```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--
-- ***************************************************************************
-- ** Copyright (c) 1995-2011 Xilinx, Inc.  All rights reserved.            **
-- **                                                                       **
-- ** Xilinx, Inc.                                                          **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
-- ** FOR A PARTICULAR PURPOSE.                                             **
-- **                                                                       **
-- ***************************************************************************
--
------------------------------------------------------------------------------
-- Filename:          user_logic.vhd
-- Version:           1.00.a
-- Description:       User logic.
-- Date:              Tue Nov 15 17:51:08 2016 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
--   combinatorial signals:                 "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                       "*cnt*"
--   clock enable signals:                  "*_ce"
--   internal version of output port:       "*_i"
--   device pins:                           "*_pin"
--   ports:                                 "- Names begin with Uppercase"
--   processes:                             "*_PROCESS"
--   component instantiations:              "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------

-- DO NOT EDIT BELOW THIS LINE --------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v3_00_a;
```

```
use proc_common_v3_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_NUM_REG                    -- Number of software accessible registers
--   C_SLV_DWIDTH                 -- Slave interface data bus width
--
-- Definition of Ports:
--   Bus2IP_Clk                   -- Bus to IP clock
--   Bus2IP_Resetn                -- Bus to IP reset
--   Bus2IP_Data                  -- Bus to IP data bus
--   Bus2IP_BE                    -- Bus to IP byte enables
--   Bus2IP_RdCE                  -- Bus to IP read chip enable
--   Bus2IP_WrCE                  -- Bus to IP write chip enable
--   IP2Bus_Data                  -- IP to Bus data bus
--   IP2Bus_RdAck                 -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck                 -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error                 -- IP to Bus error response
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_NUM_REG                      : integer              := 8;
    C_SLV_DWIDTH                   : integer              := 32
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
    digit_0 : out std_logic;
    digit_1 : out std_logic;
    digit_2 : out std_logic;
    digit_3 : out std_logic;
    digit_4 : out std_logic;
    digit_5 : out std_logic;

    seg_a : out std_logic;
    seg_b : out std_logic;
    seg_c : out std_logic;
    seg_d : out std_logic;
    seg_e : out std_logic;
    seg_f : out std_logic;
    seg_g : out std_logic;
```

```vhdl
      -- ADD USER PORTS ABOVE THIS LINE ------------------

      -- DO NOT EDIT BELOW THIS LINE ---------------------
      -- Bus protocol ports, do not add to or delete
      Bus2IP_Clk                     : in  std_logic;
      Bus2IP_Resetn                  : in  std_logic;
      Bus2IP_Data                    : in  std_logic_vector(C_SLV_DWIDTH-1 downto 0);
      Bus2IP_BE                      : in  std_logic_vector(C_SLV_DWIDTH/8-1 downto 0);
      Bus2IP_RdCE                    : in  std_logic_vector(C_NUM_REG-1 downto 0);
      Bus2IP_WrCE                    : in  std_logic_vector(C_NUM_REG-1 downto 0);
      IP2Bus_Data                    : out std_logic_vector(C_SLV_DWIDTH-1 downto 0);
      IP2Bus_RdAck                   : out std_logic;
      IP2Bus_WrAck                   : out std_logic;
      IP2Bus_Error                   : out std_logic
      -- DO NOT EDIT ABOVE THIS LINE ---------------------
    );

  attribute MAX_FANOUT : string;
  attribute SIGIS : string;

  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Resetn : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic

  ------------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ------------------------------------------
  signal slv_reg0                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg1                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg2                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg3                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg4                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg5                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg6                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg7                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg_write_sel              : std_logic_vector(7 downto 0);
