Required reading

- P. Chu, RTL Hardware Design using VHDL

Chapters
14.5 For Generate Statement
14.6 Conditional Generate Statement
15.2 Data Types for Two-Dimensional Signals
15.3 Commonly Used Intermediate-Sized RT-Level Components

Dataflow VHDL

Major instructions

Concurrent statements
- concurrent signal assignment (\(\leftarrow\))
- conditional concurrent signal assignment (when-else)
- selected concurrent signal assignment (with-select-when)
- generate scheme for equations (for-generate)

PARITY Example

PARITY: Block Diagram
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY parity IS
PORT(
    parity_in : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    parity_out : OUT STD_LOGIC
);
END parity;

ARCHITECTURE parity_dataflow OF parity IS
SIGNAL xor_out: std_logic_vector (6 downto 1);
BEGIN
    xor_out(1) <= parity_in(0) XOR parity_in(1);
    xor_out(2) <= xor_out(1) XOR parity_in(2);
    xor_out(3) <= xor_out(2) XOR parity_in(3);
    xor_out(4) <= xor_out(3) XOR parity_in(4);
    xor_out(5) <= xor_out(4) XOR parity_in(5);
    xor_out(6) <= xor_out(5) XOR parity_in(6);
    xor_out(7) <= xor_out(6) XOR parity_in(7);
    parity_out <= xor_out(7) XOR parity_in(7);
END parity_dataflow;

ARCHITECTURE parity_dataflow (2) OF parity IS
SIGNAL xor_out: STD_LOGIC_VECTOR (6 DOWNTO 1);
BEGIN
    G2: FOR i IN 1 TO 7 GENERATE
        left_xor: IF i=1 GENERATE
            xor_out(i) <= parity_in(i-1) XOR parity_in(i);
        END GENERATE;
        middle_xor: IF (i >1) AND (i<7) GENERATE
            xor_out(i) <= xor_out(i-1) XOR parity_in(i);
        END GENERATE;
        right_xor: IF i=7 GENERATE
            parity_out <= xor_out(i-1) XOR parity_in(i);
        END GENERATE;
    END GENERATE;
END parity_dataflow;
PARITY: Architecture (2)

ARCHITECTURE parity_dataflow OF parity IS
SIGNAL xor_out: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
    xor_out(0) <= parity_in(0);
    G2: FOR i IN 1 TO 7 GENERATE
        xor_out(i) <= xor_out(i-1) XOR parity_in(i);
    END GENERATE G2;
    parity_out <= xor_out(7);
END parity_dataflow;

For Generate Statement

For - Generate

label:
FOR identifier IN range GENERATE
    (Concurrent Statements)
END GENERATE;

Conditional Generate Statement

If - Generate

label:
IF boolean_expression GENERATE
    (Concurrent Statements)
END GENERATE;

Generate scheme for components

Structural VHDL

Major instructions

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

Example 1
Example 1

A 4-to-1 Multiplexer

Straightforward code for Example 1

Modified code for Example 1

Example 2
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity up_or_down_counter is
generic(
  WIDTH: natural:=4;
  UP: natural:=0
);
port(
  clk, reset: in std_logic;
  q: out std_logic_vector(WIDTH-1 downto 0)
);
end up_or_down_counter;

architecture arch of up_or_down_counter is
signal r_reg: unsigned(WIDTH-1 downto 0);
signal r_next: unsigned(WIDTH-1 downto 0);
begin
  -- register
  process(clk, reset)
  begin
    if (reset='1') then
      r_reg <= (others=>'0');
    elsif (clk'event and clk='1') then
      r_reg <= r_next;
    end if;
  end process;

  -- next-state logic
  inc_gen: -- incrementor
    if UP=1 generate
      r_next <= r_reg + 1;
    end generate;
  dec_gen: -- decrementor
    if UP/=1 generate
      r_next <= r_reg – 1;
    end generate;

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

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity up_and_down_counter is
  generic(WIDTH: natural:=4);
port(
  clk, reset: in std_logic;
  mode: in std_logic;
  q: out std_logic_vector(WIDTH-1 downto 0)
);
end up_and_down_counter;

Example 4
Up-and-down Free Running Counter

Up-and-down Free Running Counter (1)
Up-and-down Free Running Counter (2)

architecture arch of up_and_down_counter is
begin
  -- register
  process(clk,reset)
  begin
    if (reset='1') then
      r_reg <= (others=>'0');
    elsif (clk'event and clk='1') then
      r_reg <= r_next;
    end if;
  end process;
end arch;

Up-and-down Free Running Counter (3)

-- next-state logic
r_next <= r_reg + 1 when mode='1' else r_reg - 1;

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

Example 5
Variable Rotator

Block diagram

VHDL code for a 16-bit 2-to-1 Multiplexer

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux2to1_16 IS
  PORT ( w0 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         w1 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         s : IN STD_LOGIC;
         f : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
END mux2to1_16;

ARCHITECTURE dataflow OF mux2to1_16 IS
BEGIN
  f <= w0 WHEN s = '0' ELSE w1;
END dataflow;
Fixed rotation

Fixed rotation by L positions

VHDL code for a fixed 16-bit rotator

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fixed_rotator_left_16 IS
    GENERIC (L : INTEGER := 1);
    PORT (a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          y : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0)) ;
END fixed_rotator_left_16;

ARCHITECTURE dataflow OF fixed_rotator_left_16 IS
BEGIN
    y <= a(15-L downto 0) & a(15 downto 15-L+1);
END dataflow;

Structural VHDL code for a variable 16-bit rotator (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY variable_rotator_16 is
    PORT(A : IN STD_LOGIC_VECTOR(15 downto 0);
         B : IN STD_LOGIC_VECTOR(3 downto 0);
         C : OUT STD_LOGIC_VECTOR(15 downto 0)) ;
END variable_rotator_16;

ARCHITECTURE structural OF variable_rotator_16 IS
COMPONENT mux2to1_16
    PORT (w0 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          w1 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          s : IN  STD_LOGIC;
          f : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)) ;
END COMPONENT;

COMPONENT fixed_rotator_left_16
    GENERIC (L : INTEGER := 1);
    PORT (a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          y : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0)) ;
END COMPONENT;

TYPE array1 IS ARRAY (0 to 4) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
TYPE array2 IS ARRAY (0 to 3) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL Al : array1;
SIGNAL Ar : array2;
BEGIN
    Al(0) <= A;
    G: FOR i IN 0 TO 3 GENERATE
        ROT_I: fixed_rotator_left_16
            GENERIC MAP (L => 2** i)
            PORT MAP ( a => Al(i),
                       y => Ar(i));
        MUX_I:  mux2to1_16
            PORT MAP (w0 => Al(i),
                       w1 => Ar(i),
                       s  => B(i),
                       f  => Al(i+1));
    END GENERATE;
    C <= Al(4);
END variable_rotator_16;

Structural VHDL code for a variable 16-bit rotator (2)

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ARCHITECTURE structural OF variable_rotator_16 IS
COMPONENT mux2to1_16
    PORT (w0 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          w1 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          s  : IN  STD_LOGIC;
          f  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)) ;
END COMPONENT;

COMPONENT fixed_rotator_left_16
    GENERIC (L : INTEGER := 1);
    PORT (a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          y : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0)) ;
END COMPONENT;

END variable_rotator_16;

Structural VHDL code for a variable 16-bit rotator (3)

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ARCHITECTURE structural OF variable_rotator_16 IS
COMPONENT mux2to1_16
    PORT (w0 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          w1 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          s  : IN  STD_LOGIC;
          f  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)) ;
END COMPONENT;

COMPONENT fixed_rotator_left_16
    GENERIC (L : INTEGER := 1);
    PORT (a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          y : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0)) ;
END COMPONENT;

END variable_rotator_16;
Example 6

XOR Tree

Example 6

XOR Tree (1)

library ieee;
use ieee.std_logic_1164.all;
use work.util_pkg.all;

entity reduced_xor is
generic(WIDTH: natural:=8);
port(
  a: in std_logic_vector(WIDTH-1 downto 0);
  y: out std_logic
);
end reduced_xor;

architecture gen_tree_arch of reduced_xor is
constant STAGE: natural:=
  log2c(WIDTH);
signal p:
  std_logic_2d(STAGE downto 0, WIDTH-1 downto 0);

begin
  -- rename input signal
  in_gen:
    for i in 0 to (WIDTH-1) generate
    p(STAGE, i) <= a(i);
  end generate;
  -- replicated structure
  stage_gen:
    for s in (STAGE-1) downto 0 generate
    row_gen:
      for r in 0 to (2**s-1) generate
      p(s, r) <= p(s+1, 2*r) xor p(s+1, 2*r+1);
    end generate;
    end generate;
  -- rename output signal
  y <= p(0, 0);
end gen_tree_arch;

util_pkg (1)

library ieee;
use ieee.std_logic_1164.all;

package util_pkg is

  type std_logic_2d is
    array(integer range <= >, integer range <= >) of std_logic;

  function log2c (n: integer) return integer;

end util_pkg ;
util_pkg (2)

```vhdl
package body util_pkg is
  function log2c(n: integer) return integer is
    variable m, p: integer;
    begin
      m := 0;
      p := 1;
      while p < n loop
        m := m + 1;
        p := p * 2;
      end loop;
      return m;
    end log2c;
  end util_pkg;
```

Array Data Type

**Array Type**

```vhdl
TYPE data_type_name IS ARRAY (range_1, range2, ...)
  OF element_data_type;
```

XOR Tree with Arbitrary Input Size (1)

```vhdl
begin
  -- rename input signal
  in_gen: for i in 0 to (WIDTH-1) generate
    p(STAGE,i) <= a(i);
  end generate;
  -- padding 0's
  pad0_gen: if WIDTH < (2**STAGE) generate
    zero_gen: for i in WIDTH to (2**STAGE-1) generate
      p(STAGE,i) <= '0';
    end generate;
  end generate;
end gen_tree1_arch;
```

XOR Tree with Arbitrary Input Size (2)

```vhdl
-- replicated structure
stage_gen: for s in (STAGE-1) downto 0 generate
  row_gen: for r in 0 to (2**s-1) generate
    p(s,r) <= p(s+1,2*r) xor p(s+1,2*r+1);
  end generate;
end generate;
-- rename output signal
y <= p(0,0);
end gen_tree2_arch;
```

Predefined Unconstrained Array Types

**Predefined**

```vhdl
bit_vector array of bits
string array of characters
```
Predefined Unconstrained Array Types

Defined in the std_logic_1164 package:

type std_logic_vector is array (natural range <>)
of std_logic;

Defined in the numeric_std package:

type unsigned is array (natural range <>)
of bit;
type signed is array (natural range <>)
of bit;

Using Predefined Unconstrained Array Types

subtype byte is bit_vector(7 downto 0);

signal channel_busy : bit_vector(1 to 4);

constant ready_message : string := "ready";

signal memory_bus: std_logic_vector (31 downto 0);

User-defined Unconstrained Array Types

type std_logic_2d is
array(integer range <>, integer range <>) of std_logic;

signal s1: std_logic_2d(3 downto 0, 5 downto 0);
signal s2: std_logic_2d(15 downto 0, 31 downto 0);
signal s3: std_logic_2d(7 downto 0, 1 downto 0);

User-defined Unconstrained Array Type in a package

use work.util_pkg.all;

define ....

generic (ROW: natural;
COL: natural);

port { p1: in std_logic_2d(ROW-1 downto 0, COL-1 downto 0);

signal s1: std_logic_2d(ROW-1 downto 0, COL-1 downto 0);

Array-of-Arrays Data Type

constant ROW : natural := 4;
constant COL : natural := 6;
type sram_row_by_col is array (ROW-1 downto 0) of std_logic_vector(COL-1 downto 0);

signal t1: sram_row_by_col;
signal v1: std_logic_vector(COL-1 downto 0);
signal b1: std_logic;

t1 <= ("000101", "101001", others => '0');
b1 <= t1(3)(0);
v1 <= t1(2);
Array Attributes

- \( A'\text{left}(N) \) : left bound of index range of dimension \( N \) of \( A \)
- \( A'\text{right}(N) \) : right bound of index range of dimension \( N \) of \( A \)
- \( A'\text{low}(N) \) : lower bound of index range of dimension \( N \) of \( A \)
- \( A'\text{high}(N) \) : upper bound of index range of dimension \( N \) of \( A \)
- \( A'\text{range}(N) \) : index range of dimension \( N \) of \( A \)
- \( A'\text{reverse_range}(N) \) : reversed index range of dimension \( N \) of \( A \)
- \( A'\text{length}(N) \) : length of index range of dimension \( N \) of \( A \)
- \( A'\text{ascending}(N) \) : true if index range of dimension \( N \) of \( A \) is an ascending range, false otherwise

Array Attributes - Examples

- Type \( A \) is array (1 to 4, 31 downto 0);
- \( A'\text{left}(1) = 1 \)
- \( A'\text{right}(2) = 0 \)
- \( A'\text{low}(1) = 1 \)
- \( A'\text{high}(2) = 31 \)
- \( A'\text{range}(1) = 1 \) to 4
- \( A'\text{length}(2) = 32 \)
- \( A'\text{ascending}(2) = \text{false} \)

