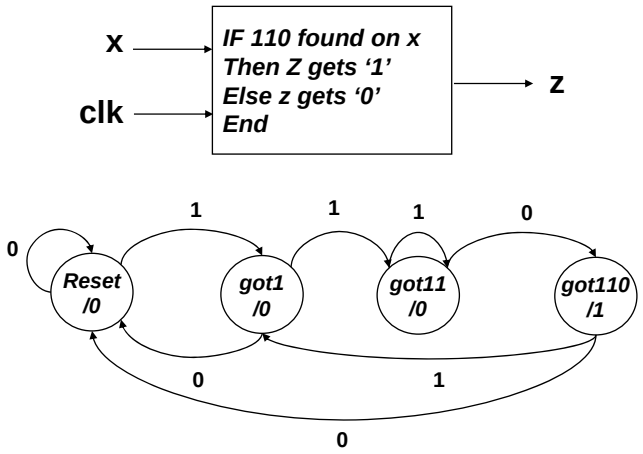
Divide-by-8 Counter

```
Entity counter IS
    Generic (td_cnt: TIME := 8 NS);
    PORT (reset, clk: IN BIT; counting: OUT BIT :='0');
    Constant limit: INTEGER :=8;
END counter ;
Architecture behavioral OF counter IS
Begin
    Process(clk)
        Variable count: INTEGER := limit;
    Begin
       IF (clk = '0'  AND  clk'Event ) THEN
            IF reset = '1' THEN          count := 0 ;
            ELSE  IF   count < limit THEN count:= count+1; END IF;
            END IF;
            IF count = limit Then        counting <= '0' AFTER td_cnt;
            ELSE                         counting <= '1' AFTER td_cnt;
           END IF;
      END IF;
   END process;
END behavioral ;
```


18

Controller Description

- **Moore Sequence Detector**
  - Detection sequence is 110



37

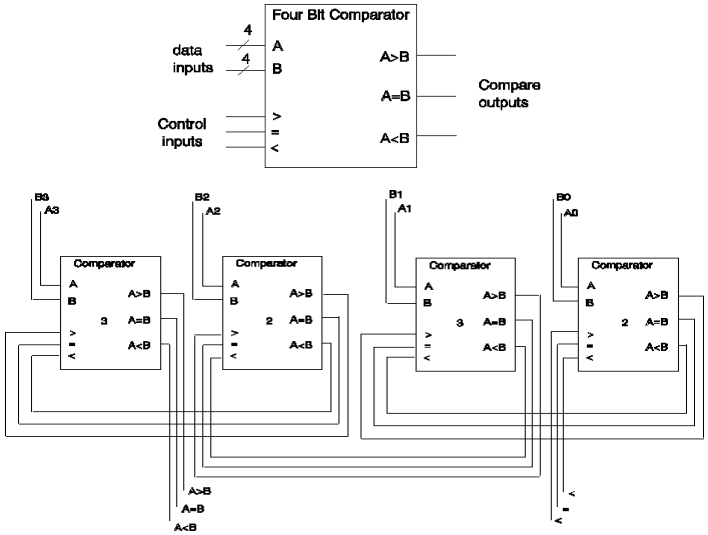
VHDL Description of Moore 110 Sequence Detector

```
ENTITY moore_110_detector IS
    PORT (x, clk : IN BIT; z : OUT BIT);
END moore_110_detector;
 ARCHITECTURE behavioral OF moore_110_detector IS
    TYPE state IS (reset, got1, got11, got110);
    SIGNAL current : state := reset;
BEGIN
    PROCESS(clk)
    BEGIN
        IF (clk = '1' AND CLK'Event) THEN
          CASE current IS
                WHEN reset =>
                        IF x = '1' THEN current <= got1;
                        ELSE current <= reset; END IF;
                WHEN got1 =>
                        IF x = '1' THEN current <= got11;
                        ELSE current <= reset; END IF;
                WHEN got11 =>
                        IF x = '1' THEN current <= got11;
                        ELSE current <= got110; END IF;
                WHEN got110 =>
                        IF x = '1' THEN current <= got1;
                        ELSE current <= reset; END IF;
          END CASE;
        END IF;
    END PROCESS;
    z <='1' WHEN current = got110 ELSE '0';
END behavioral;
```

38

19

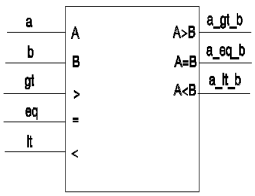
Structural 4-Bit Comparator

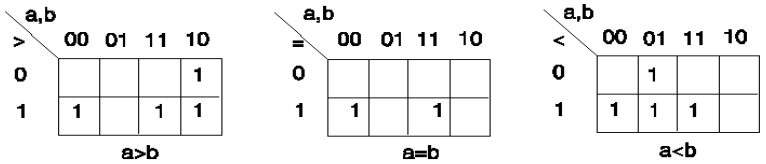


39

A Cascadable Single-Bit Comparator

- When a > b the a_gt_b becomes 1
- When a < b the a_lt_b becomes 1
- If a = b outputs become the same as corresponding inputs

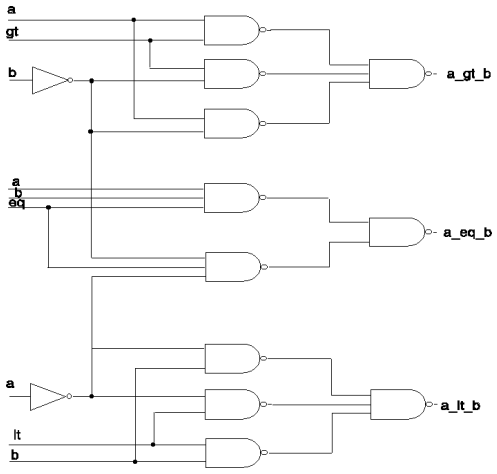




40

Structural Single-Bit Comparator

- Design uses basic components
- The less-than and greater-than outputs use the same logic



41

Structural Model of Single-Bit Comparator …

```
ENTITY bit_comparator IS
        PORT (a, b, gt,  eq,  lt : IN BIT;  a_gt_b,  a_eq_b, a_lt_b : OUT BIT);
END bit_comparator;
ARCHITECTURE gate_level OF bit_comparator IS
--
COMPONENT n1 PORT (i1: IN BIT; o1: OUT BIT); END COMPONENT ;
COMPONENT n2 PORT (i1,i2: IN BIT; o1:OUT BIT); END COMPONENT;
COMPONENT n3 PORT (i1, i2, i3: IN BIT; o1: OUT BIT); END COMPONENT;
--  Component Configuration
FOR ALL : n1 USE ENTITY WORK.inv (single_delay);
FOR ALL : n2 USE ENTITY WORK.nand2 (single_delay);
FOR ALL : n3 USE ENTITY WORK.nand3 (single_delay);
--Intermediate signals
SIGNAL im1,im2, im3, im4, im5, im6, im7, im8, im9, im10 : BIT;
```

42

21

… Structural Model of Single-Bit Comparator

```
BEGIN
-- a_gt_b output
    g0 : n1 PORT MAP (a, im1);
    g1 : n1 PORT MAP (b, im2);
    g2 : n2 PORT MAP (a, im2, im3);
    g3 : n2 PORT MAP (a, gt, im4);
    g4 : n2 PORT MAP (im2, gt, im5);
    g5 : n3 PORT MAP (im3, im4, im5, a_gt_b);
--  a_eq_b output
    g6 : n3 PORT MAP (im1, im2, eq, im6);
    g7 : n3 PORT MAP (a, b, eq, im7);
    g8 : n2 PORT MAP (im6, im7, a_eq_b);
-- a_lt_b output
    g9 : n2  PORT MAP (im1, b, im8);
    g10 : n2 PORT MAP (im1, lt, im9);
    g11 : n2 PORT MAP (b, lt, im10);
    g12 : n3 PORT MAP (im8, im9, im10, a_lt_b);
END gate_level;
```

43

Netlist Description of Single-Bit Comparator

```
ARCHITECTURE netlist OF bit_comparator IS
SIGNAL im1,im2, im3, im4, im5, im6, im7, im8, im9, im10 : BIT;
BEGIN
-- a_gt_b output
    g0 : ENTITY Work.inv(single_delay)  PORT MAP (a, im1);
    g1 : ENTITY Work.inv(single_delay) PORT MAP (b, im2);
    g2 : ENTITY Work.nand2(single_delay) PORT MAP (a, im2, im3);
    g3 : ENTITY Work.nand2(single_delay) PORT MAP (a, gt, im4);
    g4 : ENTITY Work.nand2(single_delay) PORT MAP (im2, gt, im5);
    g5 : ENTITY Work.nand3(single_delay) PORT MAP (im3, im4, im5, a_gt_b);
--  a_eq_b output
    g6 : ENTITY Work.nand3(single_delay) PORT MAP (im1, im2, eq, im6);
    g7 : ENTITY Work.nand3(single_delay) PORT MAP (a, b, eq, im7);
    g8 : ENTITY Work.nand2(single_delay) PORT MAP (im6, im7, a_eq_b);
-- a_lt_b output
    g9 : ENTITY Work.nand2(single_delay) PORT MAP (im1, b, im8);
    g10 : ENTITY Work.nand2(single_delay) PORT MAP (im1, lt, im9);
    g11 : ENTITY Work.nand2(single_delay) PORT MAP (b, lt, im10);
    g12 : ENTITY Work.nand3(single_delay) PORT MAP (im8, im9, im10, a_lt_b);
END netlist;
```

44

4-Bit Comparator Iterative Structural Wiring: "For .... Generate"Statement...

```
ENTITY nibble_comparator IS
   PORT (a, b : IN BIT_VECTOR (3 DOWNTO 0); -- a and b data inputs
   gt, eq,  lt : IN BIT;  -- previous greater, equal & less than
   a_gt_b, a_eq_b, a_lt_b : OUT BIT);  -- a > b,  a = b, a < b
END nibble_comparator;
   --
ARCHITECTURE iterative OF nibble_comparator IS

   COMPONENT comp1
        PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
   END COMPONENT;
   FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
   SIGNAL im : BIT_VECTOR ( 0 TO 8);
BEGIN

   c0: comp1 PORT MAP (a(0), b(0), gt, eq, lt, im(0), im(1), im(2));
```