  signal slv_reg_read_sel               : std_logic_vector(7 downto 0);
  signal slv_ip2bus_data                : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_read_ack                   : std_logic;
  signal slv_write_ack                  : std_logic;

  signal counter: std_logic_vector(15 downto 0) := "0000000000000000";  -- should work
  signal current_register: std_logic_vector(31 downto 0);
  signal digit_out: std_logic_vector(5 downto 0);
  signal seg: std_logic_vector(6 downto 0);
begin

  --------------------------------------------------------------------------
```

```
    -- Begin logic required for 7 segment display

      digit_0 <= digit_out(0);
      digit_1 <= digit_out(1);
      digit_2 <= digit_out(2);
      digit_3 <= digit_out(3);
      digit_4 <= digit_out(4);
      digit_5 <= digit_out(5);

      seg_a <=  seg(6);
      seg_b <=  seg(5);
      seg_c <=  seg(4);
      seg_d <=  seg(3);
      seg_e <=  seg(2);
      seg_f <=  seg(1);
      seg_g <=  seg(0);

  --current_register <= slv_reg0;
    with counter(15 downto 13) select current_register <=
      slv_reg0 when "000",
      slv_reg1 when "001",
      slv_reg2 when "010",
      slv_reg3 when "011",
      slv_reg4 when "100",
      slv_reg5 when "101",
      slv_reg6 when others; -- Error, should never happen

    -- Digit out is 1-hot encoded
    -- (well, not really, but if the digits are all different it is)
    with counter(15 downto 13) select digit_out <=
      "000001" when "000",  -- Digit 0
      "000010" when "001",  -- Digit 1
      "000100" when "010",  -- Digit 2
      "001000" when "011",  -- Digit 3
      "010000" when "100",  -- Digit 4
      "100000" when "101",  -- Digit 5
      "000000" when others; -- This should never happen
  --    digit_out <= "111000";


    -- The binary is arranged in
    -- A B C D E F G in seg
    -- In current_register it is whatever is in the register.
    --seg <= current_register(6 downto 0);  -- with 2^7, 64 different combinations.
  with current_register select seg <=
    "1111110" when "00000000000000000000000000000000",  -- 0
    "0110000" when "00000000000000000000000000000001",  -- 1
    "1101101" when "00000000000000000000000000000010",  -- 2
    "1111001" when "00000000000000000000000000000011",  -- 3
    "0110011" when "00000000000000000000000000000100",  -- 4
    "1011011" when "00000000000000000000000000000101",  -- 5
    "1011111" when "00000000000000000000000000000110",  -- 6
    "1110000" when "00000000000000000000000000000111",  -- 7
    "1111111" when "00000000000000000000000000001000",  -- 8
    "1110011" when "00000000000000000000000000001001",  -- 9

    "1110111" when "00000000000000000000000000001010",  -- a
    "0011111" when "00000000000000000000000000001011",  -- b
    "0001101" when "00000000000000000000000000001100",  -- c
    "0111101" when "00000000000000000000000000001101",  -- d
    "1001111" when "00000000000000000000000000001110",  -- e
    "1000111" when "00000000000000000000000000001111",  -- f
    "1111011" when "00000000000000000000000000010000",  -- g
  ---------------------- 11 is missing
    "0010111" when "00000000000000000000000000010010",  -- h
    "0110000" when "00000000000000000000000000010011",  -- i
    "0111100" when "00000000000000000000000000010100",  -- j
    "0010111" when "00000000000000000000000000010101",  -- k
    "0001110" when "00000000000000000000000000010110",  -- l
    "0010101" when "00000000000000000000000000010111",  -- m
    "0010110" when "00000000000000000000000000011000",  -- n
    "1111110" when "00000000000000000000000000011001",  -- o
    "1100111" when "00000000000000000000000000011010",  -- p
    "0011101" when "00000000000000000000000000011011",  -- q
    "0000101" when "00000000000000000000000000011100",  -- r
    "1011011" when "00000000000000000000000000011101",  -- s
    "1000110" when "00000000000000000000000000011110",  -- t
    "0011100" when "00000000000000000000000000011111",  -- u
    "0011100" when "00000000000000000000000000100000",  -- v
    "0011100" when "00000000000000000000000000100001",  -- w
    "0110111" when "00000000000000000000000000100010",  -- x
    "0110011" when "00000000000000000000000000100011",  -- y
    "1101101" when "00000000000000000000000000100100",  -- z

    "0000001" when "00000000000000000000000000100101", -- _
    "0000000" when "11111111111111111111111111111111", -- ' '
    "0111111" when others; -- displays a '-' on error


    process(Bus2IP_Clk)
    begin
    --finish the clock ticking.
