user_logic.vhd  Tue Feb 17 16:23:53 2009  1

-- 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: Tue Feb 17 15:55:25 2009 (by Create and Import Peripheral Wizard)
-- VHDL Standard: VHDL'93

-- Naming Conventions:
-- active low signals: "_*n"
-- clock signals: "clk", "clk_div#", "clock_#x"
-- reset signals: "rst", "rst_n"
-- generics: "C_*"
-- user defined types: "*TYPE"
-- state machine next state: "_*ns"
-- state machine current state: "_*cs"
-- combinational 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
-- C_NUM_INTR -- Number of interrupt event

-- 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
-- IP2Bus_IntrEvent             -- IP to Bus interrupt event
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ----------------
    --USER generics added here
    NUM_SWITCHES : integer := 8; -- Width of input & output
    DELAY : integer := 8;
    -- 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              := 1;
    C_NUM_INTR  : integer              := 1
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE -------------------
    --USER ports added here
    INPUT_SWITCH_ARRAY : in std_logic_vector((NUM_SWITCHES-1) downto 0);
    -- 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;
    IP2Bus_IntrEvent : out std_logic_vector(0 to C_NUM_INTR-1)
    -- 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 IMP of user_logic is

component SWITCH_DEBOUNCER_CORE
  generic
    (
      NUM_SWITCHES : integer := 8; -- Width of input & output
      DELAY : integer := 8
    );
  port
    (
      RESET : in std_logic;
      CLOCK : in std_logic;
      INPUT_SWITCH_ARRAