Examples of VHDL Descriptions

Advanced Electronic Design Automation

Examples of VHDL Descriptions

Author: Ian Elliott of Northumbria University

This file contains a selection of VHDL source files which serve to illustrate the diversity and power of the language when used to describe various types of hardware. The examples range from simple combinational logic, described in terms of basic logic gates, to more complex systems, such as a behavioural model of a microprocessor and associated memory. All of the examples can be simulated using any IEEE compliant VHDL simulator and many can be synthesised using current synthesis tools.

Use the hierarchical links below to navigate your way through the examples:

- Combinational Logic
- Counters
- Shift Registers
- Memory
- State Machines
- Registers
- Systems
- ADC and DAC
- Arithmetic

Combinational Logic

- Exclusive-OR Gate (Dataflow style)
- Exclusive-OR Gate (Behavioural style)
- Exclusive-OR Gate (Structural style)
- Miscellaneous Logic Gates
- Three-input Majority Voter
- Magnitude Comparator
- Quad 2-input Nand (74x00)
- BCD to Seven Segment Decoder
- Dual 2-to-4 Decoder
- Octal Bus Transceiver
- Quad 2-input OR
- 8-bit Identity Comparator
- Hamming Encoder
- Hamming Decoder
- Multiplexer 2-to-4 using Selected Signal Assignment Statement
- Multiplexer 2-to-4 with Testbench and Configuration
- Multiplexer 16-to-4 using Selected Signal Assignment Statement
- Multiplexer 16-to-4 using Conditional Signal Assignment Statement
- Multiplexer 16-to-4 using if-then-elsif-else Statement
- M68008 Address Decoder
- Highest Priority Encoder
- N-input AND Gate
Examples of VHDL Descriptions

- Counter using a Conversion Function
- Generated Binary Up Counter
- Counter using Multiple Wait Statements
- Synchronous Down Counter with Parallel Load
- Mod-16 Counter using JK Flip-flops
- Pseudo Random Bit Sequence Generator
- Universal Counter/Register
- n-Bit Synchronous Counter

Shift Registers

- Universal Shift Register/Counter
- TTL164 Shift Register
- Behavioural description of an 8-bit Shift Register
- Structural Description of an 8-bit Shift Register

Memory

- ROM-based Waveform Generator
- A First-in First-out Memory
- Behavioural model of a 16-word, 8-bit Random Access Memory
- Behavioural model of a 256-word, 8-bit Read Only Memory

State Machines

- Classic 2-Process State Machine and Test Bench
- State Machine using Variable
- State Machine with Asynchronous Reset
- Pattern Detector FSM with Test Bench
- State Machine with Moore and Mealy outputs
- Moore State Machine with Explicit State encoding
- Mealy State Machine with Registered Outputs
- Moore State Machine with Concurrent Output Logic

Systems

- Pelican Crossing Controller
- Simple Microprocessor System
- Booth Multiplier
- Lottery Number Generator
- Digital Delay Unit
- Chess Clock

ADC and DAC

- Package defining a Basic Analogue type
- 16-bit Analogue to Digital Converter
- 16-bit Digital to Analogue Converter
- 8-bit Analogue to Digital Converter
- 8-bit Unipolar Successive Approximation ADC
Examples of VHDL Descriptions

Arithmetic

- 8-bit Unsigned Multiplier
- n-bit Adder using the Generate Statement
- A Variety of Adder Styles
- Booth Multiplier

Registers

- Universal Register
- Octal D-Type Register with 3-State Outputs
- Quad D-Type Flip-flop
- 8-bit Register with Synchronous Load and Clear

Universal Register

Description - This design is a universal register which can be used as a straightforward storage register, a bi-directional shift register, an up counter and a down counter. The register can be loaded from a set of parallel data inputs and the mode is controlled by a 3-bit input. The 'termcnt' (terminal count) output goes high when the register contains zero.

LIBRARY ieee;
USE ieee.Std_logic_1164.ALL;
USE ieee.Std_logic_unsigned.ALL;
ENTITY unicntr IS
  GENERIC(n : Positive := 8); -- size of counter/shifter
  PORT(clock, serinl, serinr : IN Std_logic; -- serial inputs
       mode : IN Std_logic_vector(2 DOWNTO 0); -- mode control
       datain : IN Std_logic_vector((n-1) DOWNTO 0); -- parallel inputs
       dataout : OUT Std_logic_vector((n-1) DOWNTO 0); -- parallel outputs
       termcnt : OUT Std_logic); -- terminal count output
END unicntr;

ARCHITECTURE v1 OF unicntr IS
  SIGNAL int_reg : Std_logic_vector((n-1) DOWNTO 0);
BEGIN
  main_proc : PROCESS
  BEGIN
    WAIT UNTIL rising_edge(clock);
    CASE mode IS
    -- reset
    WHEN "000" => int_reg <= (OTHERS => '0');
    -- parallel load
    WHEN "001" => int_reg <= datain;
    -- count up
    WHEN "010" => int_reg <= int_reg + 1;
    -- count down
    WHEN "011" => int_reg <= int_reg - 1;
    -- shift left
    WHEN "100" => int_reg <= int_reg((n-2) DOWNTO 0) & serinl;
    -- shift right
    WHEN "101" => int_reg <= int_reg & serin & int_reg((n-1) DOWNTO 1);
    -- do nothing
    WHEN OTHERS => NULL;
    END CASE;
  END PROCESS;

  det_zero : PROCESS(int_reg) -- detects when count is 0
  BEGIN
    termcnt <= '1';
    FOR i IN int_reg'Range LOOP
      IF int_reg(i) = '0' THEN
        termcnt <= '1';
      END IF;
    END LOOP;
  END PROCESS;
END unicntr;
Examples of VHDL Descriptions

---

Octal D-Type Register with 3-State Outputs

Simple model of an Octal D-type register with three-state outputs using two concurrent statements.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ttl374 IS
PORT(clock, oebar : IN std_logic;
data : IN std_logic_vector(7 DOWNTO 0);
qout : OUT std_logic_vector(7 DOWNTO 0));
END ENTITY ttl374;
ARCHITECTURE using_1164 OF ttl374 IS
--internal flip-flop outputs
SIGNAL qint : std_logic_vector(7 DOWNTO 0);
BEGIN
qint <= data WHEN rising_edge(clock); --d-type flip flops
qout <= qint WHEN oebar = '0' ELSE "ZZZZZZZZ"; --three-state buffers
END ARCHITECTURE using_1164;
```

---

Exclusive-OR Gate (Dataflow style)

```
-- 2 input exclusive or
-- Modeled at the RTL level.
entity x_or is
port (
in1 : in  bit;
in2 : in  bit;
out1 : out bit);
end x_or;
architecture rtl of x_or is
begin
out1 <= in1 xor in2 after 10 ns;
end rtl;
```

---

Exclusive-OR Gate (Behavioural style)

```
-- Exclusive or gate
-- modeled at the behavioral level.
entity x_or is
port (
in1 : in  bit;
in2 : in  bit;
out1 : out bit);
end x_or;
architecture behavior of x_or is
```
Examples of VHDL Descriptions

begin
  process(in1, in2)
  begin
    if in1 = in2 then
      out1 <= '0' after 10 ns;
    else
      out1 <= '1' after 10 ns;
    end if;
  end process;
end behavior;

Exclusive-OR Gate (Structural style)

-- 2 input exclusive-or gate.
-- Modeled at the structural level.

entity x_or is
port ( in1 : in  bit  ;
       in2 : in  bit  ;
       out1 : out bit);
end x_or;
entity and_gate is
port ( a : in  bit  ;
       b : in  bit  ;
       c : out bit);
end and_gate;
architecture behavior of and_gate is
begin
  process(a,b)
  begin
    c <= a and b after 5 ns;
  end process;
end behavior;
entity or_gate is
port ( d : in  bit  ;
       e : in  bit  ;
       f : out bit);
end or_gate;
architecture behavior of or_gate is
begin
  process(d,e)
  begin
    f <= d or e after 4 ns;
  end process;
end behavior;
entity inverter is
port ( g : in  bit  ;
       h : out bit);
end inverter;
architecture behavior of inverter is
begin
  process(g)
  begin
    h <= not g after 3 ns;
  end process;
end behavior;
Examples of VHDL Descriptions

architecture structural of x_or is
-- signal declarations
signal t1, t2, t3, t4 : bit;
-- local component declarations
component and_gate
    port (a, b : in bit; c : out bit);
end component;
component or_gate
    port (d, e : in bit; f : out bit);
end component;
component inverter
    port (g : in bit; h : out bit);
end component;
begin
-- component instantiation statements
u0: and_gate port map (a => t1, b => in2, c => t3);
u1: and_gate port map (a => in1, b => t2, c => t4);
u2: inverter port map (g => in1, h => t1);
u3: inverter port map (g => in2, h => t2);
u4: or_gate port map (d => t3, e => t4, f => out1);
end structural;

Three-input Majority Voter

The entity declaration is followed by three alternative architectures which achieve the same functionality in different ways.

ENTITY maj IS
    PORT(a, b, c : IN BIT; m : OUT BIT);
END maj;
-- Dataflow style architecture
ARCHITECTURE concurrent OF maj IS
BEGIN
    -- selected signal assignment statement (concurrent)
    WITH a&b&c SELECT
    m <= '1' WHEN "110"|"101"|"011"|"111","0" WHEN OTHERS;
END concurrent;
-- Structural style architecture
ARCHITECTURE structure OF maj IS
    -- declare components used in architecture
    COMPONENT and2 PORT(in1, in2 : IN BIT; out1 : OUT BIT);
    END COMPONENT;
    COMPONENT or3 PORT(in1, in2, in3 : IN BIT; out1 : OUT BIT);
    END COMPONENT;
    -- declare local signals
    SIGNAL w1, w2, w3 : BIT;
BEGIN
    -- component instantiation statements
    -- ports of component are mapped to signals
    -- within architecture by position.
    gate1 : and2 PORT MAP (a, b, w1);
gate2 : and2 PORT MAP (b, c, w2);
gate3 : and2 PORT MAP (a, c, w3);
gate4 : or3 PORT MAP (w1, w2, w3, m);
end structural;
Examples of VHDL Descriptions

--Behavioural style architecture using a look-up table
ARCHITECTURE using_table OF major IS
BEGIN
PROCESS(a,b,c)
    CONSTANT lookuptable : BIT_VECTOR(0 TO 7) := "00010111"
    VARIABLE index : NATURAL;
BEGIN
    index := 0; --index must be cleared each time process executes
    IF a = '1' THEN index := index + 1; END IF;
    IF b = '1' THEN index := index + 2; END IF;
    IF c = '1' THEN index := index + 4; END IF;
    m <= lookuptable(index);
END PROCESS;
END using_table;

Magnitude Comparator

--VHDL description of a 4-bit magnitude comparator with expansion inputs
--first architecture demonstrates use of relational operators on
--bit vectors (=,>,<). Second architecture shows sequential behaviour
--description. Both descriptions do not fully model behaviour of real
--device for all possible combinations of inputs.
ENTITY mag4comp IS
    GENERIC(eqdel,gtdel,ltdel : TIME := 10 ns); --output delay parameters
    PORT(a,b : IN BIT_VECTOR(3 DOWNTO 0); --input words, DOWNTO ordering
        needed for comparison operators
        aeqbin,agtbin,altbin : IN BIT; --expansion inputs
    aeqbout,agtbout,altbout : OUT BIT); --outputs
END mag4comp;

ARCHITECTURE dataflow OF mag4comp IS
    --this architecture assumes that only one of the expansion inputs
    --is active at any time, if more than one expansion input is active,
    --more than one output may be active.
BEGIN
    aeqbout <= '1' AFTER eqdel WHEN ((a = b) AND (aeqbin = '1'))
        ELSE '0' AFTER eqdel;
    agtbout <= '1' AFTER gtdel WHEN ((a > b) OR ((a = b) AND (agtbin = '1')))
        ELSE '0' AFTER gtdel;
    altbout <= '1' AFTER ltdel WHEN ((a < b) OR ((a = b) AND (altbin = '1')))
        ELSE '0' AFTER ltdel;
END dataflow;

ARCHITECTURE behaviour OF mag4comp IS
BEGIN
    PROCESS(a,b,aeqbin,agtbin,altbin)
BEGIN
    IF (a > b) THEN
        agtbout <= '1' AFTER gtdel;
        aeqbout <= '0' AFTER eqdel;
        altbout <= '0' AFTER ltdel;
    ELSIF (a < b) THEN
        altbout <= '1' AFTER ltdel;
        aeqbout <= '0' AFTER eqdel;
        agtbout <= '0' AFTER gtdel;
    ELSE --a=b, expansion inputs have priority ordering
        IF (aeqbin = '1') THEN
            aeqbout <= '1' AFTER eqdel;
            agtbout <= '0' AFTER gtdel;
            altbout <= '0' AFTER ltdel;
        ELSIF (agtbin = '1') THEN
            agtbout <= '1' AFTER gtdel;
            altbout <= '0' AFTER ltdel;
            aeqbout <= '0' AFTER eqdel;
        ELSE -- a=b
            aeqbout <= '0' AFTER eqdel;
            agtbout <= '0' AFTER gtdel;
            altbout <= '0' AFTER ltdel;
        ENDIF;
    ENDIF;
    END PROCESS;
END behaviour;
Examples of VHDL Descriptions

The design entity shows the standard way of describing a register using a synchronous process, i.e., a process containing a single wait statement which is triggered by a rising edge on the clock input.

8-bit Register with Synchronous Load and Clear

The use of the std_logic literal '-' (don't care) is primarily for the synthesis tool. This example illustrates the use of the selected signal assignment.

BCD to Seven Segment Decoder

The use of the std_logic literal '-' (don't care) is primarily for the synthesis tool. This example illustrates the use of the selected signal assignment.
Examples of VHDL Descriptions

"0000001" WHEN X"9",
"-------" WHEN OTHERS;
END ver3;

2-to-4 Decoder with Testbench and Configuration

This set of design units illustrates several features of the VHDL language including:

- Using generics to pass time delay values to design entities.
- Design hierarchy using instantiated components.
- Test benches for design verification.
- Configuration declaration for binding components to design entities and setting delay values.

--ANATOMY OF A VHDL MODEL
--This VHDL source description illustrates the use
--of the basic constructs of VHDL.
--The model describes a 2-input/4-output decoder
--comprising two behavioural primitives 'inv' and 'and3'
--instanced in a structure.

ENTITY inv IS
  GENERIC(tplh,tphl,tplhe,tphle : TIME := 1 ns);
  PORT(a : IN BIT; b : OUT BIT);
END inv;

ARCHITECTURE behaviour OF inv IS
BEGIN
  PROCESS(a)
  VARIABLE state : BIT;
  BEGIN
    state := NOT(a);
    IF state = '1' THEN
      b <= state AFTER (tplh + tplhe);
    ELSE
      b <= state AFTER (tphl + tphle);
    END IF;
  END PROCESS;
END behaviour;

ENTITY and3 IS
  GENERIC(tplh,tphl,tplhe,tphle : TIME := 1 ns);
  PORT(a1,a2,a3 : IN BIT; o1 : OUT BIT);
END and3;

ARCHITECTURE behaviour OF and3 IS
BEGIN
  PROCESS(a1,a2,a3)
  VARIABLE state : BIT;
  BEGIN
    state := a1 AND a2 AND a3;
    IF state = '1' THEN
      o1 <= state AFTER (tplh + tplhe);
    ELSE
      o1 <= state AFTER (tphl + tphle);
    END IF;
  END PROCESS;
END behaviour;

ENTITY dec2to4 IS
  PORT(s0,s1,en : IN BIT; y0,y1,y2,y3 : OUT BIT);
END dec2to4;

ARCHITECTURE structural OF dec2to4 IS
COMPONENT inv

http://www.ami.bolton.ac.uk/courseware/adveda/vhdl/vhdlexmp.html (9 of 67) [23/1/2002 4:15:08 ]
Examples of VHDL Descriptions

```
PORT(a : IN BIT; b : OUT BIT); END COMPONENT;

COMPONENT and3
PORT(a1,a2,a3 : IN BIT; o1 : OUT BIT); END COMPONENT;

SIGNAL ns0,ns1 : BIT;
BEGIN
  i1 : inv PORT MAP(s0,ns0);
  i2 : inv PORT MAP(s1,ns1);
  a1 : and3 PORT MAP(en,ns0,ns1,y0);
  a2 : and3 PORT MAP(en,s0,ns1,y1);
  a3 : and3 PORT MAP(en,ns0,s1,y2);
  a4 : and3 PORT MAP(en,s0,s1,y3);
END structural;

-------------------------------------------------------------------

ENTITY dec2to4_stim IS
  PORT(stimulus : OUT BIT_VECTOR(0 TO 2); response : IN BIT_VECTOR(0 TO 3));
END dec2to4_stim;

ARCHITECTURE behavioural OF dec2to4_stim IS
BEGIN
  stimulus <= TRANSPORT "000" AFTER 0 ns,
               "100" AFTER 100 ns,
               "010" AFTER 200 ns,
               "110" AFTER 300 ns,
               "001" AFTER 400 ns,
               "101" AFTER 500 ns,
               "011" AFTER 600 ns,
               "111" AFTER 700 ns;
END behavioural;

-------------------------------------------------------------------

ENTITY dec2to4_bench IS
END dec2to4_bench;

ARCHITECTURE structural OF dec2to4_bench IS
  COMPONENT dec2to4
    PORT(s0,s1,en : IN BIT; y0,y1,y2,y3 : OUT BIT);
  END COMPONENT;

  COMPONENT dec2to4_stim
    PORT(stimulus : OUT BIT_VECTOR(0 TO 2); response : IN BIT_VECTOR(0 TO 3));
  END COMPONENT;

  SIGNAL stimulus : BIT_VECTOR(0 TO 2);
  SIGNAL response : BIT_VECTOR(0 TO 3);

BEGIN
  generator : dec2to4_stim PORT MAP(stimulus,response);
  circuit : dec2to4 PORT MAP(stimulus(1),stimulus(2),stimulus(0),
                               response(0),response(1),response(2),response(3));
END structural;

---------------------------------------------------------------

CONFIGURATION parts OF dec2to4_bench IS
  FOR structural
    FOR generator : dec2to4_stim
      USE ENTITY work.dec2to4_stim(behavioural);
    END FOR;
    FOR circuit : dec2to4
      USE ENTITY work.dec2to4(structural);
    END FOR;
    FOR ALL : inv
      USE ENTITY work.inv(behaviour)
      GENERIC MAP(tpsh => 10 ns);
  END FOR;
```

http://www.ami.bolton.ac.uk/courswear/adveda/vhdl/vhdlexmp.html (10 of 67) [23/1/2002 4:15:08]
Examples of VHDL Descriptions

library ieee;
use ieee.std_logic_1164.all;

entity tff is
port(clk, t, clear : in std_logic; q : buffer std_logic);
end tff;

architecture v1 of tff is
begin
process(clear, clk)
begin
if clear = '1' then
q <= '0';
elsif rising_edge(clk) then
if t = '1' then
q <= not q;
else
null;
end if;
end if;
end process;
end v1;

entity bigcntr is
generic(size : positive := 32);
port(clk, clear : in std_logic; q : buffer std_logic_vector((size-1) downto 0));
end bigcntr;

architecture v1 of bigcntr is
component tff is
port(clk, t, clear : in std_logic; q : buffer std_logic);
end component;
signal tin : std_logic_vector((size-1) downto 0);

begin
gentff : for i in (size-1) downto 0 generate
     ttype : tff port map (clk, tin(i), clear, q(i));
end generate;
genand : for i in 0 to (size-1) generate
     t0 : if i = 0 generate
      ...;
end generate;
end architecture;

Generated Binary Up Counter
The first design entity is a T-type flip-flop. The second is an scalable synchronous binary up counter illustrating the use of the generate statement to produce regular structures of components.
Examples of VHDL Descriptions

```vhdl
-- tin(i) <= '1';
end generate;
t1_size : if i > 0 generate
  tin(i) <= q(i-1) and tin(i-1);
end generate;
end generate;
end v1;
```

**Counter using Multiple Wait Statements**

This example shows an inefficient way of describing a counter.

--vhdl model of a 3-state counter illustrating the use
--of the WAIT statement to suspend a process. At each wait
--statement the simulation time is updated one cycle, transferring
--the driver value to the output count.
--This architecture shows that there is no difference between
--WAIT UNTIL (clock'EVENT AND clock = '1') and WAIT UNTIL clock = '1'

```vhdl
ENTITY cntr3 IS
  PORT(clock : IN BIT; count : OUT NATURAL);
END cntr3;
ARCHITECTURE using_wait OF cntr3 IS
BEGIN
  PROCESS
  BEGIN
    --WAIT UNTIL (clock'EVENT AND clock = '1');
    WAIT UNTIL clock = '1';
    count <= 0;
    --WAIT UNTIL (clock'EVENT AND clock = '1');
    WAIT UNTIL clock = '1';
    count <= 1;
    --WAIT UNTIL (clock'EVENT AND clock = '1');
    WAIT UNTIL clock = '1';
    count <= 2;
  END PROCESS;
END using_wait;
```

**Counter using a Conversion Function**

This counter uses a natural number to hold the count value and converts it into a bit_vector for output. Illustrates the use of a function.

--4-bit binary up counter with asynchronous reset 2/2/93

```vhdl
ENTITY cntr4bit IS
  PORT(reset,clock : IN BIT; count : OUT BIT_VECTOR(0 TO 3));
END cntr4bit;
ARCHITECTURE dataflow OF cntr4bit IS
--interface function to generate output bit_vector from
--internal count value.
FUNCTION nat_to_bv(input : NATURAL; highbit : POSITIVE) RETURN BIT_VECTOR IS
  VARIABLE temp : NATURAL := 0;
  VARIABLE output : BIT_VECTOR(0 TO highbit);
BEGIN
  temp := input;
  --check that input fits into (highbit+1) bits
  ASSERT (temp <= (2**(highbit + 1) - 1))
  REPORT "input no. is out of range" SEVERITY ERROR;
  FOR i IN 0 TO highbit LOOP
    output(i) <= temp MOD 2;
    temp := temp DIV 2;
  END LOOP;
  RETURN output;
END nat_to_bv;
```

http://www.ami.bolton.ac.uk/courseware/adveda/vhdl/vhlexmp.html (12 of 67) [23/1/2002 4:15:08]
Examples of VHDL Descriptions

-- generate bit values
FOR i IN highbit DOWNTO 0 LOOP
  IF temp >= (2**i) THEN
    output(i) := '1';
    temp := temp - (2**i);
  ELSE
    output(i) := '0';
  END IF;
END LOOP;
RETURN output;
END nat_to_bv;

-- signal to hold current count value
SIGNAL intcount : NATURAL := 0;
BEGIN
  -- conditional natural signal assignment models counter
  intcount <= 0 WHEN (reset = '1' ELSE ((intcount + 1) MOD 16) WHEN (clock'EVENT AND clock = '1') ELSE intcount;
  -- interface function converts natural count to bit_vector count
  count <= nat_to_bv(intcount,3);
END;

Quad 2-input Nand

Simple concurrent model of a TTL quad nand gate.
-- uses 1993 std VHDL
library IEEE;
use IEEE.Std_logic_1164.all;
entity HCT00 is
  port(A1, B1, A2, B2, A3, B3, A4, B4 : in std_logic;
       Y1, Y2, Y3, Y4 : out std_logic);
end HCT00;
architecture VER1 of HCT00 is
begin
  Y1 <= A1 nand B1 after 10 ns;
  Y2 <= A2 nand B2 after 10 ns;
  Y3 <= A3 nand B3 after 10 ns;
  Y4 <= A4 nand B4 after 10 ns;
end VER1;

Dual 2-to-4 Decoder

A set of conditional signal assignments model a dual 2-to-4 decoder
-- uses 1993 std VHDL
library IEEE;
use IEEE.Std_logic_1164.all;
entity HCT139 is
  port(A2, B2, G2BAR, A1, B1, G1BAR : in std_logic;
       Y20, Y21, Y22, Y23, Y10, Y11, Y12, Y13 : out std_logic);
end HCT139;
architecture VER1 of HCT139 is
begin
  Y10 <= '0' when (B1 = '0') and ((A1 = '0') and (G1BAR = '0')) else '1';
  Y11 <= '0' when (B1 = '0') and ((A1 = '1') and (G1BAR = '0')) else '1';
  Y12 <= '0' when (B1 = '1') and ((A1 = '0') and (G1BAR = '0')) else '1';
  Y13 <= '0' when (B1 = '1') and ((A1 = '1') and (G1BAR = '0')) else '1';
  Y20 <= '0' when (B2 = '0') and ((A2 = '0') and (G2BAR = '0')) else '1';
  Y21 <= '0' when (B2 = '0') and ((A2 = '1') and (G2BAR = '0')) else '1';
end VER1;
Examples of VHDL Descriptions

Quad D-Type Flip-flop

This example shows how a conditional signal assignment statement could be used to describe sequential logic (it is more common to use a process). The keyword 'unaffected' is equivalent to the 'null' statement in the sequential part of the language. The model would work exactly the same without the clause 'else unaffected' attached to the end of the statement.

```vhdl
Y21 <= '0' when (B2 = '0') and ((A2 = '1') and (G2BAR = '0')) else '1';
Y22 <= '0' when (B2 = '1') and ((A2 = '0') and (G2BAR = '0')) else '1';
Y23 <= '0' when (B2 = '1') and ((A2 = '1') and (G2BAR = '0')) else '1';
end VER1;
```

Octal Bus Transceiver

This example shows the use of the high impedance literal 'Z' provided by std_logic. The aggregate '(others => 'Z')' means all of the bits of B must be forced to 'Z'. Ports A and B must be resolved for this model to work correctly (hence std_logic rather than std_ulogic).

```vhdl
library IEEE;
use IEEE.Std_logic_1164.all;
entity HCT245 is
port(A, B : inout std_logic_vector(7 downto 0);
DIR, GBAR : in std_logic);
end HCT245;
architecture VER1 of HCT245 is
begin
A <= B when (GBAR = '0') and (DIR = '0') else (others => 'Z');
B <= A when (GBAR = '0') and (DIR = '1') else (others => 'Z');
end VER1;
```

Quad 2-input OR

```vhdl
library IEEE;
use IEEE.Std_logic_1164.all;
entity HCT32 is
port(A1, B1, A2, B2, A3, B3, A4, B4 : in std_logic;
Y1, Y2, Y3, Y4 : out std_logic);
end HCT32;
architecture VER1 of HCT32 is
begin
Y1 <= A1 or B1 after 10 ns;
Y2 <= A2 or B2 after 10 ns;
Y3 <= A3 or B3 after 10 ns;
Y4 <= A4 or B4 after 10 ns;
end VER1;
```
8-bit Identity Comparator

-- uses 1993 std VHDL
library IEEE;
use IEEE.Std_logic_1164.all;

entity HCT688 is
  port(Q, P : in std_logic_vector(7 downto 0);
       GBAR : in std_logic; PEQ : out std_logic);
end HCT688;

architecture VER1 of HCT688 is
begin
  PEQ <= '0' when ((To_X01(P) = To_X01(Q)) and (GBAR = '0')) else '1';
end VER1;

Hamming Encoder

A 4-bit Hamming Code encoder using concurrent assignments. The output vector is connected to the individual parity bits using an aggregate assignment.

ENTITY hamenc IS
  PORT(datain : IN BIT_VECTOR(0 TO 3);    --d0 d1 d2 d3
       hamout : OUT BIT_VECTOR(0 TO 7));  --d0 d1 d2 d3 p0 p1 p2 p4
END hamenc;

ARCHITECTURE ver2 OF hamenc IS
  SIGNAL p0, p1, p2, p4 : BIT;    --check bits
BEGIN
  --generate check bits
  p0 <= (datain(0) XOR datain(1)) XOR datain(2);
p1 <= (datain(0) XOR datain(1)) XOR datain(3);
p2 <= (datain(0) XOR datain(2)) XOR datain(3);
p4 <= (datain(1) XOR datain(2)) XOR datain(3);
  --connect up outputs
  hamout(4 TO 7) <= (p0, p1, p2, p4);
  hamout(0 TO 3) <= datain(0 TO 3);
END ver2;

Hamming Decoder

This Hamming decoder accepts an 8-bit Hamming code (produced by the encoder above) and performs single error correction and double error detection.

ENTITY hamdec IS
  PORT(hamin : IN BIT_VECTOR(0 TO 7);  --d0 d1 d2 d3 p0 p1 p2 p4
       dataout : OUT BIT_VECTOR(0 TO 3); --d0 d1 d2 d3
       sec, ded, ne : OUT BIT);  --diagnostic outputs
END hamdec;

ARCHITECTURE ver1 OF hamdec IS
BEGIN
  PROCESS(hamin)
  VARIABLE syndrome : BIT_VECTOR(3 DOWNTO 0);
  BEGIN
Examples of VHDL Descriptions

```vhdl
-- generate syndrome bits
syndrome(0) := (((((((hamin(0) XOR hamin(1)) XOR hamin(2)) XOR hamin(3))
XOR hamin(4)) XOR hamin(5)) XOR hamin(6)) XOR hamin(7));
syndrome(1) := (((hamin(0) XOR hamin(1)) XOR hamin(3)) XOR hamin(5));
syndrome(2) := (((hamin(0) XOR hamin(2)) XOR hamin(3)) XOR hamin(6));
syndrome(3) := (((hamin(1) XOR hamin(2)) XOR hamin(3)) XOR hamin(7));
IF (syndrome = "0000") THEN    -- no errors
    ne <= '1';
    ded <= '0';
    sec <= '0';
dataout(0 TO 3) <= hamin(0 TO 3);
ELSIF (syndrome(0) = '1') THEN    -- single bit error
    ne <= '0';
    ded <= '0';
    sec <= '1';
dataout(0 TO 3) <= hamin(0 TO 3);
CASE syndrome(3 DOWNTO 1) IS
    WHEN "000" | "001" | "010" | "100" =>
        dataout(0 TO 3) <= hamin(0 TO 3);    -- parity errors
    WHEN "011" => dataout(0) <= NOT hamin(0);
        dataout(1 TO 3) <= hamin(1 TO 3);
    WHEN "101" => dataout(1) <= NOT hamin(1);
        dataout(0 TO 2) <= hamin(0 TO 2);
    WHEN "110" => dataout(2) <= NOT hamin(2);
        dataout(0 TO 1) <= hamin(0 TO 1);
    WHEN "111" => dataout(3) <= NOT hamin(3);
        dataout(0 TO 2) <= hamin(0 TO 2);
END CASE;
-- double error
ELSIF (syndrome(0) = '0') AND (syndrome(3 DOWNTO 1) /= "000") THEN
    ne <= '0';
    ded <= '1';
    sec <= '0';
dataout(0 TO 3) <= "0000";
END IF;
END PROCESS;
END ver1;
```

Synchronous Down Counter with Parallel Load

This example shows the use of the package 'std_logic_unsigned'. The minus operator '-' is overloaded by this package, thereby allowing an integer to be subtraced from a std_logic_vector.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
ENTITY pldcntr8 IS
    PORT (clk, load : IN Std_logic;
          datain : IN Std_logic_vector(7 DOWNTO 0);
          q : OUT Std_logic_vector(7 DOWNTO 0);
          tc : OUT Std_logic);
END pldcntr8;
ARCHITECTURE using_std_logic OF pldcntr8 IS
    SIGNAL count : Std_logic_vector(7 DOWNTO 0);
BEGIN
```

Examples of VHDL Descriptions

```vhdl
PROCESS
BEGIN
WAIT UNTIL rising_edge(clk);
IF load = '1' THEN
  count <= datain;
ELSE
  count <= count - 1;
END IF;
END PROCESS;

tc <= '1' WHEN count = "00000000" ELSE '0';
q <= count;
END using_std_logic;
```

Mod-16 Counter using JK Flip-flops

Structural description of a 4-bit binary counter. The first two design entities describe a JK flip-flop and a 2-input AND gate respectively. These are then packaged together along with a signal named ‘tied_high’ into a package named ‘jkpack’. The counter design uses the package ‘jkpack’, giving it access to the components and the signal declared within the package. The flip-flops and AND-gates are wired together to form a counter. Notice the use of the keyword OPEN to indicate an open-cct output port.

```vhdl
ENTITY jkff IS
  PORT(clock, j, k : IN BIT; q, qbar : BUFFER BIT);
END jkff;

ARCHITECTURE using_process OF jkff IS
BEGIN
  --sequential process to model JK flip-flop
  PROCESS
    VARIABLE state : BIT := '0';
  BEGIN
    --synchronise process to rising edge of clock
    WAIT UNTIL (clock'EVENT AND clock = '1');
    IF (j = '1' AND k = '1') THEN --toggle
      state := NOT state;
    ELSIF (j = '0' AND k = '1') THEN --reset
      state := '0';
    ELSIF (j = '1' AND k = '0') THEN --set
      state := '1';
    ELSE --no change
      state := state;
    END IF;
    --assign values to output signals
    q <= state AFTER 5 ns;
    qbar <= NOT state AFTER 5 ns;
  END PROCESS;
END using_process;
```

```vhdl
ENTITY and_gate IS
  PORT(a, b : IN BIT; f : OUT BIT);
END and_gate;

ARCHITECTURE simple OF and_gate IS
BEGIN
  f <= a AND b AFTER 2 ns;
END simple;
```

```vhdl
PACKAGE jkpack IS
  ...
END jkpack;
```

ENTITY jkff IS
  PORT(clock, j, k : IN BIT; q, qbar : BUFFER BIT);
END jkff;

ARCHITECTURE using_process OF jkff IS
BEGIN
  --sequential process to model JK flip-flop
  PROCESS
    VARIABLE state : BIT := '0';
  BEGIN
    --synchronise process to rising edge of clock
    WAIT UNTIL (clock'EVENT AND clock = '1');
    IF (j = '1' AND k = '1') THEN --toggle
      state := NOT state;
    ELSIF (j = '0' AND k = '1') THEN --reset
      state := '0';
    ELSIF (j = '1' AND k = '0') THEN --set
      state := '1';
    ELSE --no change
      state := state;
    END IF;
    --assign values to output signals
    q <= state AFTER 5 ns;
    qbar <= NOT state AFTER 5 ns;
  END PROCESS;
END using_process;
```
Examples of VHDL Descriptions

```vhdl
SIGNAL tied_high : BIT := '1';

COMPONENT jkff
  PORT(clock, j, k : IN BIT; q, qbar : BUFFER BIT);
END COMPONENT;

COMPONENT and_gate
  PORT(a, b : IN BIT; f : OUT BIT);
END COMPONENT;

END jkpack;

USE work.jkpack.ALL;

ENTITY mod16_cntr IS
  PORT(clock : IN BIT; count : BUFFER BIT_VECTOR(0 TO 3));
END mod16_cntr;

ARCHITECTURE net_list OF mod16_cntr IS
  SIGNAL s1, s2 : BIT;
  BEGIN
    a1 : and_gate PORT MAP (count(0), count(1), s1);
    a2 : and_gate PORT MAP (s1, count(2), s2);
    jk1 : jkff PORT MAP (clock, tied_high, tied_high, count(0), OPEN);
    jk2 : jkff PORT MAP (clock, count(0), count(0), count(1), OPEN);
    jk3 : jkff PORT MAP (clock, s1, s1, count(2), OPEN);
    jk4 : jkff PORT MAP (clock, s2, s2, count(3), OPEN);
  END net_list;
```

Pseudo Random Bit Sequence Generator

This design entity uses a single conditional signal assignment statement to describe a PRBSG register. The length of the register and the two tapping points are defined using generics. The `&` (aggregate) operator is used to form a vector comprising the shifted contents of the register combined with the XOR feedback which is clocked into the register on the rising edge.

```vhdl
--The following Design Entity defines a parameterised Pseudo-random
--bit sequence generator, it is useful for generating serial or parallel test
--waveforms.
--(for parallel waveforms you need to add an extra output port)
--The generic 'length' is the length of the register minus one.
--the generics 'tap1' and 'tap2' define the feedback taps

ENTITY prbsgen IS
  GENERIC(length : Positive := 8; tap1 : Positive := 8; tap2 : Positive := 4);
  PORT(clk, reset : IN Bit; prbs : OUT Bit);
END prbsgen;

ARCHITECTURE v2 OF prbsgen IS
  --create a shift register
  SIGNAL prreg : Bit_Vector(length DOWNTO 0);
  BEGIN
    --conditional signal assignment shifts register and feeds in xor value
    prreg <= (0 => '1', OTHERS => '0') WHEN reset = '1' ELSE
      (prreg((length - 1) DOWNTO 0) & (prreg(tap1) XOR prreg(tap2))) WHEN clk'EVENT AND clk = '1' ELSE
      prreg;
    --connect msb of register to output
    prbs <= prreg(length);
  END v2;
```
Examples of VHDL Descriptions

**Pelican Crossing Controller**

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity pelcross is
  port(clock, reset, pedestrian : in std_logic;
       red, amber, green : out std_logic); --traffic lights
end pelcross;

architecture v1 of pelcross is
  signal en, st, mt, lt, fr : std_logic;

  --timer for light sequence
  interval_timer : block
    constant stime : natural := 50;
    constant mtime : natural := 80;
    constant ltime : natural := 200;
    signal tcount : natural range 0 to ltime;
  begin
    process
      begin
        wait until rising_edge(clock);
        if (en = '0') or (tcount = ltime) then
          tcount <= 0;
        else
          tcount <= tcount + 1;
        end if;
    end process;
    st <= '1' when tcount = stime else '0';
    mt <= '1' when tcount = mtime else '0';
    lt <= '1' when tcount = ltime else '0';
  end block;

  --free running timer for amber flashing
  free_run : block
    constant frtime : natural := 5;
    signal frcount : natural range 0 to frtime;
  begin
    process
      begin
        wait until rising_edge(clock);
        if frcount = frtime then
          frcount <= 0;
        else
          frcount <= frcount + 1;
        end if;
    end process;
    fr <= '1' when frcount = frtime else '0';
  end block;

  --moore state machine to control light sequence
  controller : block
    type peltype is (res, stop, amb, amb_on, amb_off, grn, ped);
    signal pelstate : peltype;
  begin
    process(clock, reset)
      begin
        if reset = '1' then
          pelstate <= res;
        elsif rising_edge(clock) then
          case pelstate is
            when res => pelstate <= stop;
            when stop => if lt = '1' then
                          pelstate <= amb;
                          when others => pelstate <= res;
          end case;
        end if;
      end process;
    end process;
  end block;
end v1;
```

Examples of VHDL Descriptions

```
else
  pelstate <= stop;
end if;

when amb => pelstate <= amb_on;
when amb_on => if mt = '1' then
  pelstate <= grn;
elsif fr = '1' then
  pelstate <= amb_off;
else
  pelstate <= amb_on;
end if;
when amb_off => if mt = '1' then
  pelstate <= grn;
elsif fr = '1' then
  pelstate <= amb_on;
else
  pelstate <= amb_off;
end if;
when grn => if pedestrian = '1' then
  pelstate <= ped;
else
  pelstate <= grn;
end if;
when ped => if st = '1' then
  pelstate <= res;
else
  pelstate <= ped;
end if;
when others => pelstate <= res;
end case;
end if;
end process;
```

```
--moore outputs
with pelstate select
  en <= '1' when stop|amb_on|amb_off|ped,
       '0' when others;
with pelstate select
  red <= '1' when res|stop,
       '0' when others;
with pelstate select
  amber <= '1' when amb|amb_on|ped,
       '0' when others;
with pelstate select
  green <= '1' when grn,
       '0' when others;
end block;
```

```
--Pelican Crossing Controller test bench
library ieee;
use ieee.std_logic_1164.all;
entity peltest is
  end peltest;
architecture v1 of peltest is
  signal clock, reset, pedestrian, red, amber, green : std_logic;
component pelcross is
  port(clock, reset, pedestrian : in std_logic;
       red, amber, green : out std_logic); --traffic lights
end component;
begin
  --10 Hz clock generator
  process begin
    --10 Hz clock generator
  ```
Examples of VHDL Descriptions

clock <= '0', '1' after 50 ms;
wait for 100 ms;
end process;

--test inputs
process begin
pedestrian <= '0';
reset <= '1';
wait for 300 ms;
reset <= '0';
wait for 40000 ms;
pedestrian <= '1';
wait for 200 ms;
pedestrian <= '0';
wait;
end process;

pelican : pelcross port map (clock, reset, pedestrian,
                          red, amber, green);
end v1;

Simple Microprocessor System

- Package Defining the Instruction Set of the CPU
- Third Party Package containing functions for Bit_Vector operations
- Behavioural model of a 256-word, 8-bit Read Only Memory
- Behavioural model of a 16-word, 8-bit Random Access Memory
- Behavioural model of a simple 8-bit CPU
- Structural description of a microprocessor system using the above components

Package Defining the Instruction Set of the CPU

PACKAGE cpu8pac IS
   --defining instruction set
   --instruction format
   --  7----4|3--0|7----------0
       opcode|page|[page offset]
   --instructions which need an address are two bytes
   --long all others are single byte
   CONSTANT lda : BIT_VECTOR(3 DOWNTO 0) := "0001";
   CONSTANT ldb : BIT_VECTOR(3 DOWNTO 0) := "0010";
   CONSTANT sta : BIT_VECTOR(3 DOWNTO 0) := "0011";
   CONSTANT stb : BIT_VECTOR(3 DOWNTO 0) := "0000";
   CONSTANT jmp : BIT_VECTOR(3 DOWNTO 0) := "0100";
   CONSTANT add : BIT_VECTOR(3 DOWNTO 0) := "0101";
   CONSTANT subr : BIT_VECTOR(3 DOWNTO 0) := "0110";
   CONSTANT inc : BIT_VECTOR(3 DOWNTO 0) := "0111";
   CONSTANT dec : BIT_VECTOR(3 DOWNTO 0) := "1000";
   CONSTANT land : BIT_VECTOR(3 DOWNTO 0) := "1001";
   CONSTANT lor : BIT_VECTOR(3 DOWNTO 0) := "1010";
   CONSTANT cmp : BIT_VECTOR(3 DOWNTO 0) := "1011";
   CONSTANT lxor : BIT_VECTOR(3 DOWNTO 0) := "1100";
   CONSTANT lita : BIT_VECTOR(3 DOWNTO 0) := "1101";
   CONSTANT litb : BIT_VECTOR(3 DOWNTO 0) := "1110";
   CONSTANT clra : BIT_VECTOR(3 DOWNTO 0) := "1111";
END cpu8pac;

Third Party Package containing functions for Bit_Vector operations

EXAMPLES OF VHDL DESCRIPTIONS

-- Cypress Semiconductor WARP 2.0

-- Copyright Cypress Semiconductor Corporation, 1994
-- as an unpublished work.

-- $Id: libbv.vhd,v 1.4 1994/12/15 18:35:28 hemmert Exp $

package bv_math

-- Bit Vector support package:

-- Contains these functions:
-- The output length of the function is the same as the input length.

-- inc_bv - increment a bit vector. If function is assigned
-- to a signal within a clocked process, the result
-- will be an up counter. Will require one macrocell
-- for each bit.

-- dec_bv - decrement a bit vector. If function is assigned
-- to a signal within a clocked process, the result
-- will be a down counter. Will require one macrocell
-- for each bit.

-- "+" regular addition function for two bit vectors.
-- "+" operator overloads the existing"+" operator
-- definition for arithmetic operations on integers.
-- Will require one macrocell for each bit. The output
-- is the same size as the input so there is no carry output.
-- If a carry out is required, the user should increase the
-- size of the input bit_vectors and use the MSB as the
-- carry_bit. There is also no separate carry-in.

-- "-" regular subtraction function for two bit vectors.
-- "-" operator overloads the existing"-" operator
-- definition for arithmetic operations on integers.

-- inv unary invert for use in port maps and sequential
-- assignments. Overloaded for bit vectors.

PACKAGE bv_math IS
FUNCTION inc_bv (a : BIT_VECTOR) RETURN BIT_VECTOR;
FUNCTION dec_bv (a : BIT_VECTOR) RETURN BIT_VECTOR;
FUNCTION "-" (a, b : BIT_VECTOR) RETURN BIT_VECTOR;
FUNCTION "-" (a : BIT_VECTOR; b : BIT) RETURN BIT_VECTOR;
FUNCTION inv (a : BIT) RETURN BIT;
FUNCTION inv (a : BIT_VECTOR) RETURN BIT_VECTOR;
END bv_math;

PACKAGE BODY bv_math IS
-- inc_bv
-- Increment Bit vector.
-- In: bit_vector.
-- Return: bit_vector.

FUNCTION inc_bv(a : BIT_VECTOR)RETURN BIT_VECTOR IS
VARIABLE s : BIT_VECTOR (a'RANGE);
VARIABLE carry : BIT;
BEGIN
    carry := '1';
    FOR i IN a'LOW TO a'HIGH LOOP
        s(i) := a(i) XOR carry;
        carry := a(i) AND carry;
    END LOOP;
    RETURN s;
END;

Examples of VHDL Descriptions

END LOOP;
RETURN (s);
END inc_bv;

-- "+
-- Add overload for:
-- In: two bit_vectors.
-- Return: bit_vector.

FUNCTION "+"(a, b : BIT_VECTOR) RETURN BIT_VECTOR IS
VARIABLE s : BIT_VECTOR (a'LENGTH);
VARIABLE carry : BIT;
VARIABLE bi : integer;  -- Indexes b.
BEGIN
ASSERT a'LENGTH <= 8 REPORT
"Addition Of vectors Of LENGTH > 8 may take exponential TIME."
SEVERITY WARNING;
carry := '0';
FOR i IN a'LOW TO a'HIGH LOOP
bi := b'low + (i - a'low);
s(i) := (a(i) XOR b(bi)) XOR carry;
carry := ((a(i) OR b(bi)) AND carry) OR (a(i) AND b(bi));
END LOOP;
RETURN (s);
END "+";  -- Two bit_vectors.

-- "+
-- Add overload for:
-- In: bit_vector and bit.
-- Return: bit_vector.

FUNCTION "+"(a : BIT_VECTOR; b : BIT) RETURN BIT_VECTOR IS
VARIABLE s : BIT_VECTOR (a'LENGTH);
BEGIN
carry := b;
FOR i IN a'LOW TO a'HIGH LOOP
s(i) := a(i) XOR carry;
carry := a(i) AND carry;
END LOOP;
RETURN (s);
END "+";  -- Bit_vector and bit.

-- dec_bv
-- Decrement Bit Vector
-- In: bit_vector.
-- Return: bit_vector.

FUNCTION dec_bv(a : BITVECTOR) RETURN BIT_VECTOR IS
VARIABLE s : BIT_VECTOR (a'LENGTH);
VARIABLE borrow : BIT;
BEGIN
borrow := '1';
FOR i IN a'LOW TO a'HIGH LOOP
s(i) := a(i) XOR borrow;
borrow := NOT (a(i) AND borrow);
END LOOP;
RETURN (s);
END dec_bv;

-- "-
-- Subtract overload for:

Examples of VHDL Descriptions

FUNCTION "-"(a,b : BIT_VECTOR) RETURN BIT_VECTOR IS
  VARIABLE s     : BIT_VECTOR (a'RANGE);
  VARIABLE borrow : BIT;
  VARIABLE bi     : integer;  -- Indexes b.
BEGIN
  ASSERT a'LENGTH <= 8 REPORT
  "Subtraction OF vectors OF LENGTH > 8 may take exponential TIME."
  SEVERITY WARNING;
  borrow := '0';
  FOR i IN a'LOW TO a'HIGH LOOP
    bi := b'low + (i - a'low);
    s(i) := (a(i) XOR b(bi)) XOR borrow;
    borrow := (NOT (a(i)) AND borrow)
      OR  (b(bi)      AND borrow)
      OR  (NOT (a(i)) AND b(bi))
      OR  (bi = '1');
  END LOOP;
  RETURN  (s);
END "-";  -- two bit_vectors

FUNCTION "-"(a : BIT_VECTOR; b : BIT) RETURN BIT_VECTOR IS
  VARIABLE s     : BIT_VECTOR (a'RANGE);
  VARIABLE borrow : BIT;
BEGIN
  borrow := b;
  FOR i IN a'LOW TO a'HIGH LOOP
    s(i) := a(i) XOR borrow;
    borrow := (NOT a(i)) AND borrow;
  END LOOP;
  RETURN  (s);
END "-";

FUNCTION inv(a : BIT) RETURN BIT IS
BEGIN
  result  := NOT(a);
  RETURN  (result);
END inv;  -- Invert bit.

FUNCTION inv(a : BIT_VECTOR) RETURN BIT_VECTOR IS
BEGIN
  FOR i IN a'RANGE LOOP
    result(i) := NOT(a(i));
  END LOOP;
  RETURN  (result);
END inv;  -- Invert bit_vector.

FUNCTION inv(a : BIT_VECTOR) RETURN BIT_VECTOR IS
BEGIN
  FOR i IN a'RANGE LOOP
    result := NOT(a(i));
  END LOOP;
  RETURN  (result);
END inv;  -- Invert bit_vector.
Examples of VHDL Descriptions

END bv_math;

Behavioural model of a 256-word, 8-bit Read Only Memory

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.cpu8pac.ALL;
ENTITY rom256x8 IS
  PORT(address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
       csbar, oebar : IN STD_LOGIC;
       data : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END rom256x8;

ARCHITECTURE version1 OF rom256x8 IS
  TYPE rom_array IS ARRAY (0 TO 255) OF BIT_VECTOR(7 DOWNTO 0);
  CONSTANT rom_values : rom_array :=
  (0 => clra & X"0",
   1 => lda & X"FE",
   2 => X"FF",
   3 => ldb & X"0",
   4 => X"FE",
   5 => lxor & X"0",
   6 => jmp & X"0",
   7 => X"01",
   254 => X"AA",
   255 => X"55",
   OTHERS => X"00");
  BEGIN
  PROCESS(address, csbar, oebar)
  VARIABLE index : INTEGER := 0;
  BEGIN
  IF (csbar = '1' OR oebar = '1')
  THEN data <= "ZZZZZZZZ"
  ELSE
    --calculate address as an integer
    index := 0;
    FOR i IN address'RANGE LOOP
      IF address(i) = '1' THEN
        index := index + 2**i;
      END IF;
    END LOOP;
    --assign to output data lines
    data <= To_StdlogicVector(rom_values(index));
  END IF;
  END PROCESS;
END version1;

ARCHITECTURE version2 OF rom256x8 IS
  TYPE rom_array IS ARRAY (0 TO 255) OF BIT_VECTOR(7 DOWNTO 0);
  CONSTANT rom_values : rom_array :=
  (0 => clra & X"0",
   1 => sta & X"1",
   2 => X"00",
   3 => lda & X"1",
   4 => X"00",
   5 => inc & X"0",
   6 => jmp & X"0",
   7 => X"01",
   OTHERS => X"00");
  BEGIN
  PROCESS(address, csbar, oebar)
  VARIABLE index : INTEGER := 0;
  BEGIN
  IF (csbar = '1' OR oebar = '1')
  THEN data <= "ZZZZZZZZ"
  ELSE
    --calculate address as an integer
    index := 0;
    FOR i IN address'RANGE LOOP
      IF address(i) = '1' THEN
        index := index + 2**i;
      END IF;
    END LOOP;
    --assign to output data lines
    data <= To_StdlogicVector(rom_values(index));
  END IF;
  END PROCESS;
END version2;

Examples of VHDL Descriptions

```
PROCESS(address, csbar, oebar)
VARIABLE index : INTEGER := 0;
BEGIN
  IF (csbar = '1' OR oebar = '1')
  THEN data <= "ZZZZZZZZ";
  ELSE
    --calculate address as an integer
    index := 0;
    FOR i IN address'RANGE LOOP
      IF address(i) = '1' THEN
        index := index + 2**i;
      END IF;
    END LOOP;
    --assign to output data lines
    data <= To_StdlogicVector(rom_values(index));
  END IF;
END PROCESS;
END version2;

Behavioural model of a 16-word, 8-bit Random Access Memory

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ram16x8 IS
  PORT(address : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       csbar, oebar, webar : IN STD_LOGIC;
       data : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ram16x8;
ARCHITECTURE version1 OF ram16x8 IS
BEGIN
  PROCESS(address, csbar, oebar, webar, data)
  TYPE ram_array IS ARRAY (0 TO 15) OF BIT_VECTOR(7 DOWNTO 0);
  VARIABLE index : INTEGER := 0;
  VARIABLE ram_store : ram_array;
  BEGIN
    IF csbar = '0' THEN
      --calculate address as an integer
      index := 0;
      FOR i IN address'RANGE LOOP
        IF address(i) = '1' THEN
          index := index + 2**i;
        END IF;
      END LOOP;
      IF rising_edge(webar) THEN
        --write to ram on rising edge of write pulse
        ram_store(index) := To_bitvector(data);
      ELSIF oebar = '0' THEN
        data <= To_StdlogicVector(ram_store(index));
      ELSE
        data <= "ZZZZZZZZ";
      END IF;
    ELSE
      data <= "ZZZZZZZZ";
    END IF;
  END PROCESS;
END version1;
```
Examples of VHDL Descriptions

**Behavioural model of a simple 8-bit CPU**

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.bv_math.ALL;
USE work.cpu8pac.ALL;

ENTITY cpu IS
  GENERIC(cycle_time : TIME := 200 ns); --must be divisible by 8
  PORT(reset : IN std_logic;
  memrd, memwr : OUT std_logic;
  address : OUT std_logic_vector(11 DOWNTO 0);
  data : INOUT std_logic_vector(7 DOWNTO 0));
END cpu;

ARCHITECTURE version1 OF cpu IS
  --internal clock signal
  SIGNAL clock : std_logic;
  BEGIN
  clock_gen : PROCESS
  BEGIN
  clock <= '1','0' AFTER cycle_time/2;
  WAIT FOR cycle_time;
  END PROCESS;
  main_sequence : PROCESS
  VARIABLE inst_reg : BIT_VECTOR(3 DOWNTO 0);
  VARIABLE mar : BIT_VECTOR(11 DOWNTO 0);
  VARIABLE acca, accb : BIT_VECTOR(7 DOWNTO 0);
  VARIABLE pc : BIT_VECTOR(11 DOWNTO 0);
  BEGIN
  IF reset = '1' THEN
  --initialisation
  memrd <= '1';
  memwr <= '1';
  pc := (OTHERS => '0');
  address <= (OTHERS => 'Z');
  data <= (OTHERS => 'Z');
  WAIT UNTIL rising_edge(clock);
  ELSE
  --fetch phase
  address <= To_StdlogicVector(pc);
  WAIT FOR cycle_time/4;
  memrd <= '0';
  WAIT FOR cycle_time/2;
  memrd <= '1';
  --read instruction
  inst_reg := To_bitvector(data(7 DOWNTO 4));
  --load page address
  mar(11 DOWNTO 8) := To_bitvector(data(3 DOWNTO 0));
  --increment program counter
  pc := inc_bv(pc);
  --wait until end of cycle
  WAIT UNTIL rising_edge(clock);
  --execute
  CASE inst_reg IS
  WHEN add =>
  --add and sub use overloaded functions from bv_math package

Examples of VHDL Descriptions

```vhdl
acca := acca + accb;
WHEN subr =>
  acca := acca - accb;
WHEN inc =>
  acca := inc_bv(acca);
WHEN dec =>
  acca := dec_bv(acca);
WHEN land =>
  acca := acca AND accb;
WHEN lor =>
  acca := acca OR accb;
WHEN cmp =>
  acca := NOT acca;
WHEN lxor =>
  acca := acca XOR accb;
WHEN lita =>
  acca := acca;
WHEN litb =>
  acca := accb;
WHEN clra =>
  acca := (OTHERS => '0');
WHEN lda|ldb|sta|stb =>
  address <= To_StdlogicVector(pc);
  WAIT FOR cycle_time/4;
  memrd <= '0';
  WAIT FOR cycle_time/2;
  memrd <= '1';
  --read page offset address
  mar(7 DOWNTO 0) := To_bitvector(data);
  --increment program counter
  pc := inc_bv(pc);
  --wait until end of cycle
  WAIT UNTIL rising_edge(clock);
  --output address of operand
  address <= To_StdlogicVector(mar);
  IF ((inst_reg = lda) OR (inst_reg = ldb)) THEN
    WAIT FOR cycle_time/4;
    memrd <= '0';
    WAIT FOR cycle_time/2;
    memrd <= '1';
    IF inst_reg = lda THEN
      --load accumulator a from bus
      acca := To_bitvector(data);
    ELSE
      --load accumulator b from bus
      accb := To_bitvector(data);
    END IF;
    --wait until end of cycle
    WAIT UNTIL rising_edge(clock);
  ELSE
    WAIT FOR cycle_time/8;
    IF inst_reg = sta THEN
      --output data
      data <= To_StdlogicVector(acca);
    ELSE
      --output data
      data <= To_StdlogicVector(accb);
    END IF;
    --wait until end of cycle
    WAIT FOR cycle_time/8;
    memwr <= '0';
    WAIT FOR cycle_time/2;
    memwr <= '1';
    WAIT FOR cycle_time/8;
    data <= (OTHERS => 'Z');
  --wait until end of cycle
```

Examples of VHDL Descriptions

WAIT UNTIL rising_edge(clock);
END IF;

WHEN jmp =>
   address <= To_StdlogicVector(pc);
   --transfer page address to pc from mar
   pc(11 DOWNTO 8) := mar(11 DOWNTO 8);
   --read in offset address
   WAIT FOR cycle_time/4;
   memrd <= '0';
   --wait until end of cycle
   WAIT UNTIL rising_edge(clock);
   pc(7 DOWNTO 0) := To_bitvector(data);
   --wait until end of cycle
   WAIT UNTIL rising_edge(clock);
END CASE;
END IF;
END PROCESS main_sequence;
END version1;

Structural description of a Microprocessor System

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY cpudemo IS
END cpudemo;
ARCHITECTURE version1 OF cpudemo IS
COMPONENT rom256x8
   PORT(address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        csbar, oebar : IN STD_LOGIC;
        data : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;
COMPONENT ram16x8
   PORT(address : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        csbar, oebar, webar : IN STD_LOGIC;
        data : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;
COMPONENT cpu
   GENERIC(cycle_time : TIME := 200 ns); --must be divisible by 8
   PORT(reset : IN std_logic;
        memrd, memwr : OUT std_logic;
        address : OUT std_logic_vector(11 DOWNTO 0);
        data : INOUT std_logic_vector(7 DOWNTO 0));
END COMPONENT;
SIGNAL reset, memrd, memwr, romenable, ramenable : std_logic;
SIGNAL address : std_logic_vector(11 DOWNTO 0);
SIGNAL data : std_logic_vector(7 DOWNTO 0);
--selecting the rom architecture (program) for simulation
FOR rom : rom256x8 USE ENTITY work.rom256x8(version2);
BEGIN
   processor : cpu PORT MAP(reset, memrd, memwr, address, data);
   rom : rom256x8 PORT MAP(address(7 DOWNTO 0), romenable, memrd, data);
   ram : ram16x8 PORT MAP(address(3 DOWNTO 0), ramenable, memrd, memwr, data);
   --memory address decoding, rom is at bottom of address space
   --ram is situated at address $100
Examples of VHDL Descriptions

```
romenable <= '0' WHEN (address(11 DOWNTO 8) = "0000") ELSE '1';
romenable <= '0' WHEN (address(11 DOWNTO 4) = "00010000") ELSE '1';
```

```
END version1;
```

---

**Lottery Number Generator**

- Lottery Number Counter
- Lottery Number Register
- BCD to 7-segment Decoder
- Controller
- Structural Model of Lottery Number Generator

```
Lottery Number Counter

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity count49 is
    port(clock, clear : in std_logic;
         cnt1to49 : buffer std_logic_vector(7 downto 0));
end entity count49;
architecture v1 of count49 is
begin
    count_proc : process
    begin
        wait until rising_edge(clock);
        if (clear = '1') or (cnt1to49 = X"49") then
            cnt1to49 <= (0 => '1', others => '0');
        elsif cnt1to49(3 downto 0) = 9 then
            cnt1to49(3 downto 0) <= (others => '0');
            cnt1to49(7 downto 4) <= cnt1to49(7 downto 4) + 1;
        else
            cnt1to49(3 downto 0) <= cnt1to49(3 downto 0) + 1;
        end if;
    end process;
end architecture v1;
```

```
Lottery Number Register

--synchronous loadable register
library ieee;
use ieee.std_logic_1164.all;
entity lottreg is
    port(clock, clear, load : in std_logic;
         d : in std_logic_vector(7 downto 0);
         q : out std_logic_vector(7 downto 0));
end entity lottreg;
architecture v1 of lottreg is
begin
    reg_proc : process
    begin
        wait until rising_edge(clock);
        if clear = '1' then
            q <= (others => '0');
        elsif load = '1' then
            q <= d;
        else
            q <= q + 1;
        end if;
    end process;
end architecture v1;
```
Controller

--controller for lottery number generator
--new version uses 6 number registers and
--compares all numbers simultaneously
library ieee;
use ieee.std_logic_1164.all;

entity lottcont2 is
port(clock, reset, next_no, match : in std_logic;
loadnum1, loadnum2, loadnum3, loadnum4,
loadnum5, loadnum6, sample : in std_logic;
workdisplay : out natural range 0 to 5;
numled : out std_logic_vector(1 to 6));

end entity lottcont2;

architecture fsm2 of lottcont2 is

begin
--next state process
fsm_state_reg : process
begin
wait until rising_edge(clock);
if reset = '1' then
lott_ps <= res;
else
lott_ps <= lott_ns;
end if;
end process;

fsm_logic : process(lott_ps, next_no, match)
begin
--assign default output values
loadnum1 <= '0';
loadnum2 <= '0';
loadnum3 <= '0';
loadnum4 <= '0';
loadnum5 <= '0';
loadnum6 <= '0';
sample <= '0';
workdisplay <= 0;
numled <= "111111";
case lott_ps is
when res => --wait for 1st no
if next_no = '1' then
lott_ns <= s1;
else
lott_ns <= res;
end if;
when s1 => --take first sample
sample <= '1';
lott_ns <= s2;
when s2 => --save first no
loadnum1 <= '1';
numled <= "011111";
lott_ns <= s3;
when s3 => --wait for 2nd no
numled <= "011111";
end process;
end architecture;
Examples of VHDL Descriptions

if next_no = '1' then
  lott_ns <= s4;
else
  lott_ns <= s3;
end if;

when s4 => -- sample 2nd no
  numled <= "011111";
  sample <= '1';
  lott_ns <= s5;
when s5 => -- check for duplicate
  numled <= "011111";
  if match = '1' then
    lott_ns <= s4;
  else
    lott_ns <= s6;
  end if;

when s6 => -- store second number
  numled <= "101111";
  loadnum2 <= '1';
  lott_ns <= s7;
when s7 => -- wait for 3rd no
  numled <= "101111";
  seldisplay <= 1;
  if next_no = '1' then
    lott_ns <= s4;
  else
    lott_ns <= s10;
  end if;

when s8 => -- sample 3rd no
  numled <= "101111";
  seldisplay <= 1;
  sample <= '1';
  lott_ns <= s9;
when s9 => -- check against other nos
  numled <= "101111";
  seldisplay <= 1;
  if match = '1' then
    lott_ns <= s8;
  else
    lott_ns <= s14;
  end if;