      if(Bus2IP_Clk'EVENT and Bus2IP_Clk = '1') then
        --if(counter = "101") then
          --counter <= "000";
        --else
          --counter <= counter + 1;
        --end if;
        counter <= counter + 1;
      end if;
    end process;


    -- End logic required for 7 segment display
    --------------------------------------------------------------------------



    ------------------------------------------
    -- Example code to read/write user logic slave model s/w accessible registers
    --
    -- Note:
    -- The example code presented here is to show you one way of reading/writing
    -- software accessible registers implemented in the user logic slave model.
```

```vhdl
  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
  -- to one software accessible register by the top level template. For example,
  -- if you have four 32 bit software accessible registers in the user logic,
  -- you are basically operating on the following memory mapped registers:
  --
  --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
  --                     "1000"   C_BASEADDR + 0x0
  --                     "0100"   C_BASEADDR + 0x4
  --                     "0010"   C_BASEADDR + 0x8
  --                     "0001"   C_BASEADDR + 0xC
  --
  ------------------------------------------
  slv_reg_write_sel <= Bus2IP_WrCE(7 downto 0);
  slv_reg_read_sel  <= Bus2IP_RdCE(7 downto 0);
  slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or Bus2IP_WrCE(5) or Bus2IP_WrCE(6) or Bus2IP_WrCE(7);
  slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or Bus2IP_RdCE(5) or Bus2IP_RdCE(6) or Bus2IP_RdCE(7);

  -- implement slave model software accessible register(s)
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin

    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Resetn = '0' then
        slv_reg0 <= (others => '0');
        slv_reg1 <= (others => '0');
        slv_reg2 <= (others => '0');
        slv_reg3 <= (others => '0');
        slv_reg4 <= (others => '0');
        slv_reg5 <= (others => '0');
        slv_reg6 <= (others => '0');
        slv_reg7 <= (others => '0');
      else
        case slv_reg_write_sel is
          when "10000000" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg0(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when "01000000" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg1(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when "00100000" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg2(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when "00010000" =>
```

```
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg3(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when "00001000" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg4(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when "00000100" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg5(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when "00000010" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg6(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when "00000001" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg7(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when others => null;
        end case;
      end if;
    end if;

  end process SLAVE_REG_WRITE_PROC;

  -- implement slave model software accessible register(s) read mux
  SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1, slv_reg2, slv_reg3, slv_reg4, slv_reg5, slv_reg6, slv_reg7 ) is
  begin

    case slv_reg_read_sel is
      when "10000000" => slv_ip2bus_data <= slv_reg0;
      when "01000000" => slv_ip2bus_data <= slv_reg1;
      when "00100000" => slv_ip2bus_data <= slv_reg2;
      when "00010000" => slv_ip2bus_data <= slv_reg3;
      when "00001000" => slv_ip2bus_data <= slv_reg4;
      when "00000100" => slv_ip2bus_data <= slv_reg5;
      when "00000010" => slv_ip2bus_data <= slv_reg6;
      when "00000001" => slv_ip2bus_data <= slv_reg7;
      when others => slv_ip2bus_data <= (others => '0');
    end case;

  end process SLAVE_REG_READ_PROC;

  ------------------------------------------
  -- Example code to drive IP to Bus signals
  ------------------------------------------
  IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
                  (others => '0');

  IP2Bus_WrAck <= slv_write_ack;
  IP2Bus_RdAck <= slv_read_ack;
  IP2Bus_Error <= '0';

end IMP;

```