COE 405
Design Organization and Parameterization

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

Outline

- **Subprograms**
  - Functions
  - Procedures
- **Examples of functions**
- **Example of procedures**
- **Packages**
  - Package declaration section
  - Package body
- **Design libraries**
  - VHDL standard packages
  - IEEE Std_Logic 1164 package
- **Design Parameterization**
- **Design Configuration**

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.**

…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.**

Functions

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

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;
```

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;
```


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;
```

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

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

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

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;
```


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;
```

Execution of a Function

- **Actual parameters are evaluated.**
- **Actuals are associated with their formals.**
- **Sequential statements are executed in order.**
- **Function exits when a return statement is executed.**
- **Return must supply a value of the return type.**
- **Direct and indirect recursion is allowed.**
- **Argument modification is not allowed.**
- **Execution of a wait statement is not allowed.**

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**

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;
```


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;
```

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;
```


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;
```

Using Procedures in a Test Bench …

```
Architecture Procedural of Nibble_Comparator_Test 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;
   Signal a, b: Bit_Vector (3 downto 0);
   Signal eql, gtr, lse: BIT;
   Signal Vdd: BIT := '1';
   Signal GND: BIT := '0';
   Type Integers IS Array (0 to 5) of Integer;
   Procedure Apply_Data (
      Signal Target: OUT Bit_Vector (3 Downto 0);
      Constant Values: IN Integers;
      Constant Period: IN Time) IS
      Variable Buf: Bit_Vector (3 Downto 0);
```


…Using Procedures in a Test Bench

```
   Begin
      For I IN 0 To 5 Loop
            Int2Bin (Values(I), Buf);
            Target <= Transport Buf After I * Period;
      End Loop;
   End Apply_Data;
Begin
   a1: Comp4 Port Map (a, b, GND, Vdd, GND, gtr, eql, lss);
   Apply_Data (a, 00&15&14&12&10&01, 500 ns);
   Apply_Data (b, 00&14&15&06&10&02, 500 ns);
End Procedural;
```

Executing a Procedure

- **Actual parameters are evaluated.**
- **Actuals are associated with their formals.**
- **Sequential statements are executed in order.**
- **Procedure exits when**
  - End of the procedure is reached
  - A return statement is executed; no value allowed
- **Direct and indirect recursion is allowed.**
- **Argument modification is allowed.**
- **Execution of a wait statement is allowed.**

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.**

…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.

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

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

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;
```

… Package Example for Component Declaration

```
USE WORK.simple_gates.ALL  ;
-- is equaivalent to:
USE
WORK.simple_gates.n1  ,
WORK.simple_gates.n2  ,
WORK.simple_gates.n3  ;
-- n1, n2 and n3 component declarations are visible
```


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;
```

…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;
```


…Package Example

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

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;
```


…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;
```

…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;
```


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}.

… 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

… Design Libraries…

- **Exiting libraries**
  - STD Library
    - Contains the STANDARD and TEXTIO packages (*See Appendix F in the Textbook*).
    - 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 (*See Appendix G in the Textbook*).

…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 ;

Standard Package…

- **Defines primitive types, subtypes, and functions.**
- **The package STANDARD is usually integrated directly in the simulation or synthesis program.**
- **It contains all basic types: Boolean, bit, bit_vector, character, integer, and the like.**
- **Additional logical, comparison and arithmetic operators are defined for these types within the package.**
- **The package STANDARD does not have to be explicitly included by the use statement.**

…Standard Package…

```
Package Standard IS
    Type Boolean IS (false, true);
    Type Bit is ('0', '1');
    Type Character IS ( nul, soh, stx, ……);
    Type Sensitivity_level IS (Note, Warning, Error, Failure);
    Type Integer IS Range –2147483648 to 2147483647;
    Type Real IS Range –1.0E308 to 1.0E308;
    Subtype Natural IS Integer Range 0 to Integer`High;
    Subtype Positive IS Integer Range 1 to Integer`High;
    Type String IS Array (positive Range <>) of Character;
```


…Standard Package

```
    Type Bit_Vector IS Array (Natural Range <>) of Bit;
    Type Time IS Range –2147483647 to 2147483647
        Units
                fs; ps = 1000 fs;  ns = 1000 ps; us = 1000 ns; ms = 1000
                us; sec = 1000 ms; min = 60 sec; hr = 60 min;
        End Units;
    Subtype Delay_length IS Time Range 0 fs to Time`High;
    Impure Function Now Return Delay_Length;
        .
        .
        .
End Standard;
```

TEXTIO Package…

- Defines types, procedures, and functions for standard text I/O from ASCII files.
- This package is also a part of the library STD.
- It is not included in every VHDL description by default.
- Therefore, if required, it has to be included by the statement USE STD.TEXTIO.all;.

…TEXTIO Package