when s10 => -- store 3rd no
  numled <= "110111";
  seldisplay <= 1;
  loadnum3 <= '1';
  lott_ns <= s11;
when s11 => -- wait for 4th no
  numled <= "110111";
  seldisplay <= 2;
  if next_no = '1' then
    lott_ns <= s12;
  else
    lott_ns <= s11;
  end if;

when s12 => -- sample 4th no
  numled <= "110111";
  seldisplay <= 2;
  sample <= '1';
  lott_ns <= s13;
when s13 => -- check against other nos
  numled <= "110111";
  seldisplay <= 2;
  if match = '1' then
    lott_ns <= s12;
  else
    lott_ns <= s14;
  end if;
Examples of VHDL Descriptions

when s14 => --store 4th no
  numled <= "111011";
  seldisplay <= 2;
  loadnum4 <= '1';
  lott_ns <= s15;
when s15 => --wait for 5th no
  numled <= "111011";
  seldisplay <= 3;
  if next_no = '1' then
    lott_ns <= s16;
  else
    lott_ns <= s15;
  end if;
when s16 => --sample 5th no
  numled <= "111011";
  seldisplay <= 3;
  sample <= '1';
  lott_ns <= s17;
when s17 => --check against other nos
  numled <= "111011";
  seldisplay <= 3;
  if match = '1' then
    lott_ns <= s16;
  else
    lott_ns <= s18;
  end if;
when s18 => --store 5th no
  numled <= "111101";
  seldisplay <= 3;
  loadnum5 <= '1';
  lott_ns <= s19;
when s19 => --wait for 6th no
  numled <= "111101";
  seldisplay <= 4;
  if next_no = '1' then
    lott_ns <= s20;
  else
    lott_ns <= s19;
  end if;
when s20 => --sample 6th no
  numled <= "111101";
  seldisplay <= 4;
  sample <= '1';
  lott_ns <= s21;
when s21 => --check against other nos
  numled <= "111101";
  seldisplay <= 4;
  if match = '1' then
    lott_ns <= s20;
  else
    lott_ns <= s22;
  end if;
when s22 => --store 6th no
  numled <= "111110";
  seldisplay <= 4;
  loadnum6 <= '1';
  lott_ns <= s23;
when s23 => --review numbers
  numled <= "111110";
  seldisplay <= 4;
  if next_no = '1' then
    lott_ns <= s24;
  else
    lott_ns <= s23;
  end if;
when s24 => --review 1st no
  numled <= "011111";
Examples of VHDL Descriptions

```vhdl
seldisplay <= 0;
if next_no = '1' then
  lott_ns <= s25;
else
  lott_ns <= s24;
end if;
when s25 => --review 2nd no
  numled <= "101111";
  seldisplay <= 1;
  if next_no = '1' then
    lott_ns <= s26;
  else
    lott_ns <= s25;
  end if;
when s26 => --review 3rd no
  numled <= "110111";
  seldisplay <= 2;
  if next_no = '1' then
    lott_ns <= s27;
  else
    lott_ns <= s26;
  end if;
when s27 => --review 4th no
  numled <= "111011";
  seldisplay <= 3;
  if next_no = '1' then
    lott_ns <= s28;
  else
    lott_ns <= s27;
  end if;
when s28 => --review 5th no
  numled <= "111101";
  seldisplay <= 4;
  if next_no = '1' then
    lott_ns <= s23;
  else
    lott_ns <= s28;
  end if;
when others =>
  lott_ns <= res;
end case;
end process;
end architecture fsm2;
```

Structural Model of Lottery Number Generator

```vhdl
--top level design for lottery number generator
--version 2 uses 6 number registers
library ieee;
use ieee.std_logic_1164.all;
entity lottery2 is
  port(clock, reset, next_no : in std_logic;
       numled : out std_logic_vector(1 to 6);
       seg0, seg1 : out std_logic_vector(6 downto 0));
end entity lottery2;
architecture structure of lottery2 is
  component lottreg
    port(clock, clear, load : in std_logic;
         d : in std_logic_vector(7 downto 0);
         q : out std_logic_vector(7 downto 0));
  end component;
  component count49
```
Examples of VHDL Descriptions

component seg7dec --see file bcd2seg.vhd
  PORT(bcdin : IN std_logic_vector(3 DOWNTO 0);
  segout : OUT std_logic_vector(6 DOWNTO 0));
end component;

component lottcont2
  PORT(clock, reset, next_no, match : in std_logic;
  loadnum1, loadnum2, loadnum3, loadnum4,
  loadnum5, loadnum6, sample : out std_logic;
  seldisplay : out natural range 0 to 5;
  numled : out std_logic_vector(1 to 6));
end component;

signal match : std_logic;
signal sample : std_logic;
signal seldisplay : natural range 0 to 5;
signal count, samp_reg, display : std_logic_vector(7 downto 0);
signal num_reg1, num_reg2, num_reg3 : std_logic_vector(7 downto 0);
signal num_reg4, num_reg5, num_reg6 : std_logic_vector(7 downto 0);
signal loadnum1, loadnum2, loadnum3, loadnum4, loadnum5, loadnum6 :
  std_logic;
begin
  counter : count49
    port map (clock => clock, clear => reset, cnt1to49 => count);
  sample_reg : lottreg
    port map (clock => clock, clear => reset,
      load => sample, d => count, q => samp_reg);
  --number registers
  numreg1 : lottreg port map
    (clock => clock, clear => reset, load => loadnum1,
      d => samp_reg, q => num_reg1);
  numreg2 : lottreg port map
    (clock => clock, clear => reset, load => loadnum2,
      d => samp_reg, q => num_reg2);
  numreg3 : lottreg port map
    (clock => clock, clear => reset, load => loadnum3,
      d => samp_reg, q => num_reg3);
  numreg4 : lottreg port map
    (clock => clock, clear => reset, load => loadnum4,
      d => samp_reg, q => num_reg4);
  numreg5 : lottreg port map
    (clock => clock, clear => reset, load => loadnum5,
      d => samp_reg, q => num_reg5);
  numreg6 : lottreg port map
    (clock => clock, clear => reset, load => loadnum6,
      d => samp_reg, q => num_reg6);
  compare : match <= '1' when (((samp_reg = num_reg1)
    or (samp_reg = num_reg2))
    or (samp_reg = num_reg3))
    or (samp_reg = num_reg4)
    or (samp_reg = num_reg5)
    else '0';
  display_mux : with seldisplay select
    display <= num_reg1 when 0,
      num_reg2 when 1,
      num_reg3 when 2,
Booth Multiplier

--This file contains all the entity-architectures for a complete
--k-bit x k-bit Booth multiplier.
--the design makes use of the new shift operators available in the VHDL-93 std
--this design passes the Synplify synthesis check

ENTITY booth_multiplier IS
  GENERIC(k : POSITIVE := 7); --input number word length less one
  PORT(multiplicand, multiplier : IN BIT_VECTOR(k DOWNTO 0);
    clock : IN BIT; product : INOUT BIT_VECTOR((2*k + 1) DOWNTO 0));
END booth_multiplier;

ARCHITECTURE structural OF booth_multiplier IS
  SIGNAL mdreg, adderout, carries, augend, tcbuffout : BIT_VECTOR(k DOWNTO 0);
  SIGNAL mrreg : BIT_VECTOR((k + 1) DOWNTO 0);
  SIGNAL adder_ovfl : BIT;
  SIGNAL comp, clr_mr, load_mr, shift_mr, clr_md, load_md, clr_pp, load_pp, shift_pp : BIT;
  SIGNAL boostate : NATURAL RANGE 0 TO 2*(k + 1);
BEGIN
  PROCESS --main clocked process containing all sequential elements
  BEGIN
    WAIT UNTIL (clock'EVENT AND clock = '1');
    --register to hold multiplicand during multiplication
    IF clr_md = '1' THEN
      mdreg <= (OTHERS => '0');
    ELSIF load_md = '1' THEN
      mdreg((k + 1) DOWNTO 1) <= multiplier;
      mdreg(0) <= '0';
    ELSE
      mdreg <= mdreg;
    END IF;
    --register/shifter to product pair of bits used to control adder
    IF clr_mr = '1' THEN
      mreg <= (OTHERS => '0');
    ELSIF load_mr = '1' THEN
      mreg((k + 1) DOWNTO 1) <= multiplier;
      mreg(0) <= '0';
    ELSIF shift_mr = '1' THEN
      ...
    END IF;
  END PROCESS;
  ...
END structural;
Examples of VHDL Descriptions

```vhdl
-- register/shifter accumulates partial product values
IF clr_pp = '1' THEN
  product <= (OThERS => '0');
ELSIF load_pp = '1' THEN
  product((k+1) DOWNTO (k+1)) <= adderout; --add to top half
  product(k DOWNTO 0) <= product(k DOWNTO 0); --refresh bootm half
ELSIF shift_pp = '1' THEN
  product <= product SRA 1; --shift right with sign extend
ELSE
  product <= product;
END IF;
END PROCESS;

--adder adds/subtracts partial product to multiplicand
augend <= product((2*k+1) DOWNTO (k+1));
addgen : FOR i IN adderout'RANGE
  GENERATE
  lsadder : IF i = 0 GENERATE
    adderout(i) <= tcbuffout(i) XOR augend(i) XOR comp;
    carries(i) <= (tcbuffout(i) AND augend(i)) OR
    (tcbuffout(i) AND comp) OR
    (comp AND augend(i));
  END GENERATE;
  otheradder : IF i /= 0 GENERATE
    adderout(i) <= tcbuffout(i) XOR augend(i) XOR carries(i-1);
    carries(i) <= (tcbuffout(i) AND augend(i)) OR
    (tcbuffout(i) AND carries(i-1)) OR
    (carries(i-1) AND augend(i));
  END GENERATE;
  END GENERATE;

--twos comp overflow bit
adder_ovfl <= carries(k-1) XOR carries(k);

--true/complement buffer to generate two's comp of mdreg
  tcbuffout <= NOT mdreg WHEN (comp = '1') ELSE mdreg;

--booth multiplier state counter
PROCESS BEGIN
  WAIT UNTIL (clock'EVENT AND clock = '1');
  IF boostate < 2*(k + 1) THEN boostate <= boostate + 1;
  ELSE boostate <= 0;
  END IF;
END PROCESS;

--assign control signal values based on state
PROCESS(boostate)
BEGIN
  --assign defaults, all registers refresh
  comp <= '0';
  clr_mr <= '0';
  load_mr <= '0';
  shift_mr <= '0';
  clr_md <= '0';
  load_md <= '0';
  clr_pp <= '0';
  load_pp <= '0';
  shift_pp <= '0';
  IF boostate = 0 THEN
    load_mr <= '1';
    load_md <= '1';
    clr_pp <= '1';
    ELSEIF boostate MOD 2 = 0 THEN
      --boostate = 2,4,6,8 ....
      shift_mr <= '1';
  END IF;
END PROCESS;
```

Examples of VHDL Descriptions

```vhdl
shift_pp <= '1';
ELSE    --boostate = 1,3,5,7......
    IF mrreg(0) = mrreg(1) THEN
        NULL; --refresh pp
    ELSE
        load_pp <= '1'; --update product
    END IF;
    comp <= mrreg(1);       --subract if mrreg(1 DOWNTO 0) =$10*
END IF;
END PROCESS;
END structural;
```

A First-in First-out Memory

--a first-in first out memory, uses a synchronising clock
--generics allow fifos of different sizes to be instantiated
library IEEE;
use IEEE.Std_logic_1164.all;
entity FIFOMXN is
  generic(m, n : Positive := 8); --m is fifo depth, n is fifo width
  port(RESET, WRREQ, RDREQ, CLOCK : in Std_logic;
       DATAIN : in Std_logic_vector((n-1) downto 0);
       DATAOUT : out Std_logic_vector((n-1) downto 0);
       FULL, EMPTY : inout Std_logic);
end FIFOMXN;
architecture V2 of FIFOMXN is
  type Fifo_array is array(0 to (m-1)) of Bit_vector((n-1) downto 0);
  signal Fifo_memory : Fifo_array;
  signal Wraddr, Rdaddr, Offset : Natural range 0 to (m-1);
  signal Rdpulse, Wrpulse, Q1, Q2, Q3, Q4 : Std_logic;
  signal Databuffer : Bit_vector((n-1) downto 0);
begin
  --pulse synchronisers for WRREQ and RDREQ
  --modified for Synplify to a process
  sync_ffs : process
    begin
      wait until rising_edge(CLOCK);
      Q1 <= WRREQ;
      Q2 <= Q1;
      Q3 <= RDREQ;
      Q4 <= Q3;
    end process;
  --concurrent logic to generate pulses
  Wrpulse <= Q2 and not(Q1);
  Rdpulse <= Q4 and not(Q3);
  Fifo_read : process
    begin
      wait until rising_edge(CLOCK);
      if RESET = '1' then
        Rdaddr <= 0;
        Databuffer <= (others => '0');
      elsif (Rdpulse = '1' and EMPTY = '0') then
        Databuffer <= Fifo_memory(Rdaddr);
        Rdaddr <= (Rdaddr + 1) mod m;
      end if;
    end process;
  end process;
  Fifo_write : process
    begin
      wait until rising_edge(CLOCK);
      if RDREQ = '1' then
        Rdaddr <= 0;
        Databuffer <= (others => '0');
      else
        if Wrpulse = '1' and FULL = '0' then
          Databuffer <= Fifo_memory(Wraddr);
          Wraddr <= (Wraddr + 1) mod m;
        end if;
      end if;
    end process;
end F2;
```

Examples of VHDL Descriptions

begin
  wait until rising_edge(CLOCK);
  if RESET = '1' then
    Wraddr <= 0;
  elsif (Wrpulse = '1' and FULL = '0') then
    Fifo_memory(Wraddr) <= To_Bitvector(DATAIN);
    Wraddr <= (Wraddr + 1) mod m;
  end if;
end process;

Offset <= (Wraddr - Rdaddr) when (Wraddr > Rdaddr)
  else (m - (Rdaddr - Wraddr)) when (Rdaddr > Wraddr)
  else 0;

EMPTY <= '1' when (Offset = 0) else '0';
FULL <= '1' when (Offset = (m-1)) else '0';

DATAOUT <= To_Stdlogicvector(Databuffer) when RDREQ = '0'
  else (others => 'Z');
end V2;

ROM-based waveform generator

PACKAGE rompac IS
  CONSTANT rom_width : POSITIVE := 3;
  CONSTANT addr_high : POSITIVE := 12;
  SUBTYPE rom_word IS BIT_VECTOR(0 TO rom_width);
  TYPE rom_table IS ARRAY(0 TO addr_high) OF rom_word;
  CONSTANT rom : rom_table :=
    ('1100'
     , '1100'
     , '0100'
     , '0000'
     , '0110'
     , '0101'
     , '0111'
     , '1100'
     , '0100'
     , '0000'
     , '0110'
     , '0101'
     , '0111');
END rompac;

BEGIN
  OFFSET <= (Wraddr - Rdaddr) when (Wraddr > Rdaddr)
    else (m - (Rdaddr - Wraddr)) when (Rdaddr > Wraddr)
    else 0;

  EMPTY <= '1' when (OFFSET = 0) else '0';
  FULL <= '1' when (OFFSET = (m-1)) else '0';

  DATAOUT <= To_Stdlogicvector(Databuffer) when RDREQ = '0'
    else (others => 'Z');

--WAVEFORM GENERATOR USING A ROM LOOK-UP TABLE 15-6-92
--THE ROM IS A CONSTANT DECLARED WITHIN THE PACKAGE rompac.

USE work.rompac.ALL;

ENTITY romwaves IS
  PORT(clock : IN BIT; reset : IN BOOLEAN;
       waves : OUT rom_word);
END romwaves;

ARCHITECTURE behaviour OF romwaves IS

  SIGNAL step : NATURAL;

BEGIN
  --address counter for rom look-up table
  step_counter:PROCESS
  BEGIN

Examples of VHDL Descriptions

```
WAIT UNTIL clock'EVENT AND clock = '1';
IF reset THEN --check for reset condition
  step <= 0;
ELSIF step = addr_high THEN --check for last wave value
  step <= addr_high;
ELSE
  step <= step + 1; --get next wave value
END IF;
END PROCESS;