45

… 4-Bit Comparator: "For ……. Generate" Statement

```
   c1to2: FOR i IN 1 TO 2 GENERATE

        c: comp1 PORT MAP ( a(i), b(i), im(i*3-3), im(i*3-2), im(i*3-1),
   im(i*3+0), im(i*3+1), im(i*3+2) );

   END GENERATE;
   c3: comp1 PORT MAP (a(3), b(3), im(6), im(7), im(8), a_gt_b,
   a_eq_b, a_lt_b);

END iterative;
```

- USE BIT_VECTOR for Ports a & b
- Separate first and last bit-slices from others
- Arrays FOR intermediate signals facilitate iterative wiring
- Can easily expand to an n-bit comparator

46

23

4-Bit Comparator: "IF …… Generate" Statement …

```
ARCHITECTURE iterative OF nibble_comparator IS
   --
   COMPONENT comp1
        PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
   END COMPONENT;
   --
   FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
   CONSTANT n : INTEGER := 4;
   SIGNAL im :  BIT_VECTOR ( 0 TO  (n-1)*3-1);
   --
BEGIN
   c_all: FOR i IN 0 TO n-1 GENERATE


 l: IF i = 0 GENERATE
   least: comp1 PORT MAP (a(i), b(i), gt, eq, lt, im(0), im(1), im(2) );
   END GENERATE;
```

47

… 4-Bit Comparator: "IF …… Generate" Statement

```
--
  m: IF i = n-1 GENERATE
       most: comp1 PORT MAP (a(i), b(i), im(i*3-3), im(i*3-2),
                                  im(i*3-1), a_gt_b, a_eq_b, a_lt_b);
      END GENERATE;
--
   r: IF i > 0 AND i < n-1 GENERATE
   rest: comp1 PORT MAP (a(i), b(i), im(i*3-3), im(i*3-2),
                           im(i*3-1), im(i*3+0), im(i*3+1), im(i*3+2) );
   END GENERATE;
--
END GENERATE; -- Outer Generate
END iterative;
```

48

4-Bit Comparator: Alternative Architecture (Single Generate)

```
ARCHITECTURE Alt_iterative OF nibble_comparator IS
constant n: Positive :=4;
COMPONENT comp1
    PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
END COMPONENT;
FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
SIGNAL im : BIT_VECTOR ( 0 TO 3*n+2);
BEGIN
im(0 To 2) <= gt&eq&lt;
cALL: FOR i IN 0 TO n-1 GENERATE
c: comp1 PORT MAP (a(i), b(i), im(i*3), im(i*3+1), im(i*3+2),
im(i*3+3), im(i*3+4), im(i*3+5) );
END GENERATE;
a_gt_b <= im(3*n);
a_eq_b  <= im(3*n+1);
a_lt_b  <= im(3*n+2);
END Alt_iterative ;
```

49

Design Parameterization …

- GENERICs can pass design parameters
- GENERICs can include default values
- New versions of gate descriptions contain timing

```
ENTITY  inv_t  IS
GENERIC (tplh : TIME := 3 NS; tphl : TIME := 5 NS);
PORT (i1 : in BIT; o1 : out BIT);
END inv_t;
--
ARCHITECTURE average_delay OF inv_t IS
BEGIN
o1 <= NOT i1 AFTER (tplh + tphl) / 2;
END average_delay;
```

50

… Design Parameterization …

```
ENTITY nand2_t IS
GENERIC (tplh : TIME := 4 NS;
tphl : TIME := 6 NS);
PORT (i1, i2 : IN BIT; o1 : OUT
BIT);
END nand2_t;
--
ARCHITECTURE average_delay
OF nand2_t IS
BEGIN
o1 <= i1 NAND i2 AFTER (tplh +
tphl) / 2;
END average_delay;
```

```
ENTITY nand3_t IS
GENERIC (tplh : TIME := 5 NS;
tphl : TIME := 7 NS);
PORT (i1, i2, i3 : IN BIT; o1 :
OUT BIT);
END nand3_t;
--
ARCHITECTURE average_delay
OF nand3_t IS
BEGIN
o1 <= NOT ( i1 AND i2 AND i3 )
AFTER (tplh + tphl) / 2;
END average_delay;
```

51

Using Default values …

```
ARCHITECTURE default_delay OF bit_comparator IS
Component n1 PORT (i1: IN BIT; o1: OUT BIT);
END Component;
Component n2 PORT (i1, i2: IN BIT; o1: OUT BIT);
END Component;
Component n3 PORT (i1, i2, i3: IN BIT; o1: OUT BIT);
END Component;
FOR ALL : n1 USE ENTITY WORK.inv_t (average_delay);
FOR ALL : n2 USE ENTITY WORK.nand2_t (average_delay);
FOR ALL : n3 USE ENTITY WORK.nand3_t (average_delay);
-- Intermediate signals
SIGNAL im1,im2, im3, im4, im5, im6, im7, im8, im9, im10 : BIT;
BEGIN
-- a_gt_b output
g0 : n1 PORT MAP (a, im1);
g1 : n1 PORT MAP (b, im2);
g2 : n2 PORT MAP (a, im2, im3);
g3 : n2 PORT MAP (a, gt, im4);
g4 : n2 PORT MAP (im2, gt, im5);
g5 : n3 PORT MAP (im3, im4, im5, a_gt_b);
```

No Generics Specified in Component Declarations

52

26

… Using Default values

```
-- a_eq_b output
g6 : n3 PORT MAP (im1, im2, eq, im6);
g7 : n3 PORT MAP (a, b, eq, im7);
g8 : n2 PORT MAP (im6, im7, a_eq_b);
-- a_lt_b output
g9 : n2 PORT MAP (im1, b, im8);
g10 : n2 PORT MAP (im1, lt, im9);
g11 : n2 PORT MAP (b, lt, im10);
g12 : n3 PORT MAP (im8, im9, im10, a_lt_b);
END default_delay;
```

- *Component declarations do not contain **GENERIC**s*
- *Component instantiation are as before*
- *If default values exist, they are used*

53

Assigning Fixed Values to Generic Parameters …

```
ARCHITECTURE fixed_delay OF bit_comparator IS
Component n1
Generic (tplh, tphl : Time); Port (i1: in Bit; o1: out Bit);
END Component;
Component n2
Generic (tplh, tphl : Time); Port (i1, i2: in Bit; o1: out Bit);
END Component;
Component n3
Generic (tplh, tphl : Time); Port (i1, i2, i3: in Bit; o1: out Bit);
END Component;
FOR ALL : n1 USE ENTITY WORK.inv_t (average_delay);
FOR ALL : n2 USE ENTITY WORK.nand2_t (average_delay);
FOR ALL : n3 USE ENTITY WORK.nand3_t (average_delay);
-- Intermediate signals
SIGNAL im1,im2, im3, im4, im5, im6, im7, im8, im9, im10 : BIT;
BEGIN
-- a_gt_b output
g0 : n1 Generic Map (2 NS, 4 NS) Port Map (a, im1);
g1 : n1 Generic Map (2 NS, 4 NS) Port Map (b, im2);
g2 : n2 Generic Map (3 NS, 5 NS) Port Map (a, im2, im3);
```

54

… Assigning Fixed Values to Generic Parameters

```
g3 : n2 Generic Map (3 NS, 5 NS) Port Map P (a, gt, im4);
g4 : n2 Generic Map (3 NS, 5 NS) Port Map (im2, gt, im5);
g5 : n3 Generic Map (4 NS, 6 NS) Port Map (im3, im4, im5, a_gt_b);
-- a_eq_b output
g6 : n3 Generic Map (4 NS, 6 NS) Port Map (im1, im2, eq, im6);
g7 : n3 Generic Map (4 NS, 6 NS) PORT MAP (a, b, eq, im7);
g8 : n2 Generic Map (3 NS, 5 NS) PORT MAP (im6, im7, a_eq_b);
-- a_lt_b output
g9 : n2 Generic Map (3 NS, 5 NS) Port Map (im1, b, im8);
g10 : n2 Generic Map (3 NS, 5 NS) PORT MAP (im1, lt, im9);
g11 : n2 Generic Map (3 NS, 5 NS) PORT MAP (b, lt, im10);
g12 : n3 Generic Map (4 NS, 6 NS) PORT MAP (im8, im9, im10, a_lt_b);
END fixed_delay;
```

- *Component declarations contain **GENERIC**s*
- *Component instantiation contain **GENERIC Values***
- ***GENERIC Values** overwrite default values*

55

Instances with OPEN Parameter Association

```
ARCHITECTURE iterative OF
nibble_comparator IS
……..
BEGIN
c0: comp1
GENERIC MAP (Open, Open, 8
NS, Open, Open, 10 NS)
PORT MAP (a(0), b(0), gt, eq, lt,
im(0), im(1), im(2));
……..
END iterative;
```

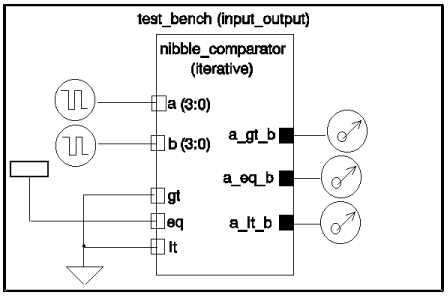
```
ARCHITECTURE iterative OF
nibble_comparator IS
……..
BEGIN
c0: comp1
GENERIC MAP (tplh3 => 8 NS,
tphl3 => 10 NS)
PORT MAP (a(0), b(0), gt, eq, lt,
im(0), im(1), im(2));
……..
END iterative;
```

- *A **GENERIC Map** may specify only some of the parameters*
- *Using **OPEN** causes use of default component values*
- *Alternatively, association by name can be used*
- *Same applies to **PORT MAP***

56

Structural Test Bench

- **A Testbench is an Entity without Ports that has a Structural Architecture**
- **The Testbench Architecture, in general, has 3 major components:**
  - Instance of the Entity Under Test (EUT)
  - Test Pattern Generator (Generates Test Inputs for the Input Ports of the EUT)
  - Response Evaluator (Compares the EUT Output Signals to the Expected Correct Output)



57

Testbench Example …

```
Entity nibble_comparator_test_bench IS
End nibble_comparator_test_bench ;
--
ARCHITECTURE input_output OF nibble_comparator_test_bench IS
--
COMPONENT comp4 PORT (a, b : IN bit_vector (3 DOWNTO 0);
gt, eq, lt : IN BIT;
a_gt_b, a_eq_b, a_lt_b : OUT BIT);
END COMPONENT;
--
FOR a1 : comp4 USE ENTITY WORK.nibble_comparator(iterative);
--
SIGNAL a, b : BIT_VECTOR (3 DOWNTO 0);
SIGNAL eql, lss, gtr, gnd : BIT;
SIGNAL vdd : BIT := '1';
--
BEGIN
a1: comp4 PORT MAP (a, b, gnd, vdd, gnd, gtr, eql, lss);
--
```