```
Package TEXTIO IS
        Type Line IS Access String;
        Type Text IS File of String;
        Type Side IS (Right, Left);
        Subtype Width IS Natural;
        File Input: Text Open Read_Mode IS “STD_INPUT”;
        File Output: Text Open Write_Mode IS “STD_OUTPUT”;
        Procedure Readline (File F: Text; L: Out Line);
        Procedure Writeline (File F: Text; L: Inout Line);
        Function Endfile (File F: Text) Return Boolean;

    End TEXTIO;
```

Example using TEXTIO Package…

```
USE STD.TEXTIO.ALL;
Entity Square IS
    Port (Go : IN Bit);
END Square;
Architecture Simple of Square IS
Begin
    Process (Go)
        File Infile : Text IS IN “example1”;
        File Outfile : Text IS OUT “Outfile1”;
        Variable Out_Line, My_Line : Line;
        Variable Int_Val : Integer;
    Begin
        While Not ( Endfile(Infile) )  Loop
                Readline( Infile, My_Line);  -- read a line from the input file
```


…Example using TEXTIO Package

```
        Read( My_Line, Int_Val);          -- read a value from the line
        Int_Val := Int_Val ** 2;          -- square the value
        Write( Out_Line, Int_Val);        -- write the squared value to the line
        WriteLine( Outfile, Out_Line);    -- write the line to the output file
    End Loop;
  End Process;
End Simple;
```

Example using TEXTIO Package…

```
USE STD.TEXTIO.ALL;
Entity RCA_test is
End;
Architecture Test of RCA_test is
Component RCA
Generic (N: Integer := 4);
Port (  A, B : IN Bit_Vector(N-1 Downto 0);
    Cin : IN Bit;
    Sum : OUT Bit_Vector(N-1 Downto 0);
    Cout : OUT BIT
     );
End Component;
Constant N: Positive :=4;
Constant k: Positive :=2**N-1;
Type Integers IS Array (0 to k) of Integer;
```


…Example using TEXTIO Package…

```
Procedure Int2Bin (Int: IN Integer; Bin : OUT BIT_VECTOR)  IS
    Variable Tmp: Integer;
    Constant size: Natural := Bin'length;
Begin
    Tmp := Int;
    if (Tmp < 0) Then
        Tmp :=2**size+Tmp;
    End If;
    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;
Procedure Apply_Data (
        Signal Target: OUT Bit_Vector;
        Constant Values: IN Integers;
        Constant Period: IN Time) IS
        Variable Buf: Bit_Vector(Target'range);
```

…Example using TEXTIO Package…

```
Begin
        For I IN 0 To Values'length-1 Loop
                Int2Bin (Values(I), Buf);
                Target <= Transport Buf After I * Period;
        End Loop;
End Apply_Data;
Signal A, B, Sum: Bit_Vector(N-1 Downto 0);
Signal Cin, Cout: Bit;
Signal First, Second: Integers;
Begin
Process
File Infile : Text IS IN "infile";
Variable Out_Line, My_Line : Line;
Variable val : Integer;
Variable i: integer :=0;
```


…Example using TEXTIO Package

```
Begin
While Not ( Endfile(Infile) )  Loop
   Readline( Infile, My_Line);  -- read a line from the input file
   Read( My_Line, val);          -- read a value from the line
   First(i)<= val;
   Read( My_Line, val);          -- read a value from the line
   Second(i)<= val;
   i := i + 1;
End Loop;
wait;
End Process;
Apply_data(A, First, 100 ns);
Apply_data(B, Second, 100 ns);
CUT: RCA Generic Map (N) Port Map (A, B, Cin, Sum, Cout);
End;
```

Package IEEE.Std_Logic_1164 …

Package Std_Logic_1164 IS

```
-- logic state system (unresolved)
Type Std_Ulogic IS
( 'U',      -- Uninitialized        {Important For Sequential Systems}
  'X',      -- Forcing Unknown      {Contention}
  '0',      -- Forcing 0
  '1',      -- Forcing 1
 'Z',       -- High Impedance
 'W',       -- Weak Unknown
  'L',      -- Weak 0 {,e.g. Logic 0 Held Dynamically on a Capacitor}
  'H',      -- Weak 1 {,e.g. Logic 1 Held Dynamically on a Capacitor}
 '-');-- Don't Care    {Used To Optimize Synthesis}