--output value from rom look-up table
waves <= rom(step);
END behaviour;
```

Classic 2-Process State Machine and Test Bench

```
--MEALY TYPE STATE MACHINE EXAMPLE
ENTITY fsm IS
  PORT(clock,x : IN BIT; z : OUT BIT);
END fsm;
-------------------------------------------------
ARCHITECTURE behaviour OF fsm IS
  TYPE state_type IS (s0,s1,s2,s3);
  SIGNAL present_state,next_state : state_type;
BEGIN
  --state register process
  state_reg:PROCESS
  BEGIN
    WAIT UNTIL clock'EVENT AND clock = '1';
    present_state <= next_state;
  END PROCESS;
  --combinational logic feedback process
  fb_logic:PROCESS(present_state,x)
  BEGIN
    CASE present_state IS
    WHEN s0 =>
      IF x = '0' THEN z <= '0'; next_state <= s0;
      ELSE z <= '1'; next_state <= s2;
      END IF;
    WHEN s1 =>
      IF x = '0' THEN z <= '0'; next_state <= s0;
      ELSE z <= '0'; next_state <= s2;
      END IF;
    WHEN s2 =>
      IF x = '0' THEN z <= '1'; next_state <= s2;
      ELSE z <= '0'; next_state <= s3;
      END IF;
    WHEN s3 =>
      IF x = '0' THEN z <= '0'; next_state <= s3;
      ELSE z <= '1'; next_state <= s1;
      END IF;
    END CASE;
  END PROCESS;
END behaviour;
-----------------------------------------------------------
--STIMULUS GENERATOR FOR FSM
ENTITY fsm_stim IS
  PORT:
  BEGIN
    Stimulus : PROCESS
    BEGIN
      WAIT UNTIL clock'EVENT AND clock = '1';
      Stimulus <= '0';
    END PROCESS;
  END PROCESS;
```

```
Examples of VHDL Descriptions

```vhdl
ARCHITECTURE behavioural OF fsm_stim IS
BEGIN
   --clock pulses : __--__--__--__--__--__
   --x input      : _____------------_____
   --each '-' represents 5 ns.
   clock <= '0' AFTER 0 ns,
            '1' AFTER 10 ns, --clock 1
            '0' AFTER 20 ns,
            '1' AFTER 30 ns, --clock 2
            '0' AFTER 40 ns,
            '1' AFTER 50 ns, --clock 3
            '0' AFTER 60 ns,
            '1' AFTER 70 ns, --clock 4
            '0' AFTER 80 ns,
            '1' AFTER 90 ns, --clock 5
            '0' AFTER 100 ns;
   x <= '0' AFTER 0 ns,
        '1' AFTER 25 ns,
        '0' AFTER 85 ns;
END behavioural;
```
State Machine with Asynchronous Reset

library ieee;
use ieee.std_logic_1164.all;

entity stmch1 is
    port(clk, in1, rst: in std_logic; out1: out std_logic);
end stmch1;

architecture behave of stmch1 is

    type state_values is (sx, s0, s1);
    signal state, next_state: state_values;

begin

    process (clk, rst)
    begin
        if rst = '1' then
            state <= s0;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;

    process (state, in1)
    begin
        -- set defaults for output and state
        out1 <= '0';
        next_state <= sx; -- catch missing assignments to next_state
        case state is
        when s0 =>
            if in1 = '0' then
                out1 <= '1';
                next_state <= s1;
            else
                out1 <= '0';
                next_state <= s0;
            end if;
        when s1 =>
            if in1 = '0' then
                out1 <= '0';
                next_state <= s0;
            end if;
    end case;
    end process;

    z <= '1';

    END using_wait;
END;
-----------------------------------------------------------
Examples of VHDL Descriptions

```
next_state <= s0;
else
out1 <= '1';
next_state <= s1;
end if;
when sx =>
next_state <= sx;
end case;
end process;
end behave;
```

Pattern Detector FSM with Test Bench

```
LIBRARY ieee;
USE ieee.Std_logic_1164.ALL;
ENTITY patdet IS
    PORT(clock, serin, reset : IN Std_logic; match : OUT Std_logic);
END patdet;
ARCHITECTURE v1 OF patdet IS
    TYPE state_type IS (s0, s1, s2, s3, s4, s5, s6, s7, s8);
    SIGNAL pstate, nstate : state_type;
BEGIN
    --state register
    PROCESS
    BEGIN
        WAIT UNTIL rising_edge(clock);
        IF reset = '1' THEN
            pstate <= s0;
        ELSE
            pstate <= nstate;
        END IF;
    END PROCESS;

    --next state logic
    PROCESS(serin, pstate)
    BEGIN
        CASE pstate IS
        WHEN s0 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s1;
        END IF;
        WHEN s1 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s2;
        END IF;
        WHEN s2 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s3;
        END IF;
        WHEN s3 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s4;
        END IF;
        WHEN s4 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s5;
        END IF;
        WHEN s5 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s6;
        END IF;
        WHEN s6 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s7;
        END IF;
        WHEN s7 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s8;
        END IF;
        WHEN s8 => IF serin = '0' THEN
            nstate <= s0;
        ELSE
            nstate <= s0;
        END IF;
        END CASE;
    END PROCESS;
```

Examples of VHDL Descriptions

END IF;
WHEN s5 =>  IF serin = '0' THEN
  nstate <= s0;
ELSE
  nstate <= s6;
END IF;
WHEN s6 =>  IF serin = '1' THEN
  nstate <= s8;
ELSE
  nstate <= s7;
END IF;
WHEN s7 =>  IF serin = '0' THEN
  nstate <= s0;
ELSE
  nstate <= s8;
END IF;
WHEN s8 =>  IF serin = '0' THEN
  nstate <= s0;
ELSE
  nstate <= s8;
END IF;
WHEN OTHERS => nstate <= s0;
END CASE;
END PROCESS;

--generate output
match <= '1' WHEN pstate = s7 ELSE '0';
END v1;

--The following Design Entity defines a parameterised Pseudo-random
--bit sequence generator, it is useful for generating serial or parallel test
--waveforms
--(for parallel waveforms you need to add an extra output port)
--the generic 'length' is the length of the register minus one.
--the generics 'tap1' and 'tap2' define the feedback taps
LIBRARY ieee;
USE ieee.Std_logic_1164.ALL;
ENTITY prbsgen IS
  GENERIC(length : Positive := 8; tap1 : Positive := 8; tap2 : Positive := 4);
  PORT(clk, reset : IN Std_logic; prbs : OUT Std_logic);
END prbsgen;
ARCHITECTURE v3 OF prbsgen IS
  --create a shift register
  SIGNAL prreg : Std_logic_vector(length DOWNTO 0);
BEGIN
  --conditional signal assignment shifts register and feeds in xor value
  prreg <= (0 => '1', OTHERS => '0') WHEN reset = '1' ELSE
    (prreg((length - 1) DOWNTO 0) & (prreg(tap1) XOR prreg(tap2)))
  WHEN rising_edge(clk) ELSE prreg;
  --connect msb of register to output
  prbs <= prreg(length);
END v3;
LIBRARY ieee;
USE ieee.Std_logic_1164.ALL;
ENTITY patdetbench IS
END patdetbench;
ARCHITECTURE precomp OF patdetbench IS
--defining architecture for pre-synthesis functional simulation
Examples of VHDL Descriptions

```
COMPONENT prbsgen
    PORT(clk, reset : IN Std_logic; prbs : OUT Std_logic);
END COMPONENT;

COMPONENT patdet
    PORT(clock, serin, reset : IN Std_logic; match : OUT Std_logic);
END COMPONENT;

--configure patdet to be functional model
FOR patdet1 : patdet USE ENTITY work.patdet(v1);

SIGNAL clock, reset, pattern, match : Std_logic;
BEGIN
    --clock generator
    PROCESS
    BEGIN
        clock <= '0', '1' AFTER 50 ns;
        WAIT FOR 100 ns;
    END PROCESS;

    patgen1 : prbsgen PORT MAP (clock, reset, pattern);
    patdet1 : patdet PORT MAP (clock, pattern, reset, match);
END precomp;
```

Chess Clock

```
PACKAGE chesspack IS
    SUBTYPE hours IS NATURAL;
    SUBTYPE minutes IS INTEGER RANGE 0 TO 60;
    SUBTYPE seconds IS INTEGER RANGE 0 TO 60;

    TYPE elapsed_time IS
        RECORD
            hh : hours;
            mm : minutes;
            ss : seconds;
        END RECORD;
    TYPE state IS (reset,hold,runb,runa);

    FUNCTION inctime (intime : elapsed_time) RETURN elapsed_time;
    FUNCTION zero_time RETURN elapsed_time;
END chesspack;

PACKAGE BODY chesspack IS

    FUNCTION inctime (intime : elapsed_time) RETURN elapsed_time IS
        VARIABLE result : elapsed_time;
        BEGIN
            result := intime;
            result.ss := result.ss + 1;
            IF result.ss = 60 THEN
                result.ss := 0;
                result.mm := result.mm + 1;
            IF result.mm = 60 THEN
                result.mm := 0;
                result.hh := result.hh + 1;
            END IF;
        END IF;
    RETURN result;
```
FUNCTION zero_time RETURN elapsed_time IS
VARIABLE result : elapsed_time;
BEGIN
result.ss := 0;
result.mm := 0;
result.hh := 0;
RETURN result;
END zero_time;

USE WORK.chesspack.ALL;
ENTITY timer IS
--time_used must be inout port since signal assignment statement
--reads it's value to compute the new value of time_used.
PORT(enable,clear,one_sec : IN BIT; time_used : INOUT elapsed_time);
END timer;
ARCHITECTURE behaviour OF timer IS
BEGIN
time_used <= zero_time WHEN clear = '1' ELSE
inctime(time_used) WHEN
(enable = '1' AND one_sec'EVENT AND one_sec = '1')
ELSE time_used;
END behaviour;

USE WORK.chesspack.ALL;
ENTITY chessclock IS
PORT(a,b,hold_time,reset_time : IN BIT;
time_a,time_b : INOUT elapsed_time);
END chessclock;
ARCHITECTURE structure OF chessclock IS
COMPONENT timer
PORT(enable,clear,one_sec : IN BIT; time_used : INOUT elapsed_time);
END COMPONENT;
SIGNAL one_sec,clock,ena,enb,clear_timers : BIT := '0';
BEGIN
--instantiating timers a and b
timer_a : timer PORT MAP(ena,clear_timers,one_sec,time_a);
timer_b : timer PORT MAP(enb,clear_timers,one_sec,time_b);
controller:BLOCK     --chessclock state machine
SIGNAL present_state,next_state : state := reset;
BEGIN
--state register
state_reg:BLOCK
BEGIN
PROCESS(clock)
BEGIN
IF (clock'EVENT AND clock = '1' AND clock'LAST_VALUE = '0')
THEN present_state <= next_state;
END IF;
END PROCESS;
END BLOCK state_reg;
Examples of VHDL Descriptions

```vhdl
-- output and feedback logic
logic: BLOCK
BEGIN
  PROCESS(a, b, hold_time, reset_time, present_state)
  VARIABLE a_b : BIT_VECTOR(0 TO 1);
  BEGIN
    a_b := a & b;  -- aggregate assignment for case statement
    CASE present_state IS
      WHEN reset =>
        clear_timers <= '1';
        ena <= '0';
        enb <= '0';
        IF reset_time = '1' THEN next_state <= reset;
        ELSE CASE a_b IS
          WHEN "00" => next_state <= hold;
          WHEN "01" => next_state <= runa;
          WHEN "10" => next_state <= runb;
          WHEN "11" => next_state <= hold;
          END CASE;
      END IF;
      WHEN hold =>
        clear_timers <= '0';
        ena <= '0';
        enb <= '0';
        IF reset_time = '1' THEN next_state <= reset;
        ELSIF hold_time = '1' THEN next_state <= hold;
        ELSE CASE a_b IS
          WHEN "00" => next_state <= hold;
          WHEN "01" => next_state <= runa;
          WHEN "10" => next_state <= runb;
          WHEN "11" => next_state <= hold;
          END CASE;
        END IF;
      WHEN runa =>
        clear_timers <= '0';
        ena <= '1';
        enb <= '0';
        IF reset_time = '1' THEN next_state <= reset;
        ELSIF hold_time = '1' THEN next_state <= hold;
        ELSIF a = '0' THEN next_state <= runa;
        ELSIF b = '1' THEN next_state <= hold;
        ELSE next_state <= runb;
        END IF;
      WHEN runb =>
        clear_timers <= '0';
        ena <= '0';
        enb <= '1';
        IF reset_time = '1' THEN next_state <= reset;
        ELSIF hold_time = '1' THEN next_state <= hold;
        ELSIF a = '0' THEN next_state <= runa;
        ELSIF b = '1' THEN next_state <= runb;
        ELSE next_state <= runb;
        END IF;
    END CASE;
  END PROCESS;
END BLOCK logic;

one_sec_clock: BLOCK
BEGIN
  PROCESS -- process to generate one second clock
  BEGIN
    one_sec <= TRANSPORT '1' AFTER 500 ms;
  END PROCESS;
END BLOCK one_sec_clock;
```

Examples of VHDL Descriptions

one_sec <= TRANSPORT '0' AFTER 1000 ms;
WAIT FOR 1000 ms;
END PROCESS;
END BLOCK one_sec_clock;

system_clock:BLOCK
BEGIN
PROCESS --process to generate 10Hz state machine clock
BEGIN
  clock <= TRANSPORT '1' AFTER 50 ms;
  clock <= TRANSPORT '0' AFTER 100 ms;
  WAIT FOR 100 ms;
END PROCESS;
END BLOCK system_clock;
END structure;

Digital Delay Unit

- Package defining types used by the system memory
- Package defining a basic analogue type
- 16-bit Analogue to Digital Converter
- 16-bit Digital to Analogue Converter
- Top-level Digital Delay Unit including RAM and control process
- Sinewave generator for testbench
- Testbench for Digital Delay Unit

Package defining types used by the system memory

PACKAGE rampac IS
  SUBTYPE addr10 IS NATURAL RANGE 0 TO 1023;
  SUBTYPE data16 IS INTEGER RANGE -32768 TO +32767;
  TYPE ram_array IS ARRAY(addr10'LOW TO addr10'HIGH) OF data16;
  CONSTANT z_val : data16 := -1;
END rampac;

Package defining a basic analogue type

PACKAGE adcpac IS
  SUBTYPE analogue IS REAL RANGE -5.0 TO +5.0;
END adcpac;

16-bit Analogue to Digital Converter

USE WORK.rampac.ALL;
USE WORK.adcpac.ALL;
ENTITY adc16 IS
  GENERIC(tconv : TIME := 10 us); --conversion time
  PORT(vin : IN analogue; digout : OUT data16; --input and output
dc : IN BIT; busy : OUT BIT); --control
END adc16;
ARCHITECTURE behaviour OF adc16 IS
INTERNAL Description

BEGIN
   PROCESS
   VARIABLE digtemp : data16;
   CONSTANT vlsb : analogue := (analogue'HIGH - analogue'LOW)/REAL(2*ABS(data16'LOW));
   BEGIN
      digtemp := data16'LOW;
      busy <= '0';
      WAIT UNTIL (sc'EVENT AND sc = '0');
      busy <= '1';
      FOR i IN 0 TO (2*data16'HIGH) LOOP
         IF vin >= (analogue'LOW + (REAL(i) + 0.5)*vlsb) THEN digtemp := digtemp + 1;
         ELSE EXIT;
      END IF;
      END LOOP;
      WAIT FOR tconv;
      digout <= digtemp;
      busy <= '0';
   END PROCESS;
END behaviour;

16-bit Digital to Analogue Converter

USE WORK.rampac.ALL;
USE WORK.adcpac.ALL;
ENTITY dac16 IS
   PORT(vout : INOUT analogue; digin : IN data16; --input and output
         en : IN BIT); --latches in data
END dac16;
ARCHITECTURE behaviour OF dac16 IS
   CONSTANT vlsb : analogue := (analogue'HIGH - analogue'LOW)/REAL(2*ABS(data16'LOW));
   BEGIN
      --store analogue equivalent of digin on vout when negative edge on en
      vout <= REAL(digin)*vlsb WHEN (en'EVENT AND en = '0') ELSE vout;
   END behaviour;

Top-level Digital Delay Unit including RAM and control process

--VHDL model of a ram-based analogue delay system.

