VHDL By Example
Table of Contents:

```
-- ------------------------------------------------
-- Header information - Bus Breakout
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity bus_breakout is
  port (
         -- Inputs
         in_1   : in  std_logic_vector(3 downto 0);
         in_2   : in  std_logic_vector(3 downto 0);
         in_3   : in  std_logic;
         -- Outputs
         out_1  : out std_logic_vector(5 downto 0)
       );
end entity bus_breakout;

architecture bus_breakout_arch of bus_breakout is

begin
    --------------- Design implementation --------

    out_1 <= (   in_2(3 downto 2)
               & (in_1(3) AND in_2(1))
               & (in_1(2) AND in_2(0))
               & in_1(1 downto 0)
             );

end architecture bus_breakout_arch;
```

```
-- ------------------------------------------------
-- Header information - Bus Signals
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity bus_sigs is
  port (
         -- Inputs
         in_1   : in  std_logic_vector(3 downto 0);
         in_2   : in  std_logic_vector(3 downto 0);
         in_3   : in  std_logic;
         -- Outputs
         out_1  : out std_logic_vector(3 downto 0)
       );
end entity bus_sigs;

architecture bus_sigs_arch of bus_sigs is

   signal in_3_bus  : std_logic_vector(3 downto 0);

begin

    -- ------------- Design implementation --------

    in_3_bus <= (in_3 & in_3 & in_3 & in_3);

    out_1 <= ( (NOT in_3_bus) AND in_1) OR (in_3_bus AND in_2);

end architecture bus_sigs_arch;
```

```
-- ------------------------------------------------
-- Header information - Clock Buffer
-- ------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity clock_buffer is
  port (
         -- Inputs
         reset    : in  std_logic;
         clk_in   : in  std_logic;
         dat_in   : in  std_logic;
         -- Outputs
         dat_out  : out std_logic
       );
end entity clock_buffer;

architecture clock_buffer_arch of clock_buffer is

   signal clk  : std_logic;

   component BUFG
      port  (   I  : in  std_logic;
                O  : out std_logic
            );
   end component;


begin

   --------------- Design implementation --------

   -- clock buffer instantiation
   clock_buf : BUFG
      port map
            (   I  => clk_in,
                O  => clk
            );

   Reg_Proc: process (clk)
   begin
      if (reset = '1') then
         dat_out <= '0';
      elsif rising_edge(clk) then
         dat_out <= dat_in;
      end if;
   end process;

end architecture clock_buffer_arch;
```

```
-- ------------------------------------------------
-- Header information - D-flop with Enable and Clear
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity dflop_en_clr is
  port (
         -- Inputs
         clk     : in  std_logic;
         reset   : in  std_logic;
         clear_n : in  std_logic;
         enable  : in  std_logic;
         in_1    : in  std_logic;
         -- Outputs
         out_1   : out std_logic
       );
end entity dflop_en_clr;

architecture dflop_en_clr_arch of dflop_en_clr is

begin
    --------------- Design implementation --------

   Reg_Proc: process (clk, reset)
   begin
      if (reset = '1') then
         out_1 <= '0';
      elsif rising_edge(clk) then

         if (clear_n = '0') then
            out_1 <= '0';
         elsif (enable = '1') then
            out_1 <= in_1;
         end if;

      end if;
   end process;

end architecture dflop_en_clr_arch;
```

```
-- ------------------------------------------------
-- Header information - D-flop with Reset
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity dflop_n_reset is
  port (
         -- Inputs
         clk    : in  std_logic;
         reset  : in  std_logic;
         in_1   : in  std_logic;
         -- Outputs
         out_1  : out std_logic
       );
end entity dflop_n_reset;

architecture dflop_n_reset_arch of dflop_n_reset is

begin
    --------------- Design implementation --------

   Reg_Proc: process (clk, reset)
   begin
      if (reset = '1') then
         out_1 <= '0';
      elsif rising_edge(clk) then
         out_1 <= in_1;
      end if;
   end process;

end architecture dflop_n_reset_arch;
```

```
-- ------------------------------------------------
-- Header information - Full Dual Port Memory
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity full_dp_mem is
  port (
        -- port A
        clk_a        : in  std_logic;
        wr_a         : in  std_logic;
        address_a    : in  std_logic_vector(9 downto 0);
        dat_in_a     : in  std_logic_vector(15 downto 0);
        dat_out_a    : out std_logic_vector(15 downto 0);

        -- port B
        clk_b        : in  std_logic;
        wr_b         : in  std_logic;
        address_b    : in  std_logic_vector(9 downto 0);
        dat_in_b     : in  std_logic_vector(15 downto 0);
        dat_out_b    : out std_logic_vector(15 downto 0)
      );
end entity full_dp_mem;

architecture full_dp_mem_arch of full_dp_mem is

    -- 2D array type for the RAM
    type array_1kx16 is array(1023 downto 0) of
                           std_logic_vector(15 downto 0);

    -- declare the RAM.