-- unconstrained array of std_ulogic
Type Std_Ulogic_Vector IS Array ( Natural Range <>) of Std_Ulogic;
```


…Package IEEE.Std_Logic_1164 …

- ***std_ulogic*** *is an Unresolved 9-Valued System*
- *The resolved function resolves* ***std_ulogic*** *values*
- *The* ***std_logic*** *and* ***std_logic_vector*** *are de-facto industry standard*
- *Std_logic is a Resolved 9-valued System*
- *The* ***std_logic*** *is a Superset of most common value systems*

Resolving Multiple Values on the Same Signal Driver:

1. *Strong Dominates Weak*
2. *Weak Dominates Z*
3. *Conflicts of Equal Strength is Unknown at That Strength*

…Package IEEE.Std_Logic_1164 …

```
-- Resolution function
Function Resolved ( S: Std_Ulogic_Vector)  Return Std_Ulogic;
-- Industry standard logic type
Subtype Std_Logic IS Resolved Std_Ulogic;
-- Unconstrained array of Std_Logic
Type Std_Logic_Vector IS Array ( Natueal Range <>) of Std_Logic;
-- Common subtypes
Subtype X01 IS Resolved Std_Ulogic Range 'X' To '1'; -- ('X', '0', '1');
Subtype X01Z IS Resolved Std_Ulogic Range 'X' To 'Z'; -- ('X', '0', '1', 'Z');
Subtype UX01 IS Resolved Std_Ulogic Range 'U' To '1'; -- ('U', 'X', '0', '1');
Subtype UX01Z IS Resolved Std_Ulogic Range 'U' To 'Z'; -- ('U', 'X', '0',
'1', 'Z');
```


…Package IEEE.Std_Logic_1164 …

```
-- Overloaded  logical operators
Function "AND" (l: Std_Ulogic; R: Std_Ulogic) Return UX01;
Function "NAND" (l: Std_Ulogic; R: Std_Ulogic) Return UX01;
Function "OR" (l: Std_Ulogic; R: Std_Ulogic) Return UX01;
Function "NOR" (l: Std_Ulogic; R: Std_Ulogic) Return UX01;
Function "XOR" (l: Std_Ulogic; R: Std_Ulogic) Return UX01;
Function "XNOR" (l: Std_Ulogic; R: Std_Ulogic) Return UX01;
Function "NOT" (l: Std_Ulogic ) Return UX01;

-- Vectorized overloaded  logical operators
 Function "AND" (l, R: Std_Logic_Vector) Return Std_Logic_Vector;
 Function "AND" (l, R: Std_Ulogic_Vector) Return Std_Ulogic_Vector;
                .
                .
                .
```

…Package IEEE.Std_Logic_1164 …

```
-- Conversion functions
Function To_Bit (S: Std_Ulogic; Xmap: Bit := '0') Return Bit;
Function To_BitVector (S: Std_Logic_Vector; Xmap: Bit := '0') Return
    Bit_Vector;
Function To_StdLogicVector (B: Bit_Vector) Return Std_Logic_Vector;

-- Strength strippers and type converters
    Function To_X01 (S: Std_Logic_Vector)  Return Std_Logic_Vector;
    Function To_X01 (S: Bit_Vector)  Return Std_Logic_Vector;
    Function To_X01 (S: Bit )  Return X01;
    Function To_X01Z (S: Std_Logic_Vector)  Return Std_Logic_Vector;
    Function To_X01Z (S: Bit_Vector)  Return Std_Logic_Vector;
    Function To_X01Z (S: Bit )  Return X01Z;
    Function To_UX01 (S: Std_Logic_Vector)  Return Std_Logic_Vector;
    Function To_UX01 (S: Bit_Vector)  Return Std_Logic_Vector;
    Function To_UX01 (S: Bit )  Return UX01;
```


…Package IEEE.Std_Logic_1164 …

```
    -- Edge detection
    Function Rising_Edge (Signal S: Std_Ulogic) Return Boolean;
    Function Falling_Edge (Signal S: Std_Ulogic) Return Boolean;

    -- Object contains unknown
    Function Is_X  (Signal S: Std_Ulogic_Vector) Return Boolean;
    Function Is_X  (Signal S: Std_Logic_Vector) Return Boolean;
    Function Is_X  (Signal S: Std_Ulogic) Return Boolean;

End Std_Logic_1164;
```

Package Body: IEEE.Std_Logic_1164

Package Body Std_Logic_1164 IS