58

29

…Testbench Example

```
a2: a <= "0000",            -- a = b (steady state)
"1111" AFTER 0500 NS, -- a > b (worst case)
"1110" AFTER 1500 NS, -- a < b (worst case)
"1110" AFTER 2500 NS, -- a > b (need bit 1 info)
"1010" AFTER 3500 NS, -- a < b (need bit 2 info)
"0000" AFTER 4000 NS, -- a < b (steady state, prepare FOR next)
"1111" AFTER 4500 NS, -- a = b (worst case)
"0000" AFTER 5000 NS, -- a < b (need bit 3 only, best case)
"0000" AFTER 5500 NS, -- a = b (worst case)
"1111" AFTER 6000 NS; -- a > b (need bit 3 only, best case)
--
a3 : b <= "0000",           -- a = b (steady state)
"1110" AFTER 0500 NS, -- a > b (worst case)
"1111" AFTER 1500 NS, -- a < b (worst case)
"1100" AFTER 2500 NS, -- a > b (need bit 1 info)
"1100" AFTER 3500 NS, -- a < b (need bit 2 info)
"1101" AFTER 4000 NS, -- a < b (steady state, prepare FOR next)
"1111" AFTER 4500 NS, -- a = b (worst case)
"1110" AFTER 5000 NS, -- a < b (need bit 3 only, best case)
"0000" AFTER 5500 NS, -- a = b (worst case)
"0111" AFTER 6000 NS; -- a > b (need bit 3 only, best case)
END input_output;
```

59

VHDL Predefined Operators

- **Logical Operators: NOT, AND, OR, NAND, NOR, XOR, XNOR**
  - Operand Type: Bit, Boolean, Bit_vector
  - Result Type: Bit, Boolean, Bit_vector
- **Relational Operators: =, /=, <, <=, >, >=**
  - Operand Type: Any type
  - Result Type: Boolean
- **Arithmetic Operators: +, -, *, /**
  - Operand Type: Integer, Real
  - Result Type: Integer, Real
- **Concatenation Operator: &**
  - Operand Type: Arrays or elements of same type
  - Result Type: Arrays
- **Shift Operators: SLL, SRL, SLA, SRA, ROL, ROR**
  - Operand Type: Bit or Boolean vector
  - Result Type: same type

60

VHDL Reserved Words

abs	**disconnect**	**label**	**package**	
access	**downto**	**library**	**Poll**	**units**
after	**linkage**		**procedure**	**until**
alias	**else**	**loop**	**process**	**use**
all	**elsif**			**variable**
and	**end**	**map**	**range**	
architecture	**entity**	**mod**	**record**	**wait**
array	**exit**	**nand**	**register**	**when**
assert	**new**	**rem**	**while**	
attribute	**file**	**next**	**report**	**with**
begin	**for**	**nor**	**return**	**xor**
block	**function**	**not**	**select**	
body	**generate**	**null**	**severity**	
buffer	**generic**	**of**	**signal**	
bus	**guarded**	**on**	**subtype**	
case	**if**	**open**	**then**	
component	**in**	**or**	**to**	
configuration	**inout**	**others**	**transport**	
constant	**is**	**out**	**type**	

61

VHDL Language Grammar

- **Formal grammar of the IEEE Standard 1076-1993 VHDL language in BNF format**
  - http://www.iis.ee.ethz.ch/~zimmi/download/vhdl93_syntax.html

62

31

VHDL Objects …

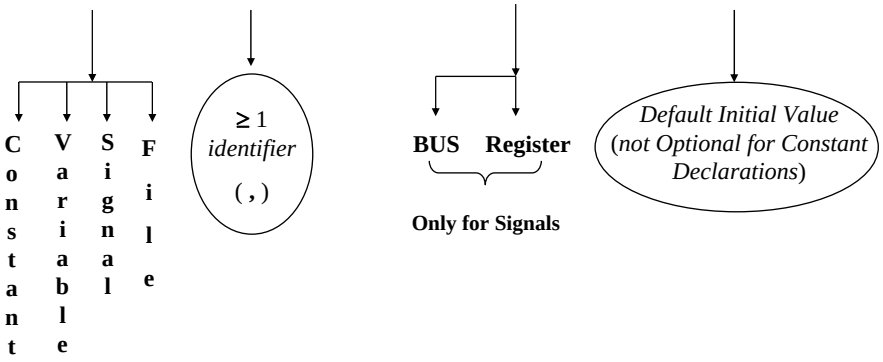
- **VHDL *OBJECT* : Something that can hold a value of a given** *Data Type***.**
- **VHDL has 3 classes of objects**
  - CONSTANTS
  - VARIABLES
  - SIGNALS
- **Every object & expression must unambiguously belong to one *named*** *Data Type*
- **Every object must be** *Declared.*

63

… VHDL Object …

Syntax

Obj_Class **<*id_list*> : Type/SubType [*signal_kind*] [:= *expression*];**



64

32

… VHDL Object …

- **Value of Constants *must* be specified when declared.**
- ***Initial* values of Variables or Signals *may* be specified when declared.**
- **If not explicitly specified*, Initial* values of Variables or Signals *default* to the value of the Left Element in the type range specified in the declaration.**
- **Examples:**
  - **Constant** Rom_Size **:** Integer := 2**16**;**
  - **Constant** Address_Field **:** Integer := 7**;**
  - **Constant** Ovfl_Msg **:** String (1 **To** 20) := ``Accumulator OverFlow``**;**
  - **Variable** Busy, Active **:** Boolean := False**;**
  - **Variable** Address **:** Bit_Vector (0 **To** Address_Field) := ``00000000``**;**
  - **Signal** Reset**:** Bit := `0`**;**

65

Variables vs. Signals

Variables & Signals

VARIABLES	SIGNALS
• Variables are only *Local* and May Only Appear within the Body of a Process or a SubProgram • Variable Declarations Are Not Allowed in Declarative Parts of Architecture Bodies or Blocks.	• Signals May be Local or Global. • Signals May not be Declared within Process or Subprogram Bodies. • All Port Declarations Are for Signals.
A Variable Has No HardWare Correspondence	A Signal Represents a Wire or a Group of Wires (BUS)
Variables Have No ***Time*** Dimension Associated With Them. (*Variable Assignment occurs instantaneously*)	Signals Have ***Time*** Dimension (*A Signal Assignment is Never Instantaneous* (Minimum Delay = δ Delay)
Variable Assignment Statement is always ***SEQUENTIAL***	Signal Assignment Statement is Mostly ***CONCURRENT*** (*Within Architectural* Body). It Can Be **SEQUENTIAL** (*Within Process Body*)
Variable Assignment Operator is :=	Signal Assignment Operator is <=

Variables Within Process Bodies are STATIC, i.e. a Variable Keeps its Value from One Process Call to Another. Variables Within Subprogram Bodies Are Dynamic, i.e. Variable Values are Not held from one Call to Another.

66

Signal Assignments …

- **Syntax:**

  Target Signal <= [Transport] ***Waveform*** **;**

  Waveform **:=** ***Waveform_element*** **{,** ***Waveform_element*** **}**

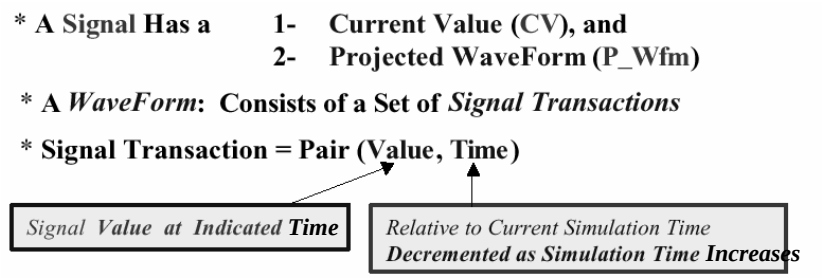
  Waveform_element **:=** ***Value_Expression*** **[** ***After Time_Expression*** **]**

- **Examples:**
  - X <= '0' ; -- Assignment executed After δ delay
  - S <= '1' After 10 ns;
  - Q <= Transport '1' After 10 ns;
  - S <= '1' After 5 ns, '0' After 10 ns, '1' After 15 ns;
- **Signal assignment statement**
  - mostly **concurrent** (within architecture bodies)
  - can be **sequential** (within process body)

67

… Signal Assignments

- **Concurrent signal assignments are order independent**
- **Sequential signal assignments are order dependent**
- **Concurrent signal assignments are executed**
  - Once *at the beginning* of simulation
  - Any time a signal on the right hand side changes



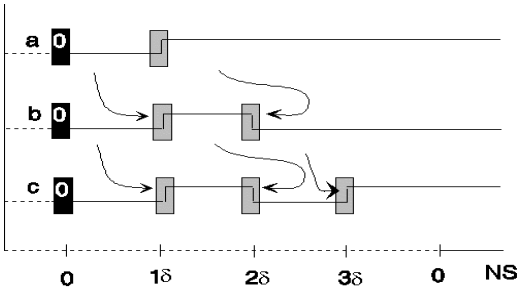
68

34

Delta Delay

- **If no Time Delay is *explicitly specified*, Signal assignment is executed after a δ-delay**
  - Delta is a simulation cycle , and not a real time
  - Delta is used for scheduling
  - A million deltas do not add to a femto second

```
ARCHITECTURE concurrent
OF timing_demo IS
SIGNAL a, b, c : BIT := '0';
BEGIN
a <= '1';
b <= NOT a;
c <= NOT b;
END concurrent;
```



69

Signal Attributes…

- **Attributes are named characteristics of an Object (or Type) which has a value that can be referenced.**
- **Signal Attributes**
  - S`Event **--** Is TRUE if Signal S has changed.
  - S`Stable(t) **--** Is TRUE if Signal S has not changed for the last ``**t**`` period. If t=0; it is written as S`Stable
  - S`Last_Value **--** Returns the previous value of S before the last change.
  - S`Active **-- --** Is TRUE if Signal S has had a transaction in the current simulation cycle.
  - S`Quiet(t) **-- --** Is TRUE if no transaction has been placed on Signal S for the last ``**t**`` period. If t=0; it is written as S`Quiet
  - S`Last_Event **--** Returns the amount of time since the last value change on S**.**