    signal ram : array_1kx16;

begin

  process(clk_a)
  begin
    if rising_edge(clk_a) then -- Port A
      if(wr_a = '1') then
         ram(to_integer(unsigned(address_a))) <= dat_in_a;
         -- Read-during-write returns NEW data
         dat_out_a <= dat_in_a;
      else
         -- Read only
         dat_out_a <= ram(to_integer(unsigned(address_a)));
      end if;
    end if;
  end process;

  process(clk_b)
  begin
```

```
    if rising_edge(clk_b)  then -- Port B
      if(wr_b = '1') then
         ram(to_integer(unsigned(address_b))) <= dat_in_b;
         -- Read-during-write returns NEW data
         dat_out_b <= dat_in_b;
      else
         -- Read only
         dat_out_b <= ram(to_integer(unsigned(address_b)));
      end if;
    end if;
  end process;

end architecture full_dp_mem_arch;
```

```
-- ------------------------------------------------
-- Header information - Intermediate Wire Signals
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity intermed_signal is
  port (
         -- Inputs
         in_1   : in  std_logic;
         in_2   : in  std_logic;
         in_3   : in  std_logic;
         -- Outputs
         out_1  : out std_logic;
         out_2  : out std_logic
       );
end entity intermed_signal;

architecture intermed_signal_arch of intermed_signal is

   signal intermediate_sig : std_logic;

begin

    -- ------------- Design implementation --------

    intermediate_sig <= in_1 AND in_2; -- here's the intermediate signal

    out_1 <= intermediate_sig AND in_3;
    out_2 <= intermediate_sig OR  in_3;

end architecture intermed_signal_arch;
```

```
-- ------------------------------------------------
-- Header information - Modular Design #1
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity modular_1 is
  port (
         -- Inputs
         clk          : in  std_logic;
         reset        : in  std_logic;

         kill_clr     : in  std_logic;

         go_1         : in  std_logic;
         kill_1       : in  std_logic;
         go_2         : in  std_logic;
         kill_2       : in  std_logic;
         go_3         : in  std_logic;
         kill_3       : in  std_logic;

         -- Outputs
         kill_ltchd   : out std_logic;
         done_1       : out std_logic;
         done_2       : out std_logic;
         done_3       : out std_logic
       );
end entity modular_1;

architecture modular_1_arch of modular_1 is

   component state_machine
     port (
            -- Inputs
            clk       : in  std_logic;
            reset     : in  std_logic;

            go        : in  std_logic;
            kill      : in  std_logic;
            -- Outputs
            done      : out std_logic
          );
   end component;

begin
    --------------- Design implementation --------

  Go_Delay_1: state_machine
  port map
      (
```

```
       -- Inputs
       clk       => clk,
       reset     => reset,

       go        => go_1,
       kill      => kill_1,
       -- Outputs
       done      => done_1
    );

Go_Delay_2: state_machine
port map
    (
       -- Inputs
       clk       => clk,
       reset     => reset,

       go        => go_2,
       kill      => kill_2,
       -- Outputs
       done      => done_2
    );

Go_Delay_3: state_machine
port map
    (
       -- Inputs
       clk       => clk,
       reset     => reset,

       go        => go_3,
       kill      => kill_3,
       -- Outputs
       done      => done_3
    );

 Reg_Proc: process (clk, reset)
 begin
    if (reset = '1') then
       kill_ltchd   <=   '0';