```
Type StdLogic_Table IS ARRAY (Std_Ulogic, Std_Ulogic)  of Std_Ulogic;
Constant Resolution_Table: StdLogic_Table := (
---  ---------------------------------------------------------------------------------------------
---|  U     X     0     1     Z     W     L     H     -          |      |
---  ---------------------------------------------------------------------------------------------
   ( 'U',   'U',   'U',   'U',   'U',   'U',   'U',   'U',   'U') ,   --   | U   |
   ( 'U',   'X',   'X',   'X',   'X',   'X',   'X',   'X',   'X') ,   --   | X   |
   ( 'U',   'X',   '0',   'X',   '0',   '0',   '0',   '0',   'X') ,   --   | 0   |
   ( 'U',   'X',   'X',   '1',   '1',   '1',   '1',   '1',   'X') ,   --   | 1   |
   ( 'U',   'X',   '0',   '1',   'Z',  'W',  'L',   'H',   'X') ,   --   | Z   |
   ( 'U',   'X',   '0',   '1',   'W',  'W',  'W',   'W',   'X') ,   --   | W   |
   ( 'U',   'X',   '0',   '1',   'L',  'W',  'L',   'W',   'X') ,   --   | L   |
   ( 'U',   'X',   '0',   '1',   'H',  'W',  'W',   'H',   'X') ,   --   | H   |
   ( 'U',   'X',   'X',   'X',   'X',   'X',   'X',   'X',   'X')) ;   --   | -   |
```


The Resolution Function

```
Function Resolved ( S: Std_Ulogic_Vector) Return Std_Ulogic IS
   Variable Result: Std_Ulogic := 'Z';
Begin
   If (S'Length = 1) Then Return S(S'Low);
   Else
        For I IN S'Range Loop
                Result := Resolution_Table(Result, S(I) );
        End Loop;
   End If;
   Return Result;
End Resolved;
```

Some Conversion Functions …

```
Function To_BitVector (S: Std_Logic_Vector; Xmap: Bit := '0')
    Return Bit_Vector IS
    Alias SV:  Std_Logic_Vector (S'Length-1 Downto 0) IS S;
    Variable Result:  Bit_Vector (S'Length-1 Downto 0);
Begin
    For I IN Result'Range Loop
        Case SV(I) IS
            When '0' | 'L' => Result(I) := '0';
            When '1' | 'H' => Result(I) := '1';
            When Others => Result(I) := xmap;
        End Case;
    End Loop;
    Return Result;
End;
```


…Some Conversion Functions

```
Type Logic_X01_Table IS Array (Std_Ulogic'Low To
Std_Ulogic'High) of X01;
Constant Cvt_to_X01 : Logic_X01_Table :=(
--   'U',   'X',   '0',   '1',   'Z',   'W',   'L',   'H',   '-'
     'X',   'X',   '0',   '1',   'X',   'X',   '0',   '1',   'X');
Function To_X01 (S: Std_Logic_Vector) Return Std_Logic_Vector
IS
    Alias SV:  Std_Logic_Vector (1 To S'Length) IS S;
    Variable Result: Std_Logic_Vector (1 To S'Length) ;
Begin
     For I IN Result'Range Loop
          Result(I) := Cvt_to_X01 (SV(I) );
     End Loop;
     Return Result;
End;
```

Edge Detectors

```
Function Rising_Edge (Signal S: Std_Ulogic) Return Boolean IS
Begin
   Return (   S'Event AND (To_X01(S) = '1') AND
              (To_X01(S'Last_Value)  = '0')
          );
End;

Function Falling_Edge (Signal S: Std_Ulogic) Return Boolean IS
Begin
   Return (   S'Event AND (To_X01(S) = '0') AND
              (To_X01(S'Last_Value)  = '1')
          );
End;
```


Unknown Detection

```
Function Is_X (Signal S: Std_Logic_Vector) Return Boolean IS
Begin
   For I IN S'Range Loop
      Case S(I) IS
            When 'U' | 'X' | 'Z' | 'W' | '-'  => Return True;
            When Others => NULL;
         End Case;
   End Loop;
   Return False;
End;
```

End std_logic_1164;

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);
x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12 : OUT std_logic_vector (7 downto 0));
end example;
architecture try of example is
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;
```


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;
```

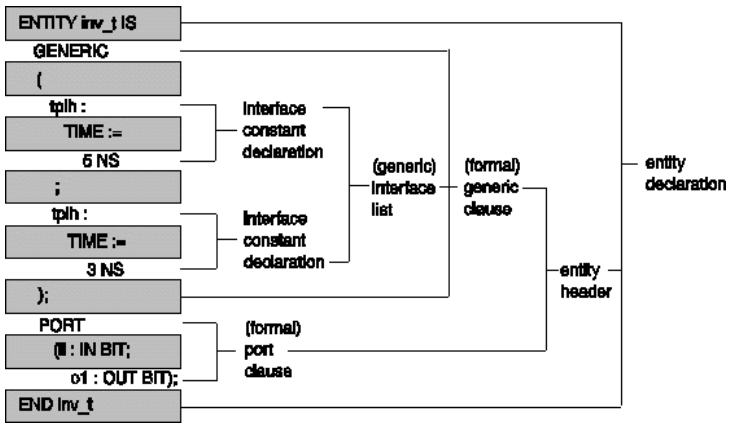
… Design Parameterization …

