System Design w/ VHDL

Generics and Configurations
pp. 104-107, 153-156, 261-264, 292-307

Generics--Motivation

- Oftentimes we want to be able to specify a property separately for each instance of a component
  - Delay
  - Bit width
- VHDL allows models to be parameterized with generics

A Generic NAND Gate

entity NAND_GATE is
  generic (N: Natural := 2;
            D: Time := 10 ns);
  port (A: in Bit_Vector (1 to N);
        Z: out Bit);
end NAND_GATE;

Algorithmic architecture for generic NAND gate

architecture NAND_N_D of NAND_GATE is begin
  NAND_CAL: process (A)
    variable RESULT: Bit;
    begin
      RESULT := '1';
      for K in 1 to N loop
        RESULT := RESULT and A(K);
        exit when RESULT = '0';
      end loop;
      Z <= not RESULT after D;
    end process;
  end NAND_N_D;

Specification of Generic Values (1/3)

entity TESTBENCH is
  end TESTBENCH;
architecture NAND_N_D of TESTBENCH is
  signal A, B, C, D, E, T1, T2, Z_OUT: Bit;
component MY_NAND_GATE
  generic (N: Natural := 2;
            D: Time := 10 ns);
  port (A: in Bit_Vector (1 to N);
        Z: out Bit);
end component;
for all: MY_NAND_GATE use entity work.NAND_GATE(NAND_N_D);

Specification of Generic Values (2/3)

Architecture NAND_N_D ...
-- architecture declarations
begin
A <= '0', '1' after 50 ns, '0' after 100 ns;
B <= '0', '1' after 50 ns;
C <= '0', '1' after 100 ns, '0' after 150 ns;
D <= '0', '1' after 100 ns;
E <= '0', '1' after 100 ns;

Specification of Generic Values (3/3)

component MY_NAND_GATE
generic (N: Natural := 2; D: Time := 10 ns);
port (A: in Bit_Vector (1 to N); Z: out Bit);
end component;

G1: MY_NAND_GATE
port map (A(1) => A, A(2) => B, Z => T1);
G2: MY_NAND_GATE
generic map (3, 15 ns)
port map (A(1) => C, A(2) => D, A(3) => E, Z => T2);
G3: MY_NAND_GATE
generic map (D => 20 ns, N => 2)
port map (A(1) => T1, A(2) => T2, Z => Z_OUT);

Notes on generic

- Generic information is static—it can’t be changed during the simulation
  - I.e. You can’t have the simulation calculate the value that is going to be passed to the generic...
  - Specified at compile time.
- Generic value is instance-specific
  - Different instances of the same component can have different values.

More generic examples

An AND gate with parameterized rise/fall times:

ENTITY and2 IS
GENERIC(rise, fall : TIME; load : INTEGER);
PORT( a, b : IN BIT;
     c : OUT BIT);
END AND2;
ARCHITECTURE load_dependent OF and2 IS
SIGNAL internal : BIT;
BEGIN
  internal <= a AND b;
  c <= internal AFTER (rise + (load * 2 ns)) WHEN internal = '1'
  ELSE internal AFTER (fall + (load * 3 ns));
END load_dependent;

More generic examples

Using the parameterized AND gate:

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY test IS
GENERIC(rise, fall : TIME; load : INTEGER);
PORT ( ina, inb, inc, ind : IN std_logic;
       out1, out2 : OUT std_logic);
END test;
ARCHITECTURE test_arch OF test IS
COMPONENT and2
GENERIC(rise, fall : TIME := 10 NS;
load : INTEGER := 0);
PORT ( a, b, c : IN std_logic;
      out1, out2 : OUT std_logic);
END COMPONENT;
BEGIN
U1: and2 GENERIC MAP(10 ns, 12 ns, 3) PORT MAP (ina, inb, in1);
U2: and2 PORT MAP (inc, ind, out2);
END test_arch;

Specifying all these generics...

- Keeping track of generic values and which architectures to use for specific instances can be difficult, especially for large projects.
  - Don’t want to be required to edit each architecture file whenever a different component or value is desired.
- VHDL provides a method to declare configurations that specify generics and component architectures for each instance.
  - User can then tell the simulator which configuration to simulate—without having to edit the architecture or create a new one...
Analogy

Printed Circuit Board

( unpopulated with chips )

BOX OF CHIPS

Analogy (cont’d)

Printed Circuit Board

( unpopulated with chips )

BOX OF CHIPS

Configurations

- Configurations
  - specify which architectures to use for a particular component
  - specify which parameter values to use for a particular component
- Two basic forms
  - configuration specifications
  - configuration declarations

VHDL Binding

- Definition:
  - Associating an architectural description with a component in a structural model.
- Configurations bind all component declarations

Choosing a System

Configure the system so that ENTITY DA uses ARCHITECTURE DA_2, which instantiates component DB, which is bound to ARCHITECTURE DB_1

Configuration Specification

- A VHDL construct which helps associate a particular architecture with an instantiated component
- Simple configuration:
  - component MYCOMP port ( ...);
  - end component;
  - for U1 : MYCOMP use entity work.MYCOMP(BEHAV);
More Configuration

- Generic maps and port maps may be included in the configuration:
  for H1 : HALF_ADDER use entity work.HA(BEHAV);
  for H2 : HALF_ADDER use entity
  EDSLIB.HALFADD(STRUCT)
  generic map (GATEDELAY => 5 ns)
  port map (I1=>A, I2=>B, S1=>SUM, C=>CARRY);

Default Mapping Rules

- If the entity name is the same as the component name, then this entity is bound to the component
- If there are multiple architectures for the same entity (for example, DA_1, DA_2,...) the last compiled architecture for the entity is chosen
- Very dangerous to use defaults

"Boiler Plate" Configuration

- Suppose you have a large system model, and you want to experiment with the configuration of a few sub-components
- Procedure:
  - Develop system model
  - Create library for model
  - Do not bind sub-components in model
  - Use separate Configuration Declaration

Example 1/3

- Example CPU:
  entity CPU
  is
  port (....port declaration.... );
  end CPU;

  architecture RISC of CPU is
  component ALU
  port (....port declaration.... );
  end component;
  begin
  U1: ALU port map (....port declaration....);
  end RISC;

Example Configuration Declaration

configuration MYCONFIG of CPU is
  for RISC
  for U1 : ALU
  use entity work.ALU(FAST);
  end for;
  end for;
  end MYCONFIG;
**Configuration Declaration Syntax**

- Configuration MYCONFIG of CPU is
- for RISC
- for U1 : ALU
- use entity work.ALU(FAST);
- end for;
- end for;
- end MYCONFIG;

**Common Errors**

- Modifying the model of a component and forgetting to reanalyze the component prior to reuse

**Common Errors**

- Generics can have their values defined in multiple places:
  - within a model
  - in a component instantiation (generic map)
  - within an architecture in a component declaration
  - within a configuration declaration
- Changing the value in one place may not have the intended effect due to precedence

**Common Errors**

- When using default bindings of components, the name, type and mode of each signal in the component declaration must exactly match that of the entity
- This is dangerous.

**Common Errors**

- Inheriting a generic value by way of default initializations in the component declaration may lead to unexpected values.
- This is dangerous.

**Summary**

- Generics
  - Parameterize a model
- Configurations
  - Specify architectures and parameters
- Next time: Start synthesis
- Reading: pp. 381-392, 439-452.