USE WORK.rampac.ALL;
USE WORK.adcpac.ALL;
ENTITY digdel2 IS
   PORT(clear : IN BIT;          --clears address counter
         offset : IN addr10;     --delay control
         sigin : IN analogue;    --signal input
         sigout : INOUT analogue);   --signal output
END digdel2;
ARCHITECTURE block_struct OF digdel2 IS
   COMPONENT adc16
      PORT(vin : IN analogue; digout : OUT data16;
           sc : IN BIT; busy : OUT BIT);
      END COMPONENT;
   COMPONENT dac16
      PORT(vout : INOUT analogue; digin : IN data16;
           en : IN BIT);
      END COMPONENT;
   SIGNAL address : addr10;                 --pointer to ram location
   BEGIN
      --
   END block_struct;
Examples of VHDL Descriptions

```vhdl
SIGNAL ram_data_out : data16;            --data output of ram
SIGNAL ram_data_in : data16;             --data input to ram
SIGNAL clock, cs, write, suboff, dacen, adcbusy : BIT;   --internal controls

BEGIN
--start conversion on positive edge of 'clock'at beginning of cycle
adcsc <= NOT clock; --|__________----------|

adc1 : adc16 PORT MAP (sigin, ram_data_in, adcsc, adcbusy);

cs <= '1'; --enable ram device

ram:BLOCK -- 16-bit * 1024 location RAM
BEGIN

ram_proc:PROCESS(cs, write, address, ram_data_in)
VARIABLE ram_data : ram_array;
VARIABLE ram_init : BOOLEAN := FALSE;
BEGIN
IF NOT(ram_init) THEN  --initialise ram locations
FOR i IN ram_data'RANGE LOOP
  ram_data(i) := 0;
END LOOP;
ram_init := TRUE;
END IF;
IF cs = '1' THEN
  IF write = '1' THEN
    ram_data(address) := ram_data_in;
  END IF;
  ram_data_out <= ram_data(address);
ELSE
  ram_data_out <= z_val;
END IF;
END PROCESS;
END BLOCK ram;

dac1 : dac16 PORT MAP (sigout, ram_data_out, dacen);

-- concurrent statement for 'suboff' (subtract offset) signal for counter
suboff <= clock; --|--------------|

suboff <= clock; --|----------__________|

suboff <= clock; --|-----_______________|

SUB <= TRANSPORT '1' AFTER 13 us, --|_____________----___|
SUB <= TRANSPORT '0' AFTER 17 us;
```

**Examples of VHDL Descriptions**

```vhdl
WAIT FOR 20 us;
END PROCESS control_waves;
END block_struct;
```

---

**Sinewave generator for testbench**

```vhdl
--entity to generate a 2.5kHz sampled sinewave (sampled at 20 us intervals)
USE WORK.adcpac.ALL;
ENTITY sinegen IS
    PORT(sinewave : OUT analogue);
END sinegen;
ARCHITECTURE behaviour OF sinegen IS
    CONSTANT ts : TIME := 20 us;  --sample interval
    TYPE sinevals IS ARRAY (0 TO 5) OF analogue;
    --sample values for one quarter period
    CONSTANT qrtrsine : sinevals := (0.0, 1.545, 2.939, 4.045, 4.755, 5.0);
    BEGIN
        PROCESS   --sequential process generates sinewave
            FOR i IN 0 TO 19 LOOP --output 20 samples per period
                IF (i >= 0) AND (i < 6) THEN   --first quarter period
                    sinewave <= qrtrsine(i);
                ELSIF (i >= 6) AND (i < 11) THEN  --second quarter period
                    sinewave <= qrtrsine(10-i);
                ELSIF (i >= 11) AND (i < 16) THEN   --third quarter period
                    sinewave <= -qrtrsine(i-10);
                ELSE   --i IN 16 TO 19
                    sinewave <= -qrtrsine(20-i);  --final quarter period
                END IF;
                WAIT FOR ts;
            END LOOP;
        END PROCESS;
    END behaviour;
```

---

**Testbench for Digital Delay Unit**

```vhdl
USE WORK.rampac.ALL;
USE WORK.adcpac.ALL;
ENTITY delay_bench IS
    PORT(reset : IN BIT; delay : IN addr10);
END delay_bench;
ARCHITECTURE version1 OF delay_bench IS
    COMPONENT sinegen
        PORT(sinewave : OUT analogue);
    END COMPONENT;
    COMPONENT digdel2
        PORT(clear : IN BIT; offset : IN addr10;
            sigin : IN analogue; sigout : INOUT analogue);
    END COMPONENT;
    SIGNAL analogue_in, analogue_out : analogue;
    BEGIN
        sig_gen : sinegen PORT MAP(analogue_in);
        delay_unit : digdel2 PORT MAP(reset, delay, analogue_in, analogue_out);
    END ARCHITECTURE version1;
```

---

Examples of VHDL Descriptions

8-bit Analogue to Digital Converter

-- 8-bit analogue to digital converter
-- demonstrates use of LOOP and WAIT statements

ENTITY adc8 IS
  GENERIC(tconv : TIME := 10 us); -- conversion time
  PORT(vin : IN REAL RANGE 0.0 TO +5.0; -- unipolar input
  digout : OUT NATURAL RANGE 0 TO 255; -- output
  sc : IN BIT; busy : OUT BIT); -- control
END adc8;

ARCHITECTURE behaviour OF adc8 IS
BEGIN
  PROCESS
    VARIABLE digtemp : NATURAL;
    CONSTANT vlsb : REAL := 5.0/256; -- least significant bit value
  BEGIN
    digtemp := 0;
    WAIT UNTIL (sc'EVENT AND sc = '0'); -- falling edge on sc starts conv
    busy <= '1'; -- flag converter busy
    WAIT FOR tconv; -- conversion time
    FOR i IN 0 TO 255 LOOP -- do ramp-up conversion
      IF vin >= REAL(i)*vlsb
      THEN IF digtemp = 255 THEN EXIT;
      ELSE digtemp := digtemp + 1;
      END IF;
      ELSE EXIT;
      END IF;
    END LOOP;
    digout <= digtemp; -- output result
    busy <= '0'; -- flag end of conversion
  END PROCESS;
END behaviour;

8-bit Unipolar Successive Approximation ADC

-- 8-bit unipolar successive approximation analogue to digital converter
-- demonstrates use of LOOP and WAIT statements

ENTITY adcsc8 IS
  PORT(vin : IN REAL RANGE 0.0 TO +5.0; -- unipolar input
  digout : OUT BIT_VECTOR(7 DOWNTO 0); -- digital output
  clock, sc : IN BIT; busy : OUT BIT); -- clock & control
END adcsc8;

ARCHITECTURE behaviour OF adcsc8 IS
BEGIN
  VARIABLE v_estimate : REAL RANGE 0.0 TO +5.0;
  PROCESS
    BEGIN
      WAIT UNTIL (sc'EVENT AND sc = '0'); -- falling edge on sc starts conv
      busy <= '1'; -- flag converter busy
      WAIT FOR tconv; -- conversion time
      FOR i IN 0 TO 255 LOOP -- do ramp-up conversion
        IF vin >= REAL(i)*vlsb
        THEN IF digtemp = 255 THEN EXIT;
        ELSE digtemp := digtemp + 1;
        END IF;
        ELSE EXIT;
        END IF;
      END LOOP;
      digout <= digtemp; -- output result
      busy <= '0'; -- flag end of conversion
    END PROCESS;
END behaviour;
BEGIN
   PROCESS
      CONSTANT v_lsb : REAL := 5.0/256;  --least significant bit value
   BEGIN
      WAIT UNTIL (sc'EVENT AND sc = '0'); --falling edge on sc starts conv
      v_estimate <= 0.0;                 --initialise v_estimate
      digout <= "00000000";             --clear SAR register
      busy <= '1';                        --flag converter busy
      FOR i IN digout'RANGE LOOP  --loop for each output bit
         WAIT UNTIL (clock'EVENT AND clock = '1');
         v_estimate <= v_estimate + (REAL(2**i))*v_lsb;
         digout(i) <= '1';
         WAIT UNTIL (clock'EVENT AND clock = '1');
         IF v_estimate >= vin THEN
            v_estimate <= v_estimate - (REAL(2**i))*v_lsb;
            digout(i) <= '0';
         END IF;
      END LOOP;
      busy <= '0';          --flag end of conversion
   END PROCESS;
END behaviour;

TTL164 Shift Register

ENTITY dev164 IS
   PORT(a, b, nclr, clock : IN BIT;
   q : BUFFER BIT_VECTOR(0 TO 7));
END dev164;
ARCHITECTURE version1 OF dev164 IS
BEGIN
   PROCESS(a, b, nclr, clock)
   BEGIN
      IF nclr  = '0' THEN
         q <= "00000000";
      ELSE
         IF clock'EVENT AND clock = '1'
            THEN
               FOR i IN q'RANGE LOOP
                  IF i = 0 THEN q(i) <= (a AND b);
                  ELSE
                     q(i) <= q(i-1);
                  END IF;
               END LOOP;
            END IF;
      END IF;
   END PROCESS;
END version1;
Examples of VHDL Descriptions

--8-bit universal shift register modelled using a process
ENTITY shftreg8 IS
 PORT(clock, serinl, serinr : IN BIT; --clock and serial inputs
      mode : IN BIT_VECTOR(0 TO 1); --"00" : disabled; "10" : shift left; "01" : shift right; "11" : Parallel load;
      parin : IN BIT_VECTOR(0 TO 7); --parallel inputs
      parout : OUT BIT_VECTOR(0 TO 7)); --parallel outputs
END shftreg8;
ARCHITECTURE behavioural OF shftreg8 IS
BEGIN
 PROCESS
      --declare variable to hold register state
      VARIABLE state : BIT_VECTOR(0 TO 7) := "00000000";
      BEGIN
      --synchronise process to rising edges of clock
      WAIT UNTIL clock'EVENT AND clock = '1';
      CASE mode IS
      WHEN "00" =>   state := state; --disabled
            WHEN "10" =>
            FOR i IN 0 TO 7 LOOP    --shift
            IF i = 7 THEN
            state(i) := serinl;
            ELSE
            state(i) := state(i + 1);
            END IF;
            END LOOP;
      WHEN "01" =>
            FOR i IN 7 DOWNTO 0 LOOP  --shift
            IF i = 0 THEN
            state(i) := serinr;
            ELSE
            state(i) := state(i - 1);
            END IF;
            END LOOP;
      WHEN "11" =>   state := parin;   --parallel load
            END CASE;
      --assign variable to parallel output port
      parout <= state;
      END PROCESS;
END behavioural;

Structural Description of an 8-bit Shift Register

ENTITY dtff IS
 GENERIC(initial : BIT := '1');  --initial value of q
 PORT(d, clock : IN BIT; q : BUFFER BIT := initial);
END dtff;
ARCHITECTURE zero_delay OF dtff IS
BEGIN
      q <= d WHEN (clock'EVENT AND clock = '1');
END zero_delay;

--Structural model of an 8-bit universal shift register
--makes use of D-type flip flop component and generate statement
ENTITY shftreg8 IS
 PORT(clock, serinl, serinr : IN BIT; mode : IN BITVECTOR(0 TO 1);
      parin : IN BIT_VECTOR(0 TO 7));
END shftreg8;
ARCHITECTURE structural OF shftreg8 IS
BEGIN
 PROCESS
      --declare variable to hold register state
      VARIABLE state : BIT_VECTOR(0 TO 7) := "00000000";
      BEGIN
      --synchronise process to rising edges of clock
      WAIT UNTIL clock'EVENT AND clock = '1';
      CASE mode IS
      WHEN "00" =>   state := state; --disabled
            WHEN "10" =>
            FOR i IN 0 TO 7 LOOP    --shift
            IF i = 7 THEN
            state(i) := serinl;
            ELSE
            state(i) := state(i + 1);
            END IF;
            END LOOP;
      WHEN "01" =>
            FOR i IN 7 DOWNTO 0 LOOP  --shift
            IF i = 0 THEN
            state(i) := serinr;
            ELSE
            state(i) := state(i - 1);
            END IF;
            END LOOP;
      WHEN "11" =>   state := parin;   --parallel load
            END CASE;
      --assign variable to parallel output port
      parout <= state;
      END PROCESS;
END structural;
Examples of VHDL Descriptions

parout : BUFFER BIT_VECTOR(0 TO 7));
END shftreg8;

ARCHITECTURE structural OF shftreg8 IS

COMPONENT dtff
    GENERIC(initial : BIT := '1');
    PORT(d, clock : IN BIT; q : BUFFER BIT := initial);
END COMPONENT;

FOR ALL : dtff USE ENTITY work.dtff(zero_delay);
SIGNAL datain : BIT_VECTOR(0 TO 7);

BEGIN

reg_cells : FOR i IN 0 TO 7
    GENERATE
        reg_stage : dtff GENERIC MAP ('0') PORT MAP (datain(i) , clock, parout(i));
        lsb_stage : IF i = 0 GENERATE
            datain(i) <= parin(i) WHEN mode = "00" ELSE serinl WHEN mode = "10"
            ELSE parout(i + 1) WHEN mode = "01" ELSE parout(i);
        END GENERATE;
        msb_stage : IF i = 7 GENERATE
            datain(i) <= parin(i) WHEN mode = "00" ELSE parout(i - 1) WHEN mode = "10"
            ELSE serinr WHEN mode = "01" ELSE parout(i);
        END GENERATE;
        middle_stages : IF (i > 0) AND (i < 7) GENERATE
            datain(i) <= parin(i) WHEN mode = "00" ELSE parout(i - 1) WHEN mode = "10"
            ELSE parout(i + 1) WHEN mode = "01" ELSE parout(i);
        END GENERATE;
    END GENERATE;
END GENERATE;

END structural;

8-bit Unsigned Multiplier

library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Std_logic_unsigned.all;
entity MUL8X8 is
    port(A, B : in Std_logic_vector(7 downto 0);
        PROD : out Std_logic_vector(15 downto 0));
end MUL8X8;
architecture SYNP of MUL8X8 is
begin
    PROD <= A * B;
end SYNP;

n-bit Adder using the Generate Statement

ENTITY addn IS
    GENERIC(n : POSITIVE := 3);   --no. of bits less one
    PORT(addend, augend : IN BIT_VECTOR(0 TO n);
        carry_in : IN BIT; carry_out, overflow : OUT BIT;
        sum : OUT BIT_VECTOR(0 TO n));
END addn;

A Variety of Adder Styles

--- Single-bit adder

library IEEE;
use IEEE.std_logic_1164.all;
entity adder is
  port (a : in std_logic;
         b : in std_logic;
         cin : in std_logic;
         sum : out std_logic;
         cout : out std_logic);
end adder;

architecture rtl of adder is
begin
  sum <= (a xor b) xor cin;
  cout <= (a and b) or (cin and a) or (cin and b);
end rtl;

--- description of adder using concurrent signal assignments
architecture structural of adder is
begin
  xor1: xorg port map(
    in1 => a,
    in2 => b,
    out1 => xor1_out);
  xor2: xorg port map(
    in1 => xor1_out,
    in2 => cin,
    out1 => sum);
  and1: andg port map(
    in1 => xor1_out,
    in2 => cin,
    out1 => sum);
end structural;

-- Miscellaneous Logic Gates
use work.gates.all;
architecture structural of adder is
begin
  xor1: xorg port map(
    in1 => a,
    in2 => b,
    out1 => xor1_out);
  xor2: xorg port map(
    in1 => xor1_out,
    in2 => cin,
    out1 => sum);
  and1: andg port map(
    in1 => xor1_out,
    in2 => cin,
    out1 => sum);
end structural;
Examples of VHDL Descriptions

```vhdl
-- N-bit adder
-- The width of the adder is determined by generic N
library IEEE;
use IEEE.std_logic_1164.all;

entity adderN is
  generic (N : integer := 16);
  port (a : in std_logic_vector(N downto 1);
        b : in std_logic_vector(N downto 1);
        cin : in std_logic;
        sum : out std_logic_vector(N downto 1);
        cout : out std_logic);
end adderN;

architecture structural of adderN is
  component adder
    port (a : in std_logic;
         b : in std_logic;
         cin : in std_logic;
         sum : out std_logic;
         cout : out std_logic);
  end component;
  signal carry : std_logic_vector(0 to N);
begin
  carry(0) <= cin;
  cout <= carry(N);

  -- instantiate a single-bit adder N times
  gen: for I in 1 to N generate
    add: adder port map(
      a => a(I),
      b => b(I),
      cin => carry(I - 1),
      sum => sum(I),
      cout => carry(I));
  end generate;
end structural;

```

Examples of VHDL Descriptions

for i in 1 to N loop
    vsum(i) := (a(i) xor b(i)) xor carry;
    carry := (a(i) and b(i)) or (carry and (a(i) or b(i)));
end loop;

sum <= vsum;
cout <= carry;
end process p1;
end behavioral;

n-Bit Synchronous Counter

LIBRARY ieee;
USE ieee.Std_logic_1164.ALL;
USE ieee.Std_logic_unsigned.ALL;
ENTITY cntrnbit IS
    GENERIC(n : Positive := 8);
    PORT(clock, reset, enable : IN Std_logic;
         count : OUT Std_logic_vector((n-1) DOWNTO 0));
END cntrnbit;
ARCHITECTURE v1 OF cntrnbit IS
    SIGNAL count_int : Std_logic_vector((n-1) DOWNTO 0);
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL rising_edge(clock);
        IF reset = '1' THEN
            count_int <= (OTHERS => '0');
        ELSIF enable = '1' THEN
            count_int <= count_int + 1;
        ELSE
            NULL;
        END IF;
    END PROCESS;
    count <= count_int;
END v1;

Moore State Machine with Concurrent Output Logic

library ieee;
use ieee.std_logic_1164.all;

dentity moore1 is port(
    clk, rst:       in std_logic;
    id:             in std_logic_vector(3 downto 0);
    y:              out std_logic_vector(1 downto 0));
end moore1;
architecture archmoore1 of moore1 is
    type states is (state0, state1, state2, state3, state4);
    signal state: states;
BEGIN
    moore: process (clk, rst) --this process defines the next state only
    BEGIN
        if rst='1' then
            state <= state0;
        elsif (clk'event and clk='1') then
            case state is
                when state0 =>
                    if id = x"3" then
                        state <= state1;
                    else
                        state <= state0;
                    end if;
            end case;
        END IF;
    END PROCESS;
    signal state: states;
BEGIN
    moore: process (clk, rst) --this process defines the next state only
    BEGIN
        if rst='1' then
            state <= state0;
        elsif (clk'event and clk='1') then
            case state is
                when state0 =>
                    if id = x"3" then
                        state <= state1;
                    else
                        state <= state0;
                    end if;
        END IF;
    END PROCESS;
    count <= count_int;
END v1;
 Examples of VHDL Descriptions

Mealy State Machine with Registered Outputs

library ieee;
use ieee.std_logic_1164.all;

entity mealy1 is port(
clk, rst:       in std_logic;
id:             in std_logic_vector(3 downto 0);
y:              out std_logic_vector(1 downto 0));
end mealy1;

architecture archmealy of mealy1 is

begin

moore: process (clk, rst)
begin
if rst='1' then
state <= state0;
y <= "00";
elsif (clk'event and clk='1') then
case state is
when state0 =>
if id = x'3' then
state <= state0;
y <= "00";
elsif (clk'event and clk='1') then
when state1 =>
if id = x'5' then
state <= state0;
y <= "00";
elsif (clk'event and clk='1') then
when state2 =>
if id = x'7' then
state <= state3;
elsif id = x'9' then
state <= state4;
else
state <= state2;
end if;
when state3 =>
if id = x'b' then
state <= state0;
else
state <= state3;
end if;
when state4 =>
if id = x'F' then
state <= state0;
else
state <= state4;
end if;
end case;
end if;
end process;

--assign state outputs concurrently;
y <= "00" when (state=state0) else
"10" when (state=state1 or state=state3) else
"11";
end archmealy;
Examples of VHDL Descriptions

y <= "11";
when state2 =>
  if id = x"7" then
    state <= state3;
    y <= "10";
  else
    state <= state2;
    y <= "11";
  end if;
when state3 =>
  if id < x"7" then
    state <= state0;
    y <= "00";
  elsif id = x"9" then
    state <= state4;
    y <= "11";
  else
    state <= state3;
    y <= "10";
  end if;
when state4 =>
  if id = x"b" then
    state <= state0;
    y <= "00";
  else
    state <= state4;
    y <= "11";
  end if;
end case;
end if;
end process;
end archmealy;

Moore State Machine with explicit state encoding

library ieee;
use ieee.std_logic_1164.all;

entity moore2 is port(
  clk, rst:       in std_logic;
  id:             in std_logic_vector(3 downto 0);
  y:              out std_logic_vector(1 downto 0));
end moore2;

architecture archmoore2 of moore2 is

  signal state: std_logic_vector(2 downto 0);
  -- State assignment is such that 2 LSBs are outputs
  constant state0: std_logic_vector(2 downto 0) := "000";
  constant state1: std_logic_vector(2 downto 0) := "010";
  constant state2: std_logic_vector(2 downto 0) := "011";
  constant state3: std_logic_vector(2 downto 0) := "110";
  constant state4: std_logic_vector(2 downto 0) := "111";

begin
  moore: process (clk, rst)
  begin
    if rst='1' then
      state <= state0;
    elsif (clk'event and clk='1') then
      case state is
        when state0 =>
          if id = x"3" then
            state <= state1;
          else
            state <= state0;
          end if;
        when state1 =>
          if id = x"4" then
            state <= state2;
          else
            state <= state1;
          end if;
        when state2 =>
          if id = x"2" then
            state <= state3;
          else
            state <= state2;
          end if;
        when state3 =>
          if id = x"7" then
            state <= state0;
          else
            state <= state3;
          end if;
        when state4 =>
          if id = x"1" then
            state <= state5;
          else
            state <= state4;
          end if;
      end case;
    end if;
  end process;
end archmoore2;
--assign state outputs (equal to state std_logics)
y <= state(1 downto 0);
end archmoore2;

State Machine with Moore and Mealy outputs

library ieee;
use ieee.std_logic_1164.all;

entity mealy1 is port(
clk, rst: in std_logic;
id: in std_logic_vector(3 downto 0);
w: out std_logic;
y: out std_logic_vector(1 downto 0));
end mealy1;

architecture archmealy1 of mealy1 is
begin
moore: process (clk, rst)
begin
if rst='1' then
state <= state0;
elsif (clk'event and clk='1') then
case state is
when state0 =>
if id = x"3" then
state <= state1;
else
state <= state0;
end if;
when state1 =>
state <= state2;
when state2 =>
if id = x"7" then
state <= state3;
else
state <= state2;
end if;
when state3 =>
if id < x"7" then
state <= state0;
elsif id = x"9" then
state <= state4;
else
state <= state3;
end if;
when state4 =>
if id = x"b" then
state <= state0;
else
state <= state4;
end if;
when others =>
state <= state0;
end case;
end if;
end process;

--assign state outputs (equal to state std_logics)
y <= state(1 downto 0);
end archmealy1;
Examples of VHDL Descriptions

```
else
  state <= state2;
end if;
when state3 =>
  if id < x"7" then
    state <= state0;
  elsif id = x"9" then
    state <= state4;
  else
    state <= state3;
  end if;
when state4 =>
  if id = x"b" then
    state <= state0;
  else
    state <= state4;
  end if;
end case;
end if;
end process;
```

```
--assign moore state outputs;
y <= "00" when (state=state0) else
   "10" when (state=state1 or state=state3) else
   "11";
--assign mealy output;
w <= '0' when (state=state3 and id < x"7") else
   '1';
end archmealy1;
```

Muliplexer 16-to-4 using if-then-elsif-else Statement

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is port(
a, b, c, d:     in std_logic_vector(3 downto 0);
s:              in std_logic_vector(1 downto 0);
x:              out std_logic_vector(3 downto 0));
end mux;

architecture archmux of mux is
begin

  mux4_1: process (a, b, c, d)
  begin
    if s = "00" then
      x <= a;
    elsif s = "01" then
      x <= b;
    elsif s = "10" then
      x <= c;
    else
      x <= d;
    end if;
  end process mux4_1;
end archmux;
```

Muliplexer 16-to-4 using Conditional Signal Assignment Statement

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is port(
a, b, c, d:     in std_logic_vector(3 downto 0);
end mux;
```
Examples of VHDL Descriptions

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mux is port(
a, b, c, d: in std_logic_vector(3 downto 0);
s: in std_logic_vector(1 downto 0);
x: out std_logic_vector(3 downto 0));
end mux;
architecture archmux of mux is
begin
    x <= a when (s = "00") else
         b when (s = "01") else
         c when (s = "10") else
         d;
end archmux;
```

### Multiplexer 16-to-4 using Selected Signal Assignment Statement

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity mux is port(
a, b, c, d: in std_logic_vector(3 downto 0);
s: in std_logic_vector(1 downto 0);
x: out std_logic_vector(3 downto 0));
end mux;
architecture archmux of mux is
begin
    with s select
    x <= a when "00",
       b when "01",
       c when "10",
       d when "11",
       (others => 'X') when others;
end archmux;
```

### Miscellaneous Logic Gates

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
package gates is
    component andg
    generic (tpd_hl : time := 1 ns;
             tpd_lh : time := 1 ns);
    port (in1, in2 : std_ulogic;
          out1 : out std_ulogic);
    end component;

    component org
    generic (tpd_hl : time := 1 ns;
             tpd_lh : time := 1 ns);
    port (in1, in2 : std_logic;
          out1 : out std_logic);
    end component;

    component xorg
    generic (tpd_hl : time := 1 ns;
             tpd_lh : time := 1 ns);
    port (in1, in2 : std_logic;
          out1 : out std_logic);
    end component;
end gates;
```
Examples of VHDL Descriptions

-- and gate
------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity andg is
    generic (tpd_hl : time := 1 ns;
             tpd_lh : time := 1 ns);
    port (in1, in2 : std_logic;
          out1 : out std_logic);
end andg;
architecture only of andg is
begin
    p1: process(in1, in2)
    variable val : std_logic;
    begin
        val := in1 and in2;
        case val is
            when '0' =>
                out1 <= '0' after tpd_hl;
            when '1' =>
                out1 <= '1' after tpd_lh;
            when others =>
                out1 <= val;
        end case;
    end process;
end only;

-- or gate
------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity org is
    generic (tpd_hl : time := 1 ns;
             tpd_lh : time := 1 ns);
    port (in1, in2 : std_logic;
          out1 : out std_logic);
end org;
architecture only of org is
begin
    p1: process(in1, in2)
    variable val : std_logic;
    begin
        val := in1 or in2;
        case val is
            when '0' =>
                out1 <= '0' after tpd_hl;
            when '1' =>
                out1 <= '1' after tpd_lh;
            when others =>
                out1 <= val;
        end case;
    end process;
end only;

-- exclusive or gate
------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity xorg is
    generic (tpd_hl : time := 1 ns;
             tpd_lh : time := 1 ns);
    port (in1, in2 : std_logic;
          out1 : out std_logic);
end xorg;
architecture only of xorg is
begin
    p1: process(in1, in2)
    variable val : std_logic;
    begin
        val := in1 or in2;
        case val is
            when '0' =>
                out1 <= '0' after tpd_hl;
            when '1' =>
                out1 <= '1' after tpd_lh;
            when others =>
                out1 <= val;
        end case;
    end process;
end only;
Examples of VHDL Descriptions

```
end xorg;
architecture only of xorg is
begin
  p1: process(in1, in2)
  begin
    variable val : std_logic;
    begin
      val := in1 xor in2;
      case val is
      when '0' =>
        out1 <= '0' after tpd_hl;
      when '1' =>
        out1 <= '1' after tpd_lh;
      when others =>
        out1 <= val;
      end case;
    end process;
end only;
```

```
M68008 Address Decoder

--Address decoder for the m68008
--asbar must be '0' to enable any output
--csbar(0) : X'00000' to X'01FFF'
--csbar(1) : X'40000' to X'43FFF'
--csbar(2) : X'08000' to X'0AFFF'
--csbar(3) : X'E0000' to X'E01FF'
library ieee;
use ieee.std_logic_1164.all;
entity addrdec is
  port(
    asbar : in std_logic;
    address : in std_logic_vector(19 downto 0);
    csbar : out std_logic_vector(3 downto 0)
  );
end entity addrdec;
architecture v1 of addrdec is
begin
  csbar(0) <= '0' when
    ((asbar  = '0') and
    ((address >=  X'00000') and (address <=  X'01FFF'))
  else '1';
  csbar(1) <= '0' when
    ((asbar  = '0') and
    ((address >=  X'40000') and (address <=  X'43FFF'))
  else '1';
  csbar(2) <= '0' when
    ((asbar  = '0') and
    ((address >=  X'08000') and (address <=  X'0AFFF'))
  else '1';
  csbar(3) <= '0' when
    ((asbar  = '0') and
    ((address >=  X'E0000') and (address <=  X'E01FF'))
  else '1';
end architecture v1;
```

Highest Priority Encoder

```
```
Examples of VHDL Descriptions

entity priority is
  port(I : in bit_vector(7 downto 0); --inputs to be prioritised
       A : out bit_vector(2 downto 0); --encoded output
       GS : out bit);  --group signal output
end priority;

architecture v1 of priority is
begin
  process(I)
  begin
    GS <= '1'; --set default outputs
    A <= "000";
    if I(7) = '1' then
      A <= "111";
    elsif I(6) = '1' then
      A <= "110";
    elsif I(5) = '1' then
      A <= "101";
    elsif I(4) = '1' then
      A <= "100";
    elsif I(3) = '1' then
      A <= "011";
    elsif I(2) = '1' then
      A <= "010";
    elsif I(1) = '1' then
      A <= "001";
    elsif I(0) = '1' then
      A <= "000";
    else
      GS <= '0';
    end if;
    end process;
  end v1;

N-input AND Gate
-- an n-input AND gate
entity and_n is
  generic(n : positive := 8); --no. of inputs
  port(A : in bit_vector((n-1) downto 0);
       F : out bit);
end and_n;

architecture using_loop of and_n is
begin
  process(A)
  variable TEMP_F : bit;
  begin
    TEMP_F := '1';
    for I in A'range loop
      TEMP_F := TEMP_F and A(I);
    end loop;
    F <= TEMP_F;
  end process;
end using_loop;
A website devoted to the provision of on-line computer-based distance learning via the internet.

The Centre for Remote Access to Learning (CRAL) specialises in developing high quality teaching material for delivery via the internet. Examples of our work and the techniques developed are available [here](http://www.cral.ac.uk/) and also at the [online campus](http://www.cral.ac.uk/) at Bolton Institute. Commissions vary in size from a small item that supports conventional classroom-based teaching to an entire degree course delivered via the internet.

A specialist team of graphic designers and computer programmers works with academic staff to ensure that the material produced is professional in the way it is presented, the teaching is based on sound principles and the effectiveness of learning can be monitored and properly assessed.

A dedicated installation of web servers and powerful "number-crunching" computers at Bolton Institute provides a reliable service to distance learning students, including remote access to computer aided design (CAD) software. [More details.](http://www.cral.ac.uk/)

The [CRAL video](http://www.cral.ac.uk/) describes how the Centre was established and funded by the DfEE under the *Centres of Excellence* initiative. It lasts seven minutes and was produced entirely in-house.

CRAL supports the business community too; for details please follow this [link](http://www.cral.ac.uk/) to our commercial web design and development service.