```
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;
```


Syntax of Generic Statement

- **GENERIC clause has syntax similar to PORT clause**
- **Interface CONSTANT declaration**



Values Passed to Generic Parameters

- **Several alternatives for passing values to GENERICs**
  - Using defaults
  - Assigning fixed values
  - Passing values from higher level components
  - Configuring designs

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

… 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*

Assigning Fixed Values to Generic Parameters …

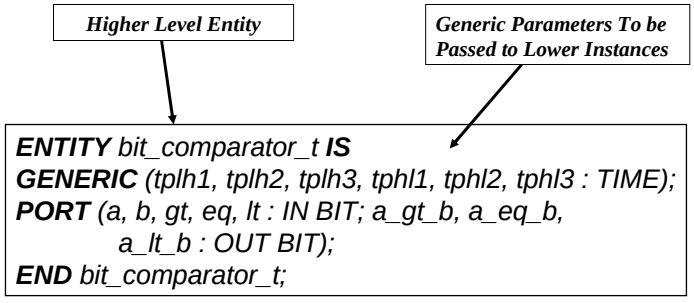
```
ARCHITECTURE fixed_delay OF bit_comparator IS
Component n1
Generic (tplh, tphl : Time); Port (i1: in Bit; o1: out Bit);
END Component;
Component n2
Generic (tplh, tphl : Time); Port (i1, i2: in Bit; o1: out Bit);
END Component;
Component n3
Generic (tplh, tphl : Time); Port (i1, i2, i3: in Bit; o1: 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);
```

… 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*

Passing Values From Higher Level Specs …



- *To pass values, upper level units must contain **GENERIC**s*
- *A timed bit_comparator is developed*

… Passing Values From Higher Level Specs …

```
ARCHITECTURE passed_delay OF bit_comparator_t 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 (tplh1, tphl1) Port Map (a, im1);
g1 : n1 Generic Map (tplh1, tphl1) Port Map (b, im2);
g2 : n2 Generic Map (tplh2, tphl2) Port Map (a, im2, im3);
```


Passing Values From Higher Level Specs …

```
g3 : n2 Generic Map (tplh2, tphl2) Port Map P (a, gt, im4);
g4 : n2 Generic Map (tplh2, tphl2) Port Map (im2, gt, im5);
g5 : n3 Generic Map (tplh3, tphl3) Port Map (im3, im4, im5, a_gt_b);
-- a_eq_b output
g6 : n3 Generic Map (tplh3, tphl3) Port Map (im1, im2, eq, im6);
g7 : n3 Generic Map (tplh3, tphl3) PORT MAP (a, b, eq, im7);
g8 : n2 Generic Map (tplh2, tphl2) PORT MAP (im6, im7, a_eq_b);
-- a_lt_b output
g9 : n2 Generic Map (tplh2, tphl2) Port Map (im1, b, im8);
g10 : n2 Generic Map (tplh2, tphl2) PORT MAP (im1, lt, im9);
g11 : n2 Generic Map (tplh2, tphl2) PORT MAP (b, lt, im10);
g12 : n3 Generic Map (tplh3, tphl3) PORT MAP (im8, im9, im10, a_lt_b);
END passed_delay;
```

- ***Component*** *declarations include* ***GENERIC****s*
- ***Component*** *instantiations include passed values*
- ***GENERIC*** *maps are required*

Passing Values Through Component Defaults …

```
ARCHITECTURE iterative OF nibble_comparator IS
Component comp1
Generic (tplh1 : Time := 2 ns; tplh2 : Time:= 3 ns; tplh3 : Time:= 4 ns;
   tphl1 : Time:= 4 ns; tphl2 : Time:= 5 ns; tphl3 : Time:= 6 ns);
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_t
   (passed_delay);
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));
c1to2: FOR i IN 1 TO 2 GENERATE
```


… Passing Values Through Component Defaults

```
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;
```

- *Using bit_comparator_t in a larger design*
- *Specification of **GENERIC** parameters is **Required***
- *Exclusion of **GENERIC** Map leaves all parameters **OPEN***
- *Association with **OPEN** causes default values to be used*

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***

Configuration Declarations