70

35

Subprograms…

- **Subprograms consist of functions and procedures.**
- **Subprograms are used to**
  - Simplify coding,
  - Achieve modularity,
  - Improve readability.
- **Functions return values and cannot alter values of their parameters.**
- **Procedures used as a statement and can alter values of their parameters.**
- **All statements inside a subprogram are sequential.**

71

…Subprograms

- **Subprograms**
  - Concurrent
  - Sequential
- **Concurrent subprograms exist outside of a process or another subprogram.**
- **Sequential subprograms exist in a process statement or another subprogram.**
- **A procedure exists as a separate statement in architecture or process.**
- **A function usually used in assignment statement or expression.**

72

Functions

- **Function specification:**
  - Name of the function
  - Formal parameters of the function
    - Name of the parameter
    - Type of the parameter
    - Mode IN is default & only allowed mode
    - Class constant is default
  - Return type of the function
  - Local declarations
- **A function body**
  - Must contain at least one return statement
  - May not contain a wait statement

73

A Left-Shift Function

```
Subtype Byte IS Bit_Vector (7 Downto 0);
Function SLL (V: Byte; N: Natural; Fill: Bit) Return Byte IS
   Variable Result: Byte := V;
Begin
   For I IN 1 To N Loop
        Result := Result (6 Downto 0) & Fill;
   End Loop;
   Return Result;
End SLL;
```

74

Using the Function

```
Architecture Functional Of LeftShifter IS
   Subtype Byte IS Bit_Vector (7 Downto 0);
   Function SLL (V: Byte; N: Natural; Fill: Bit) Return Byte is
      Variable Result: Byte := V;
   Begin
      For I IN 1 To N Loop
            Result := Result (6 Downto 0) & Fill;
      End Loop;
      Return Result;
   End SLL;

Begin
   Sout <= SLL(Sin, 1, '0') After 12 ns;
End Functional;
```

75

A Single-Bit Comparator

```
Entity Bit_Comparator IS
   Port ( a, b,                  -- data inputs
          gt,                    -- previous greater than
          eq,                    -- previous equal
          lt: IN BIT;            -- previous less than
     a_gt_b,                     -- greater
     a_eq_b,                     -- equal
     a_lt_b: OUT BIT);           -- less than
End Bit_Comparator;
```

$$a\_gt\_b = a \cdot gt + b` \cdot gt + a \cdot b`$$

$$a\_eq\_b = a \cdot b \cdot eq + a` \cdot b` \cdot eq$$

$$a\_lt\_b = b \cdot lt + a` \cdot lt + b \cdot a`$$

76

38

A Single-Bit Comparator using Functions

```
Architecture Functional of Bit_Comparator IS
   Function fgl (w, x, gl: BIT) Return BIT IS
     Begin
       Return (w AND gl) OR (NOT x AND gl) OR (w AND NOT x);
     End fgl;
   Function feq (w, x, eq: BIT) Return BIT IS
     Begin
       Return (w AND x AND eq) OR (NOT w AND NOT x AND eq);
     End feq;
Begin
   a_gt_b <= fgl (a, b, gt) after 12 ns;
   a_eq_b <= feq (a, b, eq) after 12 ns;
   a_lt_b  <= fgl (b, a, lt) after 12 ns;
End Functional;
```

77

Binary to Integer Conversion Function

```
Function To_Integer (Bin : BIT_VECTOR) Return Integer IS
   Variable Result: Integer;
Begin
   Result := 0;
   For I IN Bin`RANGE Loop
        If Bin(I) = '1' then
                Result := Result + 2**I;
        End if;
   End Loop;
   Return Result;
End To_Integer;
```

78

39

Procedure Specification

- **Name of the procedure**
- **Formal parameters of the procedure**
  - Class of the parameter
    - optional
    - defaults to constant
  - Name of the parameter
  - Mode of the parameter
    - optional
    - defaults to IN
  - Type of the parameter
- **Local declarations**

79

A Left-Shift Procedure

```
Subtype Byte is Bit_Vector (7 downto 0);
Procedure SLL (Signal Vin : In Byte; Signal Vout :out
Byte; N: Natural; Fill: Bit;
ShiftTime: Time) IS
    Variable Temp: Byte := Vin;
Begin
    For I IN 1 To N Loop
        Temp := Temp (6 downto 0) & Fill;
    End Loop;
    Vout <= Temp after N * ShiftTime;
End SLL;
```

80

Using the Procedure

```
Architecture Procedural of LeftShifter is
    Subtype Byte is Bit_Vector (7 downto 0);
    Procedure SLL (Signal Vin : In Byte; Signal Vout :out Byte; N: Natural;
    Fill: Bit; ShiftTime: Time) IS
        Variable Temp: Byte := Vin;
    Begin
        For I IN 1 To N Loop
                Temp := Temp (6 downto 0) & Fill;
        End Loop;
        Vout <= Temp after N * ShiftTime;
    End SLL;
Begin
    Process (Sin)
    Begin
        SLL(Sin, Sout, 1, '0', 12 ns) ;
    End process;
End Procedural;
```

81

Binary to Integer Conversion Procedure

```
Procedure Bin2Int (Bin : IN BIT_VECTOR;  Int: OUT Integer)  IS
    Variable Result: Integer;
Begin
    Result := 0;
    For I IN Bin`RANGE Loop
        If Bin(I) = '1' Then
                Result := Result + 2**I;
        End If;
    End Loop;
    Int := Result;
End Bin2Int;
```

82

41

Integer to Binary Conversion Procedure

```
Procedure Int2Bin (Int: IN Integer; Bin : OUT BIT_VECTOR)  IS
   Variable Tmp: Integer;
Begin
   Tmp := Int;
   For I IN 0 To (Bin`Length - 1) Loop
        If ( Tmp MOD 2 = 1) Then
                Bin(I) := '1';
        Else Bin(I) := '0';
        End If;
        Tmp := Tmp / 2;
   End Loop;
End Int2Bin;
```

83

Packages…

- A package is a common storage area used to hold data to be shared among a number of entities.
- Packages can encapsulate subprograms to be shared.
- A package consists of
  - Declaration section
  - Body section
- The package declaration section contains subprogram declarations, not bodies.
- The package body contains the subprograms' bodies.
- The package declaration defines the interface for the package.

84

…Packages

- **All items declared in the package declaration section are visible to any design unit that uses the package.**
- **A package is used by the USE clause.**
- **The interface to a package consists of any subprograms or deferred constants declared in the package declaration.**
- **The subprogram and deferred constant declarations must have a corresponding subprogram body and deferred constant value in the package body.**
- **Package body May contain other declarations needed solely within the package body.**
  - Not visible to external design units.

85

Package Declaration

- **The package declaration section can contain:**
  - Subprogram declaration
  - Type, subtype declaration
  - Constant, deferred constant declaration
  - Signal declaration creates a global signal
  - File declaration
  - Alias declaration
  - Component declaration
  - Attribute declaration, a user-defined attribute
  - Attribute specification
  - Use clause

86

Package Body

- **The package body main purpose is**
  - Define the values of deferred constants
  - Specify the subprogram bodies for subprograms declared in the package declaration
- **The package body can also contain:**
  - Subprogram declaration
  - Subprogram body
  - Type, subtype declaration
  - Constant declaration, which fills in the value for deferred constants
  - File declaration
  - Alias declaration
  - Use clause

87

Existing Packages

- **Standard Package**
  - Defines primitive types, subtypes, and functions.
  - e.g. Type Boolean IS (false, true);
  - e.g. Type Bit is (‘0’, ‘1’);
- **TEXTIO Package**
  - Defines types, procedures, and functions for standard text I/O from ASCII files.

88

Package Example for Component Declaration

```
Package simple_gates is
COMPONENT n1 PORT (i1: IN BIT; o1: OUT BIT); END COMPONENT ;
COMPONENT n2 PORT (i1,i2: IN BIT;o1:OUT BIT);END COMPONENT;
COMPONENT n3 PORT (i1, i2, i3: IN BIT; o1: OUT BIT); END COMPONENT;
end simple_gates;

Use work.simple_gates.all;
ENTITY bit_comparator IS
        PORT (a, b, gt,  eq,  lt : IN BIT;  a_gt_b,  a_eq_b, a_lt_b : OUT BIT);
END bit_comparator;
ARCHITECTURE gate_level OF bit_comparator IS
FOR ALL : n1 USE ENTITY WORK.inv (single_delay);
FOR ALL : n2 USE ENTITY WORK.nand2 (single_delay);
FOR ALL : n3 USE ENTITY WORK.nand3 (single_delay);
--Intermediate signals
SIGNAL im1,im2, im3, im4, im5, im6, im7, im8, im9, im10 : BIT;
BEGIN
-- description of architecture
END gate_level;
```

89

Package Example…

```
Package Shifters IS
   Subtype Byte IS Bit_Vector (7 Downto 0);
   Function SLL (V: Byte; N: Natural; Fill: Bit := '0') Return Byte;
   Function SRL (V: Byte; N: Natural; Fill: Bit := '0') Return Byte;
   Function SLA (V: Byte; N: Natural; Fill: Bit := '0') Return Byte;
   Function SRA (V: Byte; N: Natural) Return Byte;
   Function RLL (V: Byte; N: Natural) Return Byte;
   Function RRL (V: Byte; N: Natural) Return Byte;
End Shifters;
```

90

45

…Package Example…

```
Package Body Shifters IS
    Function SLL (V: Byte; N: Natural; Fill: Bit) Return Byte is
        Variable Result: Byte := V;
    Begin
        If N >= 8 Then
                Return (Others => Fill);
        End If;
        For I IN 1 To N Loop
                Result := Result (6 Downto 0) & Fill;
        End Loop;
        Return Result;
    End SLL;
        .
        .
        .
End Shifters;
```

91

…Package Example

```
USE WORK.Shifters.ALL
Architecture Functional of LeftShifter IS
Begin
    Sout <= SLL(Sin, 1, '0') After 12 ns;
End Functional;
```

92

46

Another Package Example…

```
Package Basic_Utilities IS
   Type Integers IS Array (0 to 5) of Integer;
   Function fgl (w, x, gl: BIT) Return BIT;
   Function feq (w, x, eq: BIT) Return BIT;
   Procedure Bin2Int (Bin : IN BIT_VECTOR;  Int: OUT Integer);
   Procedure Int2Bin (Int: IN Integer; Bin : OUT BIT_VECTOR);
   Procedure Apply_Data (
        Signal Target: OUT Bit_Vector (3 Downto 0);
        Constant Values: IN Integers;
        Constant Period: IN Time);
   Function To_Integer (Bin : BIT_VECTOR) Return Integer;

End Basic_Utilities;
```

93

…Another Package Example…

```
Package Body Basic_Utilities IS
   Function fgl (w, x, gl: BIT) Return BIT IS
     Begin
        Return (w AND gl) OR (NOT x AND gl) OR (w AND NOT x);
     End fgl;
   Function feq (w, x, eq: BIT) Return BIT IS
     Begin
        Return (w AND x AND eq) OR (NOT w AND NOT x AND eq);
     End feq;
        .
        .
        .
End Basic_Utilities;
```

94

…Another Package Example

```
USE WORK.Basic_Utilities.ALL
Architecture Functional of Bit_Comparator IS
Begin
   a_gt_b <= fgl (a, b, gt) after 12 ns;
   a_eq_b <= feq (a, b, eq) after 12 ns;
   a_lt_b  <= fgl (b, a, lt) after 12 ns;
End Functional;
```

95

Design Libraries…

- **VHDL supports the use of design libraries for categorizing components or utilities.**
- **Applications of libraries include**
  - Sharing of components between designers
  - Grouping components of standard logic families
  - Categorizing special-purpose utilities such as subprograms or types
- **Two Types of Libraries**
  - Working Library (WORK) {*A Predefined library into which a Design Unit is Placed after Compilation.*},
  - Resource Libraries {Contain design units that can be referenced within the design unit being compiled}.

