```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--
-- ***************************************************************************
-- ** Copyright (c) 1995-2008 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:              Sat Apr 04 16:39:42 2009 (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_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

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

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_DWIDTH                 -- Slave interface data bus width
--   C_NUM_REG                    -- Number of software accessible registers
--
-- Definition of Ports:
--   Bus2IP_Clk                   -- Bus to IP clock
--   Bus2IP_Reset                 -- 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_SLV_DWIDTH                   : integer              := 32;
    C_NUM_REG                      : integer              := 3
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
    -- 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_Reset                   : in  std_logic;
    Bus2IP_Data                    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );

  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : 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

   component DFT_CORE
      generic
         (
         SLV_DWIDTH: integer := 32
         );
      port
         (
         Clk: in std_logic;
         Reset: in std_logic;
         data_written : in  std_logic_vector(0 to 2);
         data_read : in  std_logic_vector(0 to 2);
         command_bit : in  std_logic;
         value_reg : in  std_logic_vector (0 to SLV_DWIDTH-1);
         dft_command : out  std_logic_vector (0 to SLV_DWIDTH-1);
         dft_index : out  std_logic_vector (0 to SLV_DWIDTH-1);
         dft_value : out  std_logic_vector (0 to SLV_DWIDTH-1)
         );
   end component;

  ------------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ------------------------------------------
  signal slv_reg0                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
```

```
  signal slv_reg1                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg2                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel              : std_logic_vector(0 to 2);
  signal slv_reg_read_sel               : std_logic_vector(0 to 2);
  signal slv_ip2bus_data                : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack                   : std_logic;
  signal slv_write_ack                  : std_logic;
  signal command_dir_reg                : std_logic;
  signal go_bit                         : std_logic;
  signal get_bit                        : std_logic;
  signal put_bit                        : std_logic;
  signal done_bit                       : std_logic;
  signal dft_command                    : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal dft_index                      : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal dft_value                      : std_logic_vector(0 to C_SLV_DWIDTH-1);

begin

  --USER logic implementation added here

   DFT_CORE_INSTANCE: DFT_CORE
      generic map
         (
         SLV_DWIDTH => C_SLV_DWIDTH
         )
      port map
      (
      Clk => Bus2IP_Clk,
      Reset => Bus2IP_Reset,
      data_written => slv_reg_write_sel,
      data_read => slv_reg_read_sel,
      command_bit => slv_reg0(31),
      value_reg => slv_reg2,
      dft_command => dft_command,
      dft_index => dft_index,
      dft_value => dft_value
      );

  ------------------------------------------
  -- 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.
  -- 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(0 to 2);
   slv_reg_read_sel  <= Bus2IP_RdCE(0 to 2);
   slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2);
   slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2);

   go_bit <= slv_reg0(31);
   get_bit <= dft_command(30);
   put_bit <= dft_command(29);
   done_bit <= dft_command(28);

  -- 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_Reset = '1' ) then
            slv_reg0 <= (others => '0');
            slv_reg1 <= (others => '0');
            slv_reg2 <= (others => '0');

-- Initialize the mux feedback register.
            command_dir_reg <= '0';
```

```
         else
            command_dir_reg <= go_bit;

-- If the command_dir_reg is set to 0, then allow the PPC to set
-- the command register. This happens just once -- before the hardware
-- dft begins computing.
            if ( command_dir_reg = '0' and slv_reg_write_sel = "100" ) then
               slv_reg0 <= Bus2IP_Data;

-- Load the command register when the hardware DFT wants to communicate
-- with the PPC.
            else
               if ( get_bit = '1' or put_bit = '1' or done_bit = '1' ) then
                  slv_reg0 <= dft_command;
               end if;
            end if;

-- If the next command is a read or write, then latch the index, otherwise
-- hold the current value.
            if ( get_bit = '1' or put_bit = '1' ) then
               slv_reg1 <= dft_index;
            end if;

-- Let PPC write the value register if PPC indicates write op. Don't qualify
-- with get_bit because get_bit is set back to 0 after the first cycle. When
-- svl_reg_write_sel is 001, then there is valid data on Bus2IP_Data, latch
-- it into slv_reg2.
            if ( slv_reg_write_sel = "001" ) then
               slv_reg2 <= Bus2IP_Data;

-- If next command is a write, then put the value to be read by the PPC in
-- the value register.
            elsif ( put_bit = '1' ) then
               slv_reg2 <= dft_value;

            end if;
         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 ) is
  begin

    case slv_reg_read_sel is
      when "100" => slv_ip2bus_data <= slv_reg0;
      when "010" => slv_ip2bus_data <= slv_reg1;
      when "001" => slv_ip2bus_data <= slv_reg2;
      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;
```