    elsif rising_edge(clk) then

       if (   kill_1 = '1'
           OR kill_2 = '1'
           OR kill_3 = '1'
          ) then
          kill_ltchd <= '1';
       elsif (kill_clr = '1') then
          kill_ltchd <= '0';
       end if;

    end if;
 end process;
```

```
end architecture modular_1_arch;




-- ------------------------------------------------
-- Header information - Modular Design #2
-- ------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity modular_2 is
  port (
         -- Inputs
         clk          : in  std_logic;
         reset        : in  std_logic;

         kill_clr     : in  std_logic;

         go_1         : in  std_logic;
         kill_1       : in  std_logic;
         go_2         : in  std_logic;
         kill_2       : in  std_logic;
         go_3         : in  std_logic;
         kill_3       : in  std_logic;

         -- Outputs
         kill_ltchd   : out std_logic;
         done_3       : out std_logic
       );
end entity modular_2;

architecture modular_2_arch of modular_2 is

   component state_machine
     port (
            -- Inputs
            clk       : in  std_logic;
            reset     : in  std_logic;

            go        : in  std_logic;
            kill      : in  std_logic;
            -- Outputs
            done      : out std_logic
          );
   end component;

   signal done_1   : std_logic;
   signal g2       : std_logic;
   signal done_2   : std_logic;
   signal g3       : std_logic;

begin
    --------------- Design implementation --------
```

```
Go_Delay_1: state_machine
port map
    (
       -- Inputs
       clk       => clk,
       reset     => reset,

       go        => go_1,
       kill      => kill_1,
       -- Outputs
       done      => done_1
    );

g2 <= (done_1 OR go_2);

Go_Delay_2: state_machine
port map
    (
       -- Inputs
       clk       => clk,
       reset     => reset,

       go        => g2,
       kill      => kill_2,
       -- Outputs
       done      => done_2
    );

g3 <= (done_1 OR done_2 OR go_3);

Go_Delay_3: state_machine
port map
    (
       -- Inputs
       clk       => clk,
       reset     => reset,

       go        => g3,
       kill      => kill_3,
       -- Outputs
       done      => done_3
    );


 Reg_Proc: process (clk, reset)
 begin
    if (reset = '1') then
       kill_ltchd <= '0';
    elsif rising_edge(clk) then

       if (   kill_1 = '1'
           OR kill_2 = '1'
           OR kill_3 = '1'
          ) then
          kill_ltchd <= '1';
       elsif (kill_clr = '1') then
          kill_ltchd <= '0';
```

```
         end if;

      end if;
   end process;

end architecture modular_2_arch;
-- ------------------------------------------------
-- Header information - Sample Design for Simulation
-- ------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity sim_sample is
  port (
         -- Inputs
         clk        : in  std_logic;
         rst        : in  std_logic;
         --
         dat_in     : in  std_logic_vector(7 downto 0);
         enable     : in  std_logic;

         -- Outputs
         comp_cnt   : out std_logic_vector(9 downto 0)
       );
end entity sim_sample;

architecture sim_sample_arch of sim_sample is

   signal dat_in_d1      : std_logic_vector(7 downto 0);
   signal comp_cnt_lcl   : unsigned(9 downto 0);

begin

   Sim_Sample_Proc: process (clk, rst)
   begin
      if (rst = '1') then
         dat_in_d1    <= X"00";
         comp_cnt_lcl <= "00" & X"00";
      elsif rising_edge(clk) then
         if (enable = '1') then

            dat_in_d1 <= dat_in;

            if (dat_in_d1 = dat_in) then
               comp_cnt_lcl <= comp_cnt_lcl + 1;
            end if;

         end if;

      end if;
   end process;

   comp_cnt <= std_logic_vector(comp_cnt_lcl);

end architecture sim_sample_arch;
```

```
-- ------------------------------------------------
-- Header information - Simple D-flop
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity simple_dflop is
  port (
         -- Inputs
         clk    : in  std_logic;
         in_1   : in  std_logic;
         -- Outputs
         out_1  : out std_logic
       );
end entity simple_dflop;

architecture simple_dflop_arch of simple_dflop is

begin
    --------------- Design implementation --------

   Reg_Proc: process (clk)
   begin
      if rising_edge(clk) then
         out_1 <= in_1;
      end if;
   end process;

end architecture simple_dflop_arch;
```

```
-- ------------------------------------------------
-- Header information - Simple Dual Port Memory
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity simple_dp_mem is
  port (
        rst          : in  std_logic;
        clk          : in  std_logic;

        dat_in       : in  std_logic_vector(15 downto 0);
        wr_adr       : in  std_logic_vector(9 downto 0);
        wr_en        : in  std_logic;

        dat_out      : out std_logic_vector(15 downto 0);
        rd_adr       : in  std_logic_vector(9 downto 0)
      );
end entity simple_dp_mem;

architecture simple_dp_mem_arch of simple_dp_mem is

    -- 2D array type for the RAM
    type array_1kx16 is array(1023 downto 0) of
                           std_logic_vector(15 downto 0);

    -- declare the RAM.
    signal ram : array_1kx16;

begin

    process(clk)
    begin
      if rising_edge(clk) then

        if(wr_en = '1') then
           ram(to_integer(unsigned(wr_adr))) <= dat_in;
           -- Read-during-write returns NEW data
           dat_out <= ram(to_integer(unsigned(rd_adr)));
        else
           -- Read only
           dat_out <= ram(to_integer(unsigned(rd_adr)));
        end if;

      end if;
    end process;

end architecture simple_dp_mem_arch;
```

```
-- ------------------------------------------------
--  “Simple In and Out”
-- Header information -- details about the context,
-- constraints, etc..
--
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity simple_in_n_out is
  port (
         -- Inputs
         in_1   : in  std_logic;
         in_2   : in  std_logic;
         in_3   : in  std_logic;
         -- Outputs
         out_1  : out std_logic;
         out_2  : out std_logic
       );
end entity simple_in_n_out;

architecture simple_in_n_out_arch of simple_in_n_out is

begin

    -- ------------- Design implementation --------

    out_1 <= in_1 AND in_2 AND in_3;
    out_2 <= in_1 OR  in_2 OR  in_3;

end architecture simple_in_n_out_arch;
```

```
-- ------------------------------------------------
-- Header information - Single-Port Memory
-- ------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity single_port_mem is
  port (
        clk      : in    std_logic;

        data_io  : inout std_logic_vector(15 downto 0);
        address  : in    std_logic_vector(9 downto 0);
        wr_en    : in    std_logic;
        rd       : in    std_logic
      );
end entity single_port_mem;

architecture single_port_mem_arch of single_port_mem is

   -- 2D array type for the RAM
   type array_1kx16 is array(1023 downto 0) of
                        std_logic_vector(15 downto 0);

   -- declare the RAM.
   signal ram : array_1kx16;

   -- intermediate RAM output
   signal dat_out    : std_logic_vector(15 downto 0);
   signal rd_d1      : std_logic;

begin

   process(clk)
   begin
     if rising_edge(clk) then

       -- writes

       if(wr_en = '1') then
         ram(to_integer(unsigned(address))) <= data_io;
       end if;

       -- reads

       dat_out <= ram(to_integer(unsigned(address)));

       rd_d1 <= rd;

     end if;
   end process;
```

```
   data_io <= dat_out  when rd_d1 = '1'
                       else
              "ZZZZZZZZZZZZZZZZ";

end architecture single_port_mem_arch;
-- ------------------------------------------------
-- Header information - Standard Mux #1
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity standard_mux_1 is
  port (
         -- Inputs
         in_1   : in  std_logic_vector(3 downto 0);
         in_2   : in  std_logic_vector(3 downto 0);
         in_3   : in  std_logic;
         -- Outputs
         out_1  : out std_logic_vector(3 downto 0)
       );
end entity standard_mux_1;

architecture standard_mux_1_arch of standard_mux_1 is

begin
    --------------- Design implementation --------

   with (in_3) select
      out_1 <=  in_1 when '0',
                in_2 when '1',
               "0000" when others;

end architecture standard_mux_1_arch;
```

```
-- ------------------------------------------------
-- Header information - Standard Mux #2
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity standard_mux_2 is
  port (
         -- Inputs
         in_1   : in  std_logic_vector(3 downto 0);
         in_2   : in  std_logic_vector(3 downto 0);
         in_3   : in  std_logic;
         -- Outputs
         out_1  : out std_logic_vector(3 downto 0)
       );
end entity standard_mux_2;

architecture standard_mux_2_arch of standard_mux_2 is

begin
    ---------------- Design implementation --------

   out_1 <=  in_1 when (in_3 = '0') else in_2;

end architecture standard_mux_2_arch;
```

```
-- ------------------------------------------------
-- Header information - State Machine 1
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity state_machine is
  port (
         -- Inputs
         clk       : in  std_logic;
         reset     : in  std_logic;

         go        : in  std_logic;
         kill      : in  std_logic;
         -- Outputs
         done      : out std_logic
       );
end entity state_machine;

architecture state_machine_arch of state_machine is

   signal count    : unsigned(7 downto 0);

   type state_labels is  (  Idle,
                            Active,
                            Finish,
                            Abort
                         );

   signal state_reg  : state_labels;

begin
    --------------- Design implementation --------

   State_Proc: process (clk, reset)
   begin
     if (reset = '1') then
        state_reg  <=  Idle;
     elsif rising_edge(clk) then

       case (state_reg) is

         when Idle =>

           if (go = '1') then          state_reg <= Active;
           end if;

         when Active =>

           if (kill = '1') then        state_reg <= Abort;
           elsif (count = X"64") then  state_reg <= Finish;
           end if;

         when Finish =>                state_reg <= Idle;
```

```
        when Abort =>

          if (kill /= '1') then       state_reg <= Idle;
          end if;

        when others =>                state_reg <= Idle;

       end case;

     end if;
   end process;

   Reg_Proc: process (clk, reset)
   begin
      if (reset = '1') then
         count  <= X"00";
         done   <=   '0';
      elsif rising_edge(clk) then

         -- duration counter
         if (   state_reg = Finish
             OR state_reg = Abort
            ) then
            count <= X"00";
         elsif (state_reg = Active) then
            count <= count + 1;
         end if;

         -- output register
         if (state_reg = Finish) then
            done <= '1';
         else
            done <= '0';
         end if;

      end if;
   end process;

end architecture state_machine_arch;
```

```
-- ------------------------------------------------
-- Header information - State Machine 2
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity state_machine_2 is
  port (
         -- Inputs
         clk       : in  std_logic;
         reset     : in  std_logic;

         go        : in  std_logic;
         kill      : in  std_logic;
         -- Outputs
         done      : out std_logic
       );
end entity state_machine_2;

architecture state_machine_2_arch of state_machine_2 is

   signal count    : unsigned(7 downto 0);

   type state_labels is  (  Idle,
                            Active,
                            Finish,
                            Abort
                         );

   signal state_reg  : state_labels;

begin
    --------------- Design implementation --------

   Combined_Proc: process (clk, reset)
   begin
      if (reset = '1') then
         state_reg  <=  Idle;
         count      <= X"00";
         done       <=   '0';
      elsif rising_edge(clk) then

         case (state_reg) is

            when Idle =>

               if (go = '1') then
                  state_reg <= Active;
               end if;

               count <= X"00";
               done  <=   '0';

            when Active =>
```

```
               count <= count + 1;
               done  <= '0';

               if (kill = '1') then
                  state_reg <= Abort;
               elsif (count = X"64") then
                  state_reg <= Finish;
               end if;

            when Finish =>

               count <= X"00";
               done  <=   '1';

               state_reg <= Idle;

            when Abort =>

               count <= X"00";
               done  <=   '0';

               if (kill /= '1') then
                  state_reg <= Idle;
               end if;

            when others =>

               count <= X"00";
               done  <=   '0';
               state_reg <= Idle;

         end case;

      end if;
   end process;

end architecture state_machine_2_arch;
```

```
-- ------------------------------------------------
-- Simple Testbench Using Embedded, Explicit Vectors
-- ------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity tb_sim_sample_1 is
end entity tb_sim_sample_1;

architecture Behavioral of tb_sim_sample_1 is



-----------------------------------------
type my_array_type is array(0 to 6) of std_logic_vector(7 downto 0);

   FUNCTION
   largest_value
      (
       array_in  : my_array_type;
       amount    : integer
      ) RETURN integer is
      variable temp_max : std_logic_vector(7 downto 0);
      variable index    : integer;
   begin
      temp_max := X"00";
      for i in 0 to amount loop
         if (array_in(i) > temp_max) then
           temp_max := array_in(i);
           index    := i;
         end if;
      end loop;
      RETURN index;
   END largest_value;

signal function_in : my_array_type;
signal function_out: integer;

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


   component sim_sample
     port (
           -- Inputs
           clk        : in  std_logic;
           rst        : in  std_logic;
           --
           dat_in     : in  std_logic_vector(7 downto 0);
           enable     : in  std_logic;

           -- Outputs
           comp_cnt   : out std_logic_vector(9 downto 0)
          );
   end component;

   constant HALF_PERIOD  : time := 5 ns;  -- 100MHz = 10ns
```

```
   -- module-under-test inputs
   signal clk          : std_logic;
   signal rst          : std_logic;
   signal data_val     : std_logic_vector(7 downto 0);
   signal en           : std_logic;

   -- module-under-test outputs
   signal comp_cnt     : std_logic_vector(9 downto 0);

begin

-------------------------------------------
  function_in(0) <= X"01";
  function_in(1) <= X"02";
  function_in(2) <= X"07";
  function_in(3) <= X"04";
  function_in(4) <= X"05";
  function_in(5) <= X"06";
  function_in(6) <= X"07";

  function_out <= largest_value(function_in, 4);

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


   -- module under test

   MUT: sim_sample
   port map
     (
       -- Inputs
       clk        => clk,
       rst        => rst,
       --
       dat_in     => data_val,
       enable     => en,

       -- Outputs
       comp_cnt   => comp_cnt
     );

   ----------------------------------------------------
   ------  Clock Generator
   ----------------------------------------------------

   clk <= '0' after HALF_PERIOD when clk = '1' else
          '1' after HALF_PERIOD;

   ----------------------------------------------------
   ------  Reset Generator
   ----------------------------------------------------

   Reset_Gen : process
   begin
     -- generate reset
     for i in 1 to 5 loop
```

```
      if (i < 4) then
         rst <= '1';
      else
         rst <= '0';
      end if;
      wait until falling_edge(clk);
    end loop;
    --
    -- de-activate this process
    wait on rst;
  end process Reset_Gen;

  -------------------------------------------------
  ------  Vector Generator
  -------------------------------------------------

  Vector_Generate_explicit: process

  begin

    data_val <= X"00";
    en       <= '0';

    wait until falling_edge(rst);

    for j in 1 to 3 loop
       wait until falling_edge(clk);
    end loop;

    en <= '1';

    wait until falling_edge(clk);
    data_val <= X"01";
    wait until falling_edge(clk);
    data_val <= X"20";
    wait until falling_edge(clk);
    data_val <= X"21";
    wait until falling_edge(clk);
    data_val <= X"21";
    wait until falling_edge(clk);
    data_val <= X"33";
    wait until falling_edge(clk);
    data_val <= X"56";
    wait until falling_edge(clk);
    data_val <= X"56";
    wait until falling_edge(clk);
    data_val <= X"33";
    wait until falling_edge(clk);

    en <= '0';

    -- de-activate this process
    wait on rst;

  end process Vector_Generate_explicit;

end architecture Behavioral;
```

```
-- ------------------------------------------------
-- Simple Testbench Using Embedded, Automatic Vectors
-- ------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity tb_sim_sample_2 is
end entity tb_sim_sample_2;

architecture Behavioral of tb_sim_sample_2 is

   component sim_sample
     port (
           -- Inputs
           clk        : in  std_logic;
           rst        : in  std_logic;
           --
           dat_in     : in  std_logic_vector(7 downto 0);
           enable     : in  std_logic;

           -- Outputs
           comp_cnt   : out std_logic_vector(9 downto 0)
          );
   end component;

   constant HALF_PERIOD   : time := 5 ns; -- 100MHz = 10ns
   constant QUANT_VECTORS : integer := 30; -- # of vectors
   -- semi-random seed
   constant SEED   : unsigned(5 downto 0) := "100111";

   -- module-under-test inputs
   signal clk          : std_logic;
   signal rst          : std_logic;
   signal data_val     : std_logic_vector(7 downto 0);
   signal en           : std_logic;

   -- module-under-test outputs
   signal comp_cnt     : std_logic_vector(9 downto 0);

begin

   -- module under test

   MUT: sim_sample
   port map
     (
       -- Inputs
       clk        => clk,
       rst        => rst,
       --
       dat_in     => data_val,
       enable     => en,

       -- Outputs
       comp_cnt   => comp_cnt
     );
```

```
-------------------------------------------------
------  Clock Generator
-------------------------------------------------

clk <= '0' after HALF_PERIOD when clk = '1' else
       '1' after HALF_PERIOD;

-------------------------------------------------
------  Reset Generator
-------------------------------------------------

Reset_Gen : process
begin
  -- generate reset
  for i in 1 to 5 loop
     if (i < 4) then
        rst <= '1';
     else
        rst <= '0';
     end if;
     wait until falling_edge(clk);
  end loop;
  --
  -- de-activate this process
  wait on rst;
end process Reset_Gen;

-------------------------------------------------
------  Vector Generator
-------------------------------------------------

Vector_Generate_Implicit: process

   variable semi_random : unsigned(5 downto 0) := "101010";
   variable data_val_un : unsigned(7 downto 0) := X"00";

begin

  data_val <= X"00";
  en       <= '0';

  wait until falling_edge(rst);

  for j in 1 to 3 loop
     wait until falling_edge(clk);
  end loop;

  en <= '1';

  for i in 1 to QUANT_VECTORS loop

     semi_random := semi_random + SEED;

     if (semi_random(5 downto 4) /= "00") then
        data_val_un := data_val_un + 1;
     end if;
```

```
         data_val <= std_logic_vector(data_val_un);

         wait until falling_edge(clk);

      end loop;

      en <= '0';

      -- de-activate this process
      wait on rst;

   end process Vector_Generate_Implicit;

end architecture Behavioral;
```