```
USE WORK.basic_utilities.ALL; -- Grants Entities Visibility
ARCHITECTURE customizable 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;
--
SIGNAL a, b : BIT_VECTOR (3 DOWNTO 0);
SIGNAL eql, lss, gtr : BIT;
SIGNAL vdd : BIT := '1';
SIGNAL gnd : BIT := '0';
--
BEGIN
a1: comp4 PORT MAP (a, b, gnd, vdd, gnd, gtr, eql, lss);
apply_data (a, 00&15&15&14&14&14&14&10&00&15&00&00&15, 500 NS);
apply_data (b, 00&14&14&15&15&12&12&12&15&15&15&00&00, 500 NS);
END customizable;
```

- *A general purpose testbench (comp4 component is unbound)*
- *Configure it to test any of the previous nibble_comparators*

Customizable Testbench Configuration Declaration

```
Configuration functional OF
nibble_comparator_test_bench IS
FOR customizable
  FOR a1 : comp4
    USE ENTITY WORK.nibble_comparator
(structural);
  END FOR;
END FOR;
END functional;
```

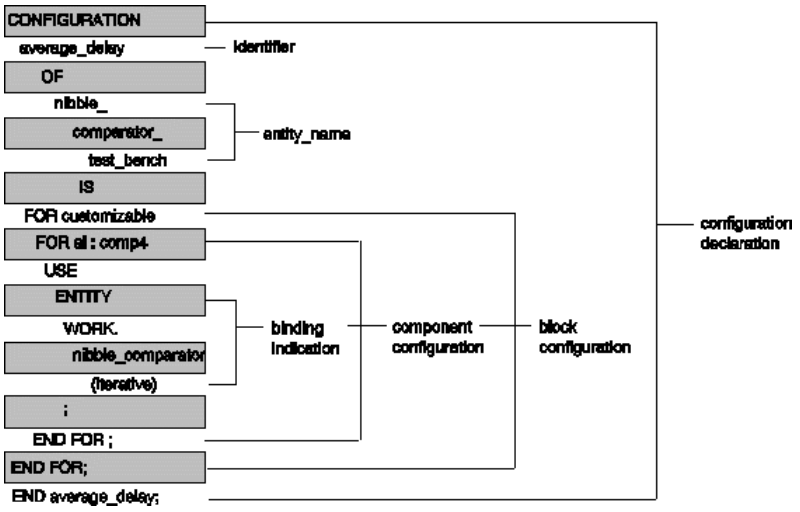
Configuration of testbench requires binding of a1:comp4

*Testbench configured for Structural **ARCHITECTURE** of nibble_comparator*

```
Configuration average_delay OF
nibble_comparator_test_bench IS
FOR customizable
  FOR a1 : comp4
    USE ENTITY WORK.nibble_comparator
(iterative );
  END FOR;
END FOR;
END functional;
```

*Testbench configured for iterattive **ARCHITECTURE** of nibble_comparator*

Syntax of Configuration Declaration



Configuration Nesting …

```
ARCHITECTURE flexible 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;
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));
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 flexible;
```

For flexible, No particular bit_comparator is specified
Using customizable testbench creates flexibility for:
1) Specifying nibble_comparator for testbench
2) Specifying bit_comparator for nibble_comparator

… Configuration Nesting

```
USE WORK.ALL;  -- Grant Entities Visibility
Configuration default_bit_level Of nibble_comparator_test_bench IS
FOR customizable
  FOR a1 : comp4
    USE ENTITY WORK.nibble_comparator(flexible);
    FOR flexible
      FOR c0, c3: comp1
        USE ENTITY WORK.bit_comparator (default_delay);
      END FOR;
      FOR c1to2  -- Grants Generate Block Visibility
        FOR c: comp1
          USE ENTITY WORK.bit_comparator (default_delay);
        END FOR;
      END FOR;
    END FOR;
  END FOR;
END FOR;
END default_bit_level;
```

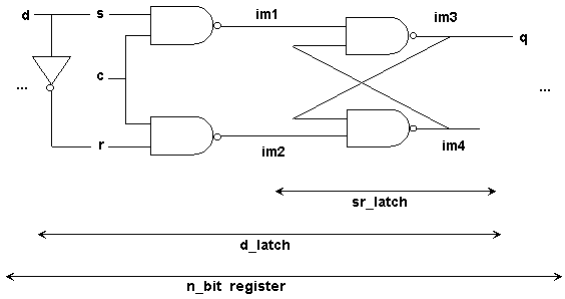
Configuring customizable testbench for testing default_delay bit_comparator instantiated through flexible nibble_comparator
Visibility is obtained before component configuration is done

Configurations and Generics