96

… Design Libraries…

- Only one library can be the **Working** library
- Any number of Resource Libraries may be used by a Design Entity
- There is a number of predefined Resource Libraries
- The **Library** clause is used to make a given library visible
- The **Use** clause causes Package Declarations within a Library to be visible
- Library management tasks, e.g. Creation or Deletion, are not part of the VHDL Language Standard → Tool Dependent

97

… Design Libraries…

- **Exiting libraries**
  - STD Library
    - Contains the **STANDARD** and **TEXTIO** packages
    - Contains all the standard types & utilities
    - Visible to all designs
  - WORK library
    - Root library for the user
- **IEEE library**
  - Contains VHDL-related standards
  - Contains the std_logic_1164 (IEEE 1164.1) package
    - Defines a nine values logic system
    - De Facto Standard for all Synthesis Tools

98

…Design Libraries

- **To make a library visible to a design**
  - LIBRARY libname;
- **The following statement is assumed by all designs**
  - LIBRARY WORK;
- **To use the std_logic_1164 package**
  - LIBRARY IEEE
  - USE IEEE.std_logic_1164.ALL
- **By default, every design unit is assumed to contain the following declarations:**
  - LIBRARY STD , work ;
  - USE STD.Standard.All ;

99

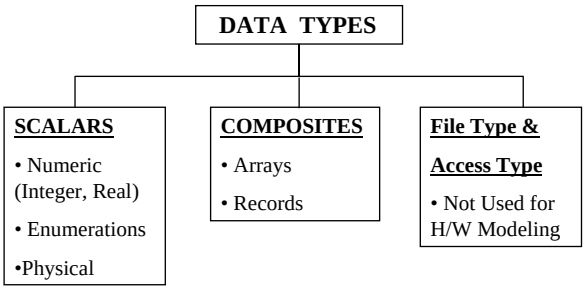
Arithmetic & Logical Operators for std_logic : Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity example is
port (a, b: IN std_logic_vector (7 downto 0));
end example;
architecture try of example is
signal x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12 : std_logic_vector (7 downto 0);
begin
      x1 <= not a;
      x2 <= a and b;
      x3 <= a nand b;
      x4 <= a or b;
      x5 <= a nor b;
      x6 <= a xor b;
      x7 <= a xnor b;
      x8 <= a + b;
      x9 <= a - b;
      x10 <= "+" (a, b);
end try;
```

100

Data Types

- **A Data Type defines a set of values & a set of operations.**
- **VHDL is a strongly-typed Language. Types cannot be mixed in Expressions or in assigning values to Objects in general**



101

Scalar Data Types

- **SYNTAX**
  - TYPE Identifier IS Type-Definition
- **Numeric Data Type**
  - Type-Definition is a Range_Constraint as follows:
  - Type-Definition := Range Initial-Value < To | DownTo> Final-Value
- **Examples**
  - TYPE address IS RANGE 0 To 127;
  - TYPE index IS RANGE 7 DownTo 0;
  - TYPE voltage IS RANGE -0.5 To 5.5;

102

51

Number Formats

- **Integers have no Decimal Point.**
- **Integers may be Signed or Unsigned (e.g. -5 356)**
- **A Real number must have either a Decimal Point, a -ive Exponent Term (Scientific Notation), or both.**
- **Real numbers may be Signed or Unsigned (e.g. -3.75 1E-9 1.5E-12)**
- **Based Numbers:**
  - Numbers Default to Base 10 (Decimal)
  - VHDL Allows Expressing Numbers Using Other Bases
  - Syntax
    - B#nnnn# -- Number nnnn is in Base B
  - Examples
    - 16#DF2# -- Base 16 Integer (HEX)
    - 8#7134# -- Base 8 Integer (OCTAL)
    - 2#10011# -- Base 2 Integer (Binary)
    - 16#65_3EB.37# -- Base 16 REAL (HEX)

103

Predefined Numeric Data Types

- **INTEGER -- Range is Machine limited but At Least $-(2^{31} - 1)$ To $(2^{31} - 1)$**
- **POSITIVE -- INTEGERS > 0**
- **NATURAL -- INTEGERS >= 0**
- **REAL -- Range is Machine limited**

104

52

Enumeration Data Type

- **Parenthesized ordered list of literals.**
  - Each may be an identifier or a character literal.
  - The list elements are separated by commas
- **A Position # is associated with each element in the List**
- **Position #`s begin with 0 for the Leftmost Element**
- **Variables & Signals of type ENUMERATION will have the leftmost element as their Default (Initial) value unless, otherwise explicitly assigned.**
- **Examples**
  - TYPE Color IS (Red, Orange, Yellow, Green, Blue, Indigo, Violet);
  - TYPE Tri_Level IS (`0`, `1`, `Z`);
  - TYPE Bus_Kind IS (Data, Address, Control);
  - TYPE state IS (Init, Xmit, Receive, Wait, Terminal);

105

Predefined Enumerated Data Types

- **TYPE BIT IS (`0` , `1`) ;**
- **TYPE BOOLEAN IS (False, True) ;**
- **TYPE CHARACTER IS (128 ASCII Chars.....) ;**
- **TYPE Severity_Level IS (Note, Warning, Error, Failure) ;**
- **TYPE Std_U_Logic IS (**

```
`U` , -- Uninitialized
`X` , -- Forcing Unknown
`0` , -- Forcing 0
`1` , -- Forcing 1
`Z` , -- High Impedence
`W` , -- Weak Unknown
`L` , -- Weak 0
`H` , -- Weak 1
`-` , -- Don`t Care
) ;
```

- **SUBTYPE Std_Logic IS resolved Std_U_Logic ;**

106

Physical Data Type

- **Specifies a Range Constraint , one Base Unit, and 0 or more secondary units.**
- **Base unit is indivisible, i.e. no fractional quantities of the Base Units are allowed.**
- **Secondary units must be integer multiple of the indivisible Base Unit.**
- **Examples**

```
TYPE   Resistance  IS   Range  1  To   Integer'High
  Units
     Ohm;        -- Base Unit
     Kohm = 1000  Ohm;     -- Secondary  Unit
     Mohm = 1000  Kohm;  -- Secondary  Unit
  end  Units ;
```

107

Predefined Physical Data Types

- **Time is the ONLY predefined Physical data type**

```
TYPE   Time  IS Range 0 To  1E20
   Units
      fs;    --       Base Unit (Femto Second = 1E-15 Second)
      ps = 1000  fs;    -- Pico_Second
      ns = 1000  ps;   -- Nano_Second
      us = 1000  ns;   -- Micro_Second
      ms = 1000  us;  -- Milli_Second
      sec = 1000  ms;-- Second
      min = 60  sec;   -- Minuite
      hr   = 60  min;    -- Hour
   end  Units ;
```

108

Composite Data Types: Arrays

- **Elements of an Array have the same data type**
- **Arrays may be Single/Multi - Dimensional**
- **Array bounds may be either Constrained or Unconstrained.**
- **Constrained Arrays**
  - Array Bounds Are Specified
  - Syntax:
    - **TYPE id Is Array (Range_Constraint) of Type;**
- **Examples**
  - TYPE word Is Array (0 To 7) of Bit;
  - TYPE pattern Is Array (31 DownTo 0) of Bit;
  - 2-D Arrays
    - **TYPE col Is Range 0 To 255;**
    - **TYPE row Is Range 0 To 1023;**
    - **TYPE Mem_Array Is Array (row, col) of Bit;**
    - **TYPE Memory Is Array (row) of word;**

109

Unconstrained Arrays

- **Array Bounds not specified through using the notation RANGE<>**
- **Type of each Dimension is specified, but the exact Range and Direction are not Specified.**
- **Useful in Interface_Lists → Allows Dynamic Sizing of Entities , e.g. Registers.**
- **Bounds of Unconstrained Arrays in such entities assume the Actual Array Sizes when wired to the Actual Signals.**
- **Example**
  - TYPE Screen Is Array (Integer Range<> , Integer Range<>) of BIT;

110

55

Predefined Array Types

- **Two UNCONSTRAINED Array Types are predefined**
- **BIT_VECTOR**
  - TYPE Bit_Vector Is Array (Natural Range<>) of Bit;
- **String**
  - TYPE String Is Array (Positive Range<>) of Character;
- **Example**
  - SUBTYPE Pixel Is Bit_Vector (7 DownTo 0);

111

DATA FLOW MODEL

- **Represents Register Transfer operations**
- **There is Direct Mapping between Data Flow Statements && Register Structural Model**
  - Implied Module Connectivity
  - Implied Muxes & Buses
- **Main Data Flow VHDL Constructs**
  - Concurrent Signal Assignment Statements
  - Block Statement

112

Signal Assignment …

- **Unconditional: Both Sequential & Concurrent.**
- **Conditional: Only Concurrent; Conditions must be Boolean, may overlap and need not be Exhaustive.**
- **Selected: Only Concurrent; Cases must not overlap and must be Exhaustive.**
- **Conditional Signal Assignment**

[Label:] target <= [Guarded] [Transport]
$Wave_1$ when $Cond_1$ Else
$Wave_2$ when $Cond_2$ Else
……..
$Wave_{n-1}$ when $Cond_{n-1}$ Else
$Wave_n$; -- Mandatory Wave

113

… Signal Assignment

- **Selected Signal Assignment**

With *Expression* **Select**
target <= [Guarded] [Transport]
$Wave_1$ **when** *$Choice_1$* **,**
$Wave_2$ **when** *$Choice_2$* **,**
……
$Wave_{n-1}$ **when** *$Choice_{n-1}$* **,**
$Wave_n$ **when OTHERS ;**

VHDL-93: Any *$Wave_i$* can be replaced by the Keyword UNAFFECTED (Which doesn't schedule any Transactions on the target signal.)

114

Signal Assignment Examples

Example: A 2x4 Decoder