Unconstrained PARITY Generator (1)

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

ENTITY parity IS
    PORT(
        parity_in : IN STD_LOGIC_VECTOR;
        parity_out : OUT STD_LOGIC
    );
END parity;
```

ARCHITECTURE parity_dataflow OF parity IS
    CONSTANT  width: natural := parity_in'length;
    SIGNAL x_or_out: STD_LOGIC_VECTOR (width-1 DOWNTO 0);
    BEGIN
        x_or_out(0) <= parity_in(0);
        G2: FOR i IN 1 TO 7 GENERATE
            x_or_out(i) <= x_or_out(i-1) XOR parity_in(i);
        END GENERATE G2;
        parity_out <= x_or_out(width-1);
    END parity_dataflow;

Unconstrained PARITY Generator (2)

```vhdl
ARCHITECTURE parity_dataflow OF parity IS
    CONSTANT width: natural := parity_in'length;
    SIGNAL xor_out: STD_LOGIC_VECTOR (width-1 DOWNTO 0);
    BEGIN
        xor_out(0) <= parity_in(0);
        G2: FOR i IN 1 TO 7 GENERATE
            xor_out(i) <= xor_out(i-1) XOR parity_in(i);
        END GENERATE G2;
        parity_out <= xor_out(width-1);
    END parity_dataflow;
```

Will the previous code work for the following types of signal parity_in?

- \( \text{std\_logic\_vector}(7 \) downto 0); \)
- \( \text{std\_logic\_vector}(0 \) to 7); \)
- \( \text{std\_logic\_vector}(15 \) downto 8); \)
- \( \text{std\_logic\_vector}(8 \) to 15); \)

Unconstrained PARITY Generator (3)

```vhdl
ARCHITECTURE parity_dataflow OF parity IS
    CONSTANT width: natural := parity_in'length;
    SIGNAL xor_out: STD_LOGIC_VECTOR (width-1 DOWNTO 0);
    BEGIN
        pp <= parity_in;
        xor_out(0) <= pp(0);
        G2: FOR i IN 1 TO 7 GENERATE
            xor_out(i) <= xor_out(i-1) XOR pp(i);
        END GENERATE G2;
        parity_out <= xor_out(width-1);
    END parity_dataflow;
```

Unconstrained PARITY Generator (4)
Aliases
Syntax:

```
ALIAS name : type := expression;
```

Example:

```
signal IR : std_logic_vector(31 downto 0);
alias IR_opcode : std_logic_vector(5 downto 0) is IR(31 downto 26);
alias IR_reg1_addr : std_logic_vector(4 downto 0) is IR(25 downto 21);
alias IR_reg2_addr : std_logic_vector(4 downto 0) is IR(20 downto 16);
```

Constants
Syntax:

```
CONSTANT name : type := value;
```

Examples:

```
CONSTANT init_value : STD_LOGIC_VECTOR(3 downto 0) := "0100";
CONSTANT ANDA_EXT : STD_LOGIC_VECTOR(7 downto 0) := X"B4";
CONSTANT counter_width : INTEGER := 16;
CONSTANT buffer_address : INTEGER := 16#FFE4#;
CONSTANT clk_period : TIME := 20 ns;
CONSTANT strobe_period : TIME := 333.333 ms;
```

Constants - features
Constants can be declared in a
PACKAGE, ENTITY, ARCHITECTURE

When declared in a PACKAGE, the constant is truly global, for the package can be used in several entities.

When declared in an ARCHITECTURE, the constant is local, i.e., it is visible only within this architecture.

When declared in an ENTITY declaration, the constant can be used in all architectures associated with this entity.
Explicit Component Declaration versus Package

- Explicit component declaration is when you declare components in main code
- When have only a few component declarations, this is fine
- When have many component declarations, use packages for readability
- Packages also help with portability and sharing of libraries among many users in a company
- **Remember, the actual instantiations always take place in main code**
- Only the declarations can be in main code or package

Explicit Component Declaration Tips

- For simple projects put entity .vhd files all in same directory
- Declare components in main code
- If using Aldec, make sure compiler knows the correct hierarchy
  - From lowest to highest
- Xilinx will figure out hierarchy automatically

METHOD #2: Package component declaration

- Components declared in package
- Actual instantiations and port maps always in main code

Packages

- Instead of declaring all components can declare all components in a PACKAGE, and INCLUDE the package once
  - This makes the top-level entity code cleaner
  - It also allows that complete package to be used by another designer
- A package can contain
  - Components
  - Functions, Procedures
  - Types, Constants

**Package – example (1)**