```
USE WORK.ALL;  -- Grant Entities Visibility
Configuration passed_bit_level Of nibble_comparator_test_bench IS
FOR customizable
 FOR a1 : comp4
  USE ENTITY WORK.nibble_comparator(flexible);
  FOR flexible
   FOR c0, c3: comp1
    USE ENTITY work.bit_comparator_t (Passed_delay)
    Generic Map (tplh1 => 2 NS, tplh2 => 3 NS, tplh3 => 4 NS, tphl1 => 4 NS, tphl2 => 5 NS, tphl3
   => 6 NS);
   END FOR;
   FOR c1to2  -- Grants Generate Block Visibility
    FOR c: comp1
     USE ENTITY work.bit_comparator_t (Passed_delay)
     Generic Map (tplh1 => 2 NS, tplh2 => 3 NS, tplh3 => 4 NS, tphl1 => 4 NS, tphl2 => 5 NS, tphl3
   => 6 NS);
    END FOR;
   END FOR;
  END FOR;
 END FOR;
END FOR;
END passed _bit_level;
```

*bit_comparator_t uses **GENERIC** parameters, Configure for:*
Component binding, and
***GENERIC** Map specification*

N-bit Register Example …



•NAND gates form sr_latch
•Add an inverter to form d_latch
•n d_latch structures form an n-bit register

… N-bit Register Example …

```
ENTITY sr_latch IS
PORT (s, r, c : IN BIT; q : OUT BIT);
END sr_latch;
ARCHITECTURE gate_level OF
sr_latch IS
Component n2
Port (i1, i2: IN Bit; o1: OUT Bit);
End Component;
SIGNAL im1, im2, im3, im4 : BIT;
BEGIN
g1 : n2 Port Map (s, c, im1);
g2 : n2 Port Map (r, c, im2);
g3 : n2 Port Map (im1, im4, im3);
g4 : n2 Port Map (im3, im2, im4);
q <= im3;
END gate_level;
```

```
ENTITY d_latch IS
PORT (d, c : IN BIT; q : OUT BIT);
END d_latch;
ARCHITECTURE sr_based OF
d_latch IS
Component sr
Port (s, r, c : in Bit; q : out Bit);
 End Component;
Component n1
Port (i1: in Bit; o1: out Bit);
 End Component;
SIGNAL dbar : BIT;
BEGIN
c1 : sr Port Map (d, dbar, c, q);
c2 : n1 Port Map (d, dbar);
END sr_based;
```


… N-bit Register Example …

```
ENTITY d_register IS
PORT (d : in Bit_Vector; c : in Bit; q : out Bit _Vector );
END d_register;
--
ARCHITECTURE latch_based OF d_register IS
Component dl PORT (d, c : in Bit; q : out Bit);
End Component;
BEGIN
dn : FOR i IN d'RANGE GENERATE
di : dl PORT MAP (d(i), c, q(i));
END GENERATE;
END latch_based;
```

- *gate_level sr_latch uses unbound gate components*
- *sr_based d_latch uses unbound sr_latch and gate*
- *latch_based d_register uses unbound d_latch components*
- *Different delay values remedy the oscillation problem*

Configuration of the N-bit Register

```
USE WORK.ALL;
CONFIGURATION average_gate_delay OF d_register IS
FOR latch_based –Architecture name
 FOR dn – Generate Visibility
  FOR di : dl --
   USE ENTITY WORK.d_latch(sr_based);
   FOR sr_based --
    FOR c1 : sr --
     USE ENTITY WORK.sr_latch(gate_level);
     FOR gate_level --
      FOR g2, g4 : n2 USE ENTITY WORK.nand2_t (average_delay)
       GENERIC MAP (5 NS, 6 NS);
      END FOR;
      FOR g1, g3 : n2 USE ENTITY WORK.nand2_t (average_delay)
       GENERIC MAP (2 NS, 4 NS);
      END FOR;
     END FOR;
    END FOR;
    FOR c2 : n1
     USE ENTITY WORK.inv_t(average_delay)
     GENERIC MAP (3 NS, 5 NS);
    END FOR;
   END FOR;
  END FOR;
 END FOR; END FOR;
END average_gate_delay;
```

To avoid timing problem: g2,g4 delays are at 5NS, 6NS while g1,g3 delays are at 2NS, 4NS

Alternative Configuration of the n-bit Register

```
USE WORK.ALL;
CONFIGURATION single_gate_delay OF d_register IS
 FOR latch_based –Architecture name
  FOR dn – Generate Visibility
   FOR di : dl --
    USE ENTITY WORK.d_latch(sr_based);
    FOR sr_based --
     FOR c1 : sr --
      USE ENTITY WORK.sr_latch(gate_level);
      FOR gate_level –
       FOR g2, g4 : n2 USE Entity Work.nand3 (single_delay)
          Port Map (i1, i1, i2, o1);
       End For;
       For g1, g3 : n2 USE Entity Work.nand2 (single_delay);
       End For;
      END FOR;
     END FOR;
     FOR c2 : n1 USE ENTITY WORK.inv (single_delay);
     END FOR;
    END FOR;
   END FOR;
  END FOR;
 END FOR;
END single_gate_delay;
```

•For g2 and g4 use NAND3 in place of NAND2
•This also eliminates the timing problem
•PORT MAP *overwrites default in component declaration*

Test Bench of 8-bit Register

```
ARCHITECTURE single OF d_register_test_bench IS
Component reg Port (d : in Bit_Vector (7 Downto 0); C : in Bit; q : Out
    Bit_Vector (7 Downto 0) );
End Component;
For r8 : reg Use Configuration Work.single_gate_delay;
SIGNAL data, outdata : BIT_VECTOR (7 Downto 0);
SIGNAL clk : BIT;
BEGIN
r8: reg   Port Map (data, clk, outdata);
data <= X"00", X"AA" After 0500 NS, X"55" After 1500 NS;
clk <=  '0', '1' After 0200 NS, '0' After 0300 NS,
'1' AFTER 0700 NS, '0' AFTER 0800 NS,
'1' AFTER 1700 NS, '0' AFTER 1800 NS;
END single;
```

- *Instantiating n-bit register in a testbench*
- *Associating d with an 8 bit vector makes an 8-bit register*

Selective Configuration of Generate-Statement Instances

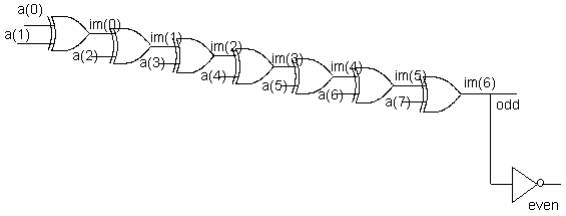
- **Example: Parity checker**

```
ENTITY xor2_t IS
GENERIC (tplh : TIME := 9 NS; tphl :
TIME := 7 NS);
PORT (i1, i2 : IN BIT; o1 : OUT BIT);
END xor2_t;
--
ARCHITECTURE average_delay OF
xor2_t IS
BEGIN
o1 <= i1 XOR i2 AFTER (tplh + tphl) /
2;
END average_delay;
```

```
ENTITY inv_t IS
GENERIC (tplh : TIME := 5 NS; tphl :
TIME := 3 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;
```

- *Develop an eight bit parity checker*
- *Use an XOR gate and an inverter*

8-bit Parity Checker …



ENTITY parity *IS*
Port *(A : In Bit_Vector (7 Downto 0); Odd, Even : Out Bit);*
END parity;

… 8-bit Parity Checker …

```
ARCHITECTURE iterative OF parity IS
Component X2
Port (I1, I2: In Bit; O1: Out Bit);
End Component;
Component N1
Port (I1: In Bit; O1: Out Bit);
End Component;
SIGNAL im : BIT_VECTOR ( 0 TO 6 );
BEGIN
first: x2 PORT MAP (a(0), a(1), im(0));
middle: FOR i IN 1 TO 6 GENERATE
m: x2 PORT MAP (im(i-1), a(i+1), im(i));
    END GENERATE;
last: odd <= im(6);
inv: n1 PORT MAP (im(6), even);
END iterative;
```

8-bit Parity Checker …

```
CONFIGURATION parity_binding OF parity IS
FOR iterative
  FOR first : x2
   USE ENTITY WORK.xor2_t (average_delay)
   GENERIC MAP (5 NS, 5 NS);
  END FOR;
  FOR middle(1 TO 5)
   FOR m : x2
    USE ENTITY WORK.xor2_t (average_delay)
    GENERIC MAP (5 NS, 5 NS);
   END FOR;
  END FOR;
  FOR middle ( 6)
   FOR m : x2
     USE ENTITY WORK.xor2_t (average_delay)
     GENERIC MAP (6 NS, 7 NS);
   END FOR;
  END FOR;
  FOR inv : n1
    USE ENTITY WORK.inv_t (average_delay) GENERIC MAP (5 NS, 5 NS);
  END FOR;
END FOR;
END parity_binding;
```

- *Due to fanout, last gate has a higher delay*
- *Element 6 of **Generate** statement specifies 6 NS, 7 NS delays*
- *Other generated elements use 5 NS, 5 NS*

***Generate** Index Value May be Used to Bind different instances of the **Generate** Statement to Different Entities/Architectures*

- *Can use **OTHERS** for indexing all other instantiations*