```
Signal  D  :  Bit_Vector(1 To 4) := "0000";
Signal  S0, S1  :  Bit;
……
Decoder:  D <= "0001" after T  When S1='0' and S0='0' else
               "0010"  after T  When S1='0' else
               "0100" after T  When S0='0' else  "1000" ;
```

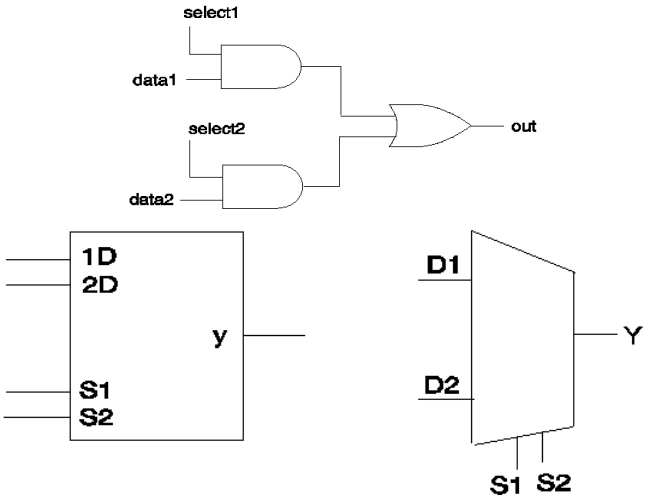
Example: 4-Phase Clock Generator

```
Signal Phi4  :  Bit_Vector(1 To 4) := "0000";
……
ClkGen:  With  Phi4  Select
                Phi4  <=  "1000" after T  When "0000" ,
                          "0100" after T  When "1000" ,
                          "0010" after T  When "0100" ,
                          "0001" after T  When "0010" ,
                          "1000" after T  When "0001" ,
                          "0000" When  Others ; -- Exhaustive
```

115

Multiplexing …

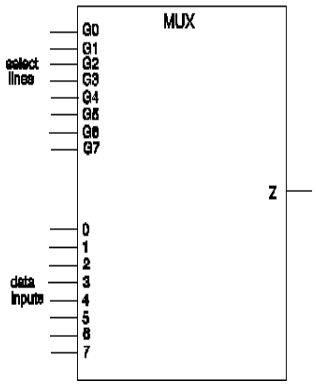
- **Multiplexers are used for data selection**



116

… Multiplexing

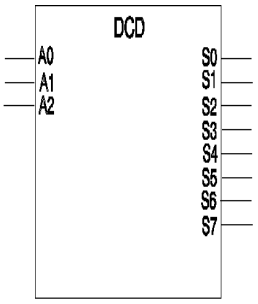
```
USE WORK.basic_utilities.ALL;
-- FROM PACKAGE USE: qit, qit_vector
ENTITY mux_8_to_1 IS
PORT (i7, i6, i5, i4, i3, i2, i1, i0 : IN qit;
s7, s6, s5, s4, s3, s2, s1, s0 : IN qit; z : OUT qit );
END mux_8_to_1;
--
ARCHITECTURE dataflow OF mux_8_to_1 IS
SIGNAL sel_lines : qit_vector ( 7 DOWNTO 0);
BEGIN
sel_lines <= s7&s6&s5&s4&s3&s2&s1&s0;
WITH sel_lines SELECT
z <= '0' AFTER 3 NS WHEN "00000000",
i7 AFTER 3 NS WHEN "10000000" | "Z0000000",
i6 AFTER 3 NS WHEN "01000000" | "0Z000000",
i5 AFTER 3 NS WHEN "00100000" | "00Z00000",
i4 AFTER 3 NS WHEN "00010000" | "000Z0000",
i3 AFTER 3 NS WHEN "00001000" | "0000Z000",
i2 AFTER 3 NS WHEN "00000100" | "00000Z00",
i1 AFTER 3 NS WHEN "00000010" | "000000Z0",
i0 AFTER 3 NS WHEN "00000001" | "0000000Z",
'X' WHEN OTHERS;
END dataflow;
```



117

3-to-8 Decoder

```
USE WORK.basic_utilities.ALL;
-- FROM PACKAGE USE : qit_vector
ENTITY dcd_3_to_8 IS
PORT (adr : IN qit_vector (2 DOWNTO 0);
so : OUT qit_vector (7 DOWNTO 0));
END dcd_3_to_8;
--
ARCHITECTURE dataflow OF dcd_3_to_8 IS
BEGIN
WITH adr SELECT
so <= "00000001" AFTER 2 NS WHEN "000",
"00000010" AFTER 2 NS WHEN "00Z" | "001",
"00000100" AFTER 2 NS WHEN "0Z0" | "010",
"00001000" AFTER 2 NS WHEN "0ZZ" | "0Z1" | "01Z" |
"011",
"00010000" AFTER 2 NS WHEN "100" | "Z00",
"00100000" AFTER 2 NS WHEN "Z0Z" | "Z01" | "10Z" |
"101",
"01000000" AFTER 2 NS WHEN "ZZ0" | "Z10" | "1Z0" |
"110",
"10000000" AFTER 2 NS WHEN "ZZZ" | "ZZ1" | "Z1Z" |
"Z11" | "1ZZ" | "1Z1" | "11Z" | "111",
"XXXXXXXX" WHEN OTHERS;
END dataflow;
```



118

59

Block Statement

- **Block statement is a Concurrent VHDL Construct which is used within an Architectural Body to group (Bind) a set of cConcurrent statements**

```
Block_Label: Block [ (Guard_Condition) ] [ IS ]
                    Block Header;
                    Block_Declarations;
             Begin
                    Concurrent_Statements;
             END Block  Block_Label  ;
```

- **A Guard Condition may be associated with a Block Statement to allow Enabling/Disabling of certain Signal Assignment statements.**
- **The Guard Condition defines an *Implicit Signal* called** *GUARD.*
- **In the simplest case, Binding (*Packing !*) statements within a** Block **has No Effect on the model.**
- **Blocks can be Nested.**

119

Block Statement Example

```
Architecture DF of D_Latch is
        Begin
        B : Block (Clk = `1`)
            Signal  I_State :Bit;  Block Local Signal
            Begin
                I_State <=  Guarded  D ;
                Q    <= I_State after 5 ns;
                QB  <= not I_State after 5 ns;
            END  Block B ;
END DF ;
```

- **UnGuarded Signal Targets (e.g.,** Q, QB**) are independent of the Guard Condition**

- **If *Guard Condition* (Clk=`1`) is TRUE, *Guarded Statements* within block are <u>Enabled</u> (<u>*Made Active*</u>)**
- **Guarded Statements (e.g.,** *I_State***) execute when**
  - **Guard Condition Becomes True, <u>AND</u>**
  - **While Guard Condition is True, a Signal on the RHS Changes Value**

120

Positive-Edge-Triggered DFF …

```
Library IEEE;
Use IEEE.Std_Logic_1164.ALL;
Entity DFF is
       Generic(TDel: Time:= 5 NS);
       Port(D, Clk: in Std_Logic; Q, QB: out Std_Logic);
End DFF;
```

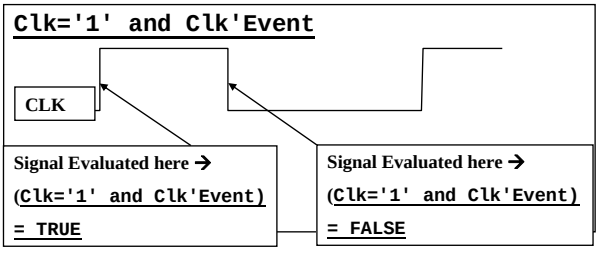
- *We will show several dataflow architectures with and without Block statement.*
- *Will show why some of these architectures do not work.*

121

… Positive-Edge-Triggered DFF …

```
Arch 1
Architecture DF1_NO_Block of DFF is
Signal I_State: Std_Logic:='1';
begin
       I_State <= D when (Clk='1' and
Clk'Event) else I_state;
       Q <= I_state after TDel ;
       QB <= not I_state after TDel ;
End  DF1_NO_Block ;
```

Works



Signal Evaluated 2-Times Per Clock Cycle

122

61

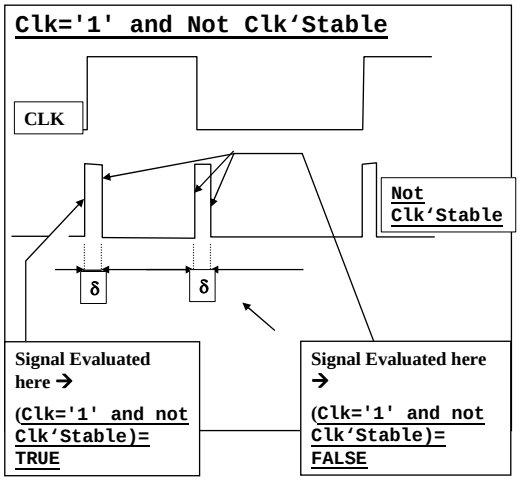
… Positive-Edge-Triggered DFF …

Doesn't Work

Arch 2

```
Architecture
DF2_NO_Block of DFF is
Signal I_State:
Std_Logic:='1';
begin
I_State <= D after TDel
when (Clk='1' and
(not(Clk'Stable))) else
I_state;
Q <= I_state;
QB <= not I_state;
End  DF2_NO_Block ;
```

Signal Evaluated 4-Times Per Clock Cycle



123

… Positive-Edge-Triggered DFF …

Works

Arch 3