```vhd
LIBRARY ieee;
USE ieee.std_logic_1164.all;
PACKAGE GatesPkg IS
  COMPONENT mux2to1
    PORT (w0, w1, s : IN STD_LOGIC;
           f : OUT STD_LOGIC);
  END COMPONENT;
  COMPONENT priority
    PORT (w : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
           y : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
           z : OUT STD_LOGIC);
  END COMPONENT;
END GatesPkg;
```

**Package – example (2)**

```vhd
COMPONENT dec2to4
  PORT (w : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        En : IN STD_LOGIC;
        y : OUT STD_LOGIC_VECTOR(0 TO 3))
  END COMPONENT;
COMPONENT regn
  GENERIC (N : INTEGER := 8);
  PORT (D     : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        Enable, Clock : IN STD_LOGIC;
        Q     : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0))
  END COMPONENT;
```

**Package – example (3)**

```vhd
PACKAGE regPkg
  COMPONENT reg
    GENERIC (N : INTEGER := 8);
    PORT (D     : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
           Enable, Clock : IN STD_LOGIC;
           Q     : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0))
  END COMPONENT;
```
constant ADDAB : std_logic_vector(3 downto 0) := "0000";
custom ADDAM : std_logic_vector(3 downto 0) := "0001";
custom SUBAB : std_logic_vector(3 downto 0) := "0010";
custom SUBAM : std_logic_vector(3 downto 0) := "0011";
custom NOTA : std_logic_vector(3 downto 0) := "0100";
custom NOTB : std_logic_vector(3 downto 0) := "0101";
custom NOTM : std_logic_vector(3 downto 0) := "0110";
custom ANDAB : std_logic_vector(3 downto 0) := "0111";
END GatesPkg;

PACKAGE example

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.GatesPkg.all;

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

ARCHITECTURE structural OF priority_resolver IS
  SIGNAL  p : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
  SIGNAL  q : STD_LOGIC_VECTOR (1  DOWNTO 0) ;
  SIGNAL  z : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
  SIGNAL  ena : STD_LOGIC ;
BEGIN
  u1: mux2to1 PORT MAP
     (w0 => r(0),
      w1 => r(1),
      s => s(0),
      p => p(0));
  p(1) <= r(2);
  p(2) <= r(3);
  u2: mux2to1 PORT MAP
     (w0 => r(4),
      w1 => r(5),
      s => s(1),
      p => p(3));
  u3: priority PORT MAP
    (w => p,
     y => q,
     z => ena);
  u4: dec2to4 PORT MAP
    (w => q,
     En => ena,
     y => z);
  u5: regn GENERIC MAP
     (N => 4)
    PORT MAP
     (D => z ,
      Enable => En ,
      Clock => Clk,
      Q => t );
END structural;

VHDL Design Styles

Mixing Design Styles
Inside of an Architecture

Aldec Compilation Order

• Include package before top-level
Mixed Style Modeling

architecture ARCHITECTURE_NAME of ENTITY_NAME is

• Here you can declare signals, constants, types, etc.
• Component declarations

begin
  Concurrent statements
  • Concurrent simple signal assignment
  • Conditional signal assignment
  • Selected signal assignment
  • Generate statement
  • Component instantiation statement
  • Process statement
    • inside process you can use only sequential statements
end ARCHITECTURE_NAME;

PRNG Example (1)

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

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

ARCHITECTURE mixed OF PRNG is

signal Ands : std_logic_vector(4 downto 0);
signal Sin : std_logic;
signal Coeff_Q : std_logic_vector(4 downto 0);
signal Shift5_Q : std_logic_vector(4 downto 0);

begin
  Concurrent Statements
  • Generate statement
    • Use only sequential statements
end ARCHITECTURE_NAME;

PRNG Example (2)

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

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

ARCHITECTURE mixed OF PRNG is

signal Ands : std_logic_vector(4 downto 0);
signal Sin : std_logic;
signal Coeff_Q : std_logic_vector(4 downto 0);
signal Shift5_Q : std_logic_vector(4 downto 0);

begin
  Concurrent Statements
  • Concurrent simple signal assignment
  • Conditional signal assignment
  • Selected signal assignment
  • Generate statement
  • Component instantiation statement
  • Process statement
    • inside process you can use only sequential statements
end ARCHITECTURE_NAME;