```
Architecture DF3_NO_Block of DFF
is
Signal I_State: Std_Logic:='1';
begin
  I_State <= D when (Clk='1' and
(not(Clk'Stable))) else I_state;
  Q <= I_state after TDel;
  QB <= not I_state after TDel ;
End DF3_NO_Block ;
```

I_State gets the value of D after 1 delta and Its value does not get overwritten

124

62

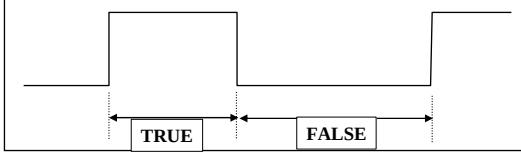
… Positive-Edge-Triggered DFF …

Arch4

```
Architecture DF1_Block of DFF is
Signal I_State: Std_Logic:='1';
begin
 D_Blk: Block(Clk='1' and Clk'Event)
    Begin
       Q  <= Guarded D after Tdel;
       QB <= Guarded not D after Tdel;
    End Block;
End  DF1_Block ;
```

Doesn't Work

GUARD <= Clk='1' and Clk'Event



Signal Evaluated Continuously while Clk = '1' !!!

125

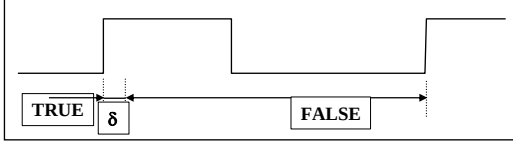
Positive-Edge-Triggered DFF …

Arch5

```
Architecture DF2_Block of DFF is
Signal I_State: Std_Logic:='1';
begin
 D_Blk: Block(Clk='1' and not Clk'Stable)
    Begin
       Q  <= Guarded D after Tdel;
       QB <= Guarded not D after Tdel;
    End Block;
End  DF2_Block ;
```

Works

GUARD <= Clk='1' and not Clk'Stable



Signal Evaluated Once Per Clock Cycle

(At Rising Edge of the Clock)

126

Use of Nested Blocks For Composite Enabling Conditions

```
ARCHITECTURE guarding OF DFF IS
    BEGIN
    edge: BLOCK ( c = '1' AND NOT c'STABLE )
    BEGIN
        gate: BLOCK ( e = '1' AND GUARD )
        BEGIN
        q <= GUARDED d AFTER delay1;
        qb <= GUARDED NOT d AFTER delay2;
        END BLOCK gate;
    END BLOCK edge;
    END guarding;
```

- Inner Guard Signal <= (e= '1') AND (c= '1' AND NOT c'STABLE)
- Can nest block statements
- Combining guard expressions must be done explicitly
- Implicit GUARD signals in each block

127

Data Flow Example …

Model A System with 2 8-Bit Registers R1 and R2, a 2-Bit Command signal "**COM**" and an external 8-Bit Input "**INP**"

- When Com= "00"→ R1 is Loaded with External Input
- When Com= "01"→ R2 is Loaded with External Input
- When Com= "10"→ R1 is Loaded with **R1+R2**
- When Com= "11"→ R1 is Loaded with **R1-R2**

```
Use Work.Utils_Pkg.ALL
Entity DF_Ex is
    Port (Clk: IN Bit; Com: IN Bit_Vector (1 DownTo
    0); INP: IN  Bit_Vector(7 DownTo 0));
End DF_Ex;
```

128

… Data Flow Example …

```
Architecture DF of DF_Ex is
Signal  Mux_R1, R1, R2, R2C, R2TC, Mux_Add,
        Sum: Bit_Vector(7 DownTo 0);
Signal  D00, D01, D10, D11, LD_R1: Bit;
Begin
   D00 <= not Com(0) and not Com(1); -- Decoder
   D01 <= not Com(0) and Com(1); -- Decoder
   D10 <= Com(0) and not Com(1); -- Decoder
   D11 <= Com(0) and Com(1); -- Decoder
   R2C  <= not R2;
   R2TC <= INC(R2C); -- Increment Function Defined  in the Package
   Mux_Add <=R2TC when D11 = '1' Else   R2 ;
```

129

… Data Flow Example

```
Sum <= ADD(R1,  Mux_Add); -- ADD Function-- Defined in Package
Mux_R1 <= INP when D00 = '1' Else  Sum;
R1E  <= D00 OR D10  OR D11;
Rising Edge: BLOCK(Clk='1' and not Clk'Stable)
   R1_Reg: BLOCK(R1E='1' AND GUARD)
        R1 <= Guarded Mux_R1 ;
   End Block R1_Reg ;

   R2_Reg: BLOCK(D01='1' AND GUARD)
        R2 <= Guarded INP ;
   End Block R2_Reg ;
End Block Rising Edge;
End DF;
```

130

Concurrent Versus Sequential Statements

Sequential Statements

- Used Within Process Bodies or SubPrograms
- Order Dependent
- Executed When Control is Transferred to the Sequential Body
  - **Assert**
  - **Signal Assignment**
  - **Procedure Call**
  - **Variable Assignment**
  - **IF Statements**
  - **Case Statement**
  - **Loops**
  - **Wait, Null, Next, Exit, Return**

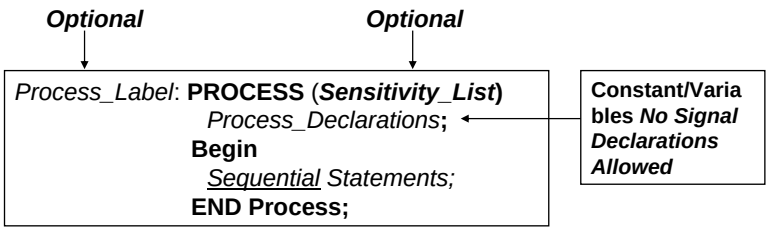
Concurrent Statements

- Used Within Architectural Bodies or Blocks
- Order Independent
- Executed Once *At the Beginning of Simulation* or Upon Some Triggered Event
  - Assert
  - **Signal Assignment**
  - Procedure Call (*None of Formal Parameters May be of Type Variable*)
  - Process
  - **Block Statement**
  - **Component Statement**
  - **Generate Statement**
  - **Instantiation Statement**

131

Process Statement …

- **Main construct for Behavioral Modeling.**
- **Other concurrent statements can be modeled by an equivalent process.**
- **Process statement is a Concurrent construct which performs a set of consecutive (Sequential) actions once it is activated. Thus, only sequential statements are allowed within the Process Body.**

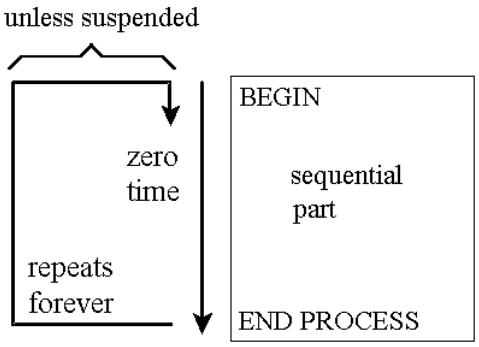


132

66

… Process Statement …

- **Unless sequential part is suspended**
  - It executes in zero real and delta time
  - It repeats itself forever



133

… Process Statement

- **Whenever a SIGNAL in the *Sensitivity_List* of the Process changes, the Process is ACTIVATED.**
- **After executing the last statement, the Process is SUSPENDED until one (or more) signal in the Process *Sensitivity_List* changes value where it will be REACTIVATED.**
- **A Process statement *Without* a *Sensitivity_List* is ALWAYS ACTIVE, i.e. after the last statement is executed, execution returns to the first statement and continues (*Infinite Looping*).**
- **It is ILLEGAL to Use WAIT-Statement Inside a Process which has a *Sensitivity_List* .**
- **In case no *Sensitivity_List* exists, a Process may be activated or suspended using the *WAIT*-Statement.**
- **Conditional and selective signal assignments are strictly concurrent and cannot be used in a process.**

134

Process Examples

```
Process
Begin
        A<= `1`;
        B <= `0`;
End Process;
```

Sequential Processing:

- **First A is Scheduled to have a value `1`**
- **Second B is Scheduled to have a value `0`**
- **A & B get their new values at the SAME TIME (1 Delta Time Later)**

```
Process
Begin
   A<= `1`;
   IF  (A= `1`) Then Action1;
   Else  Action2;
   End IF;
End Process;
```

Assuming a `0` Initial Value of A,

- **First A is Scheduled to Have a Value `1` One Delta Time Later**
- **Thus, Upon Execution of IF_Statement, A Has a Value of `0` and *Action 2* will be Taken.**
- **If A *was Declared as a Process* Variable, *Action1* Would Have Been Taken.**

135

Wait Statement

- **Syntax of Wait Statement**
  - WAIT**;** **-- Forever**
  - WAIT ON *Signal_List*; **-- On event on a signal**
  - WAIT UNTIL *Condition*; **-- until event makes condition true;**
  - WAIT FOR *Time_Out_Expression*;
  - WAIT FOR **0** *any_time_unit*; -- Process Suspended for 1 delta
- **When a WAIT-Statement is executed, the process** suspends **and conditions for its** Reactivation **are set.**
- **Process Reactivation conditions may be Mixed as follows**
  - **WAIT ON** *Signal_List* **UNTIL** *Condition* **FOR** *Time_Expression* **;**
  - **wait on** X,Y **until** (Z = 0) **for** 70 NS**; --** Process Resumes After 70 NS **OR** (in Case X or Y Changes Value and Z=0 is True) ***Whichever Occurs First***
  - Process Reactivated IF:
    - Event Occurred on the *Signal_List* while the *Condition* is True*, OR*
    - *Wait* Period Exceeds ``*Time_Expression* ``

136

68

Positive Edge-Triggered D-FF Examples

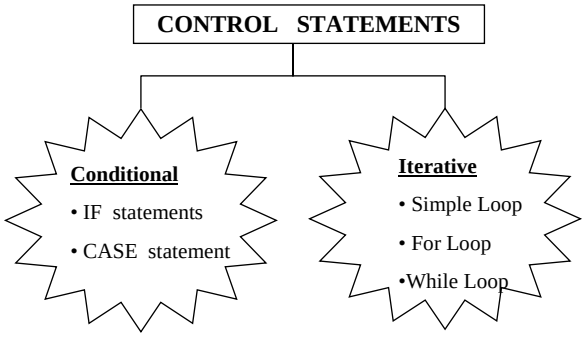
```
D_FF: PROCESS (CLK)
      Begin
        IF (CLK`Event and CLK = `1`) Then
            Q <= D After TDelay;
        END IF;
      END Process;
```

```
D_FF: PROCESS      -- No Sensitivity_List
      Begin
        WAIT UNTIL  CLK = `1`;
            Q <= D After TDelay;
      END Process;
```

```
D_FF: PROCESS (Clk, Clr)  -- FF With Asynchronous Clear
            Begin
                  IF   Clr= `1`   Then  Q <= `0` After TD0;
                  ELSIF (CLK`Event and CLK = `1`) Then   Q <= D After TD1;
                  END IF;
            END Process;
```

137

Sequential Statements



138

69

Conditional Control – IF Statement

- **Syntax: 3-Possible Forms**

(i) **IF** *condition* **Then**
*statement**s***;
End IF;

(ii) **IF** *condition* **Then**
statements;
Else
statements;
End IF;

(iii) **IF** *condition* **Then**
statements;
Elsif *condition* **Then**
statements;
…….
Elsif *condition* **Then**
statements;
End IF;

139

Conditional Control – Case Statement

- **Syntax:**

(i) **CASE** *Expression* **is**
when value **=>** *statement**s***;
when value*1* | value*2*| ...|value*n* **=>** *statement**s***;
when *discrete range of values* **=>** *statement**s***;
when **others** **=>** *statement**s***;
End CASE;

- Values/Choices should not overlap (*Any value of the Expression should evaluate to only one arm of the case statement*).
- All possible choices for the *Expression* should be accounted for Exactly Once.

140

Conditional Control – Case Statement

- If ``**others**`` is used, It must be the last ``arm`` of the CASE statement.
- There can be any number of arms in Any Order (*Except for the **others** arm which should be Last*).

```
CASE x is
        when 1 => y :=0;
        when 2 | 3 => y :=1;
        when 4 to 7 => y :=2;
        when others => y :=3;
End CASE;
```

141

Loop Control …

- **Simple Loops**
- **Syntax:**

[Loop_Label:] LOOP

statements;

End LOOP [Loop_Label];

- **The Loop_Label is Optional.**
- **The exit statement may be used to exit the Loop. It has two possible Forms:**
  - exit [Loop_Label]; -- This may be used in an if statement
  - exit [Loop_Label] when condition;

142

71

…Loop Control

```
Process
   variable A : Integer :=0;
   variable B : Integer :=1;
Begin
   Loop1: LOOP
                A := A + 1;
                B := 20;
                Loop2: LOOP
                          IF B < (A * A) Then
                             exit Loop2;
                          End IF;
                          B := B - A;
                          End LOOP Loop2;
                exit Loop1 when  A > 10;
          End  LOOP Loop1;
End Process;
```

143

FOR Loop

- **Syntax:**

Need Not Be Declared

[Loop_Label]: **FOR** *Loop_Variable* **in** *range* **LOOP**
statements;
End LOOP *Loop_Label*;

```
Process
          variable B : Integer :=1;
Begin
          Loop1: FOR  A  in  1  TO  10   LOOP
                              B := 20;
                              Loop2: LOOP
                                        IF B < (A * A) Then
                                             exit Loop2;
                                        End IF;
                                        B := B - A;
                                       End LOOP Loop2;
                         End  LOOP Loop1;
End Process;
```

144

WHILE Loop

- **Syntax:**

[Loop_Label]: **WHILE** *condition* **LOOP**
statements;
End LOOP *Loop_Label*;

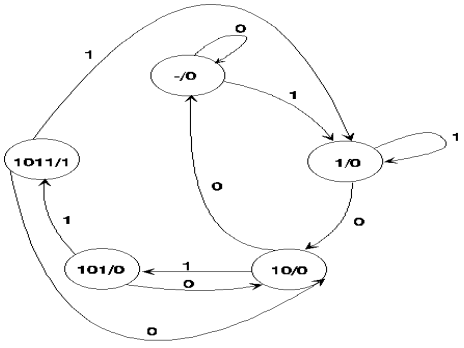
```
Process
        variable B:Integer :=1;
Begin
        Loop1: FOR  A  in  1  TO  10   LOOP
                        B := 20;
                        Loop2: WHILE  B < (A * A)  LOOP
                                B := B - A;
                                End LOOP Loop2;
                End  LOOP Loop1;
End Process;
```

145

A Moore 1011 Detector using Wait

```
ENTITY moore_detector IS
PORT (x, clk : IN BIT;
z : OUT BIT);
END moore_detector;
```

•*Can use WAIT in a Process statement to check for events on clk.*




```
ARCHITECTURE behavioral_state_machine OF moore_detector IS
TYPE state IS (reset, got1, got10, got101, got1011);
SIGNAL current : state := reset;
BEGIN
```

146

73

A Moore 1011 Detector using Wait

```
PROCESS
BEGIN
CASE current IS
WHEN reset =>  WAIT UNTIL clk = '1';
    IF x = '1' THEN current <= got1; ELSE current <= reset;  END IF;
WHEN got1 =>  WAIT UNTIL clk = '1';
    IF x = '0' THEN current <= got10; ELSE current <= got1;  END IF;
WHEN got10 => WAIT UNTIL clk = '1';
    IF x = '1' THEN current <= got101; ELSE current <= reset;  END IF;
WHEN got101 => WAIT UNTIL clk = '1';
    IF x = '1' THEN current <= got1011; ELSE current <= got10;  END IF;
WHEN got1011 => z <= '1';  WAIT UNTIL clk = '1';
    IF x = '1' THEN current <= got1; ELSE current <= got10;  END IF;
END CASE;
WAIT FOR 1 NS; z <= '0';
END PROCESS;
END behavioral_state_machine;
```

147

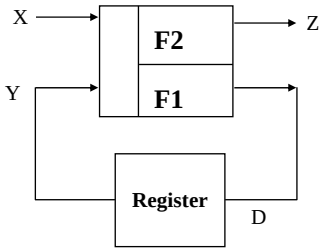
A Moore 1011 Detector without Wait

```
ARCHITECTURE most_behavioral_state_machine OF moore_detector IS
TYPE state IS (reset, got1, got10, got101, got1011);
SIGNAL current : state := reset;
BEGIN
PROCESS (clk)
BEGIN
IF (clk = '1' and CLK'Event) THEN
CASE current IS
WHEN reset =>
    IF x = '1' THEN current <= got1; ELSE current <= reset; END IF;
WHEN got1 =>
    IF x = '0' THEN current <= got10; ELSE current <= got1; END IF;
WHEN got10 =>
    IF x = '1' THEN current <= got101; ELSE current <= reset; END IF;
WHEN got101 =>
    IF x = '1' THEN current <= got1011; ELSE current <= got10; END IF;
WHEN got1011 =>
    IF x = '1' THEN current <= got1; ELSE current <= got10; END IF;
END CASE;
END IF;
END PROCESS;
z <= '1' WHEN current = got1011 ELSE '0';
END most_behavioral_state_machine;
```

148

Generalized VHDL Mealy Model

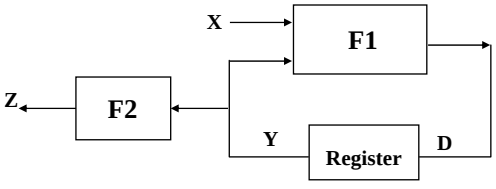
```
Architecture  Mealy of  fsm  is
    Signal D, Y: Std_Logic_Vector( ...); -- Local Signals
Begin
Register: Process( Clk)
   Begin
     IF (Clk`EVENT and Clk = `1`)  Then  Y <= D;
     End IF;
   End Process;
Transitions: Process(X, Y)
   Begin
     D <= F1(X, Y);
   End Process;
Output: Process(X, Y)
   Begin
     Z <= F2(X, Y);
   End Process;
End Mealy;
```



149

Generalized VHDL Moore Model

```
Architecture  Moore of  fsm  is
    Signal D, Y: Std_Logic_Vector( ...); -- Local Signals
Begin
Register: Process( Clk)
   Begin
     IF (Clk`EVENT and Clk = `1`)  Then  Y <= D;
     End IF;
   End Process;
Transitions: Process(X, Y)
   Begin
     D <= F1(X, Y);
   End Process;
Output: Process(Y)
   Begin
     Z <= F2(Y);
   End Process;
End Moore;
```



150

75

FSM Example …

```
Entity   fsm  is
 port ( Clk, Reset    : in   Std_Logic;
              X              : in  Std_Logic_Vector(0 to 1);
              Z              : out Std_Logic_Vector(1 downto 0));
End   fsm;
```

```
Architecture  behavior of fsm  is
          Type States is  (st0, st1, st2, st3);
          Signal Present_State, Next_State : States;
Begin
   register: Process(Reset, Clk)
   Begin
            IF Reset = `1`  Then
                     Present_State <= st0; --  Machine Reset to st0
            elsIF (Clk`EVENT and Clk = `1`)  Then
                     Present_State <= Next_state;
            End IF;
   End Process;
```

151

… FSM Example

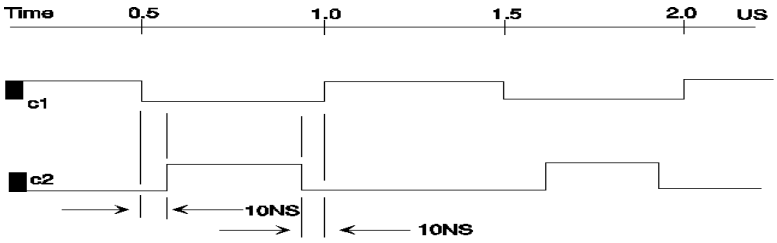
```
Transitions: Process(Present_State, X)
   Begin
            CASE Present_State is
                     when  st0  =>
                              Z <= ``00``;
                              IF X = ``11`` Then Next_State <= st0;
                              else Next_State <= st1; End IF;
                     when  st1  =>
                              Z <= ``01``;
                              IF X = ``11`` Then Next_State <= st0;
                              else Next_State <= st2; End IF;
                     when  st2  =>
                              Z <= ``10``;
                              IF X = ``11`` Then Next_State <= st2;
                              else Next_State <= st3; End IF;
                     when  st3  =>
                              Z <= ``11``;
                              IF X = ``11`` Then Next_State <= st3;
                              else Next_State <= st0; End IF;
            End CASE;
 End Process;
End behavior;
```

152

76

Using Wait for Two-Phase Clocking

```
c1 <= not c1 after 500ns;
phase2: PROCESS
BEGIN
WAIT UNTIL c1 = '0';
WAIT FOR 10 NS;
c2 <= '1';
WAIT FOR 480 NS;
c2 <= '0';
END PROCESS phase2;
. . .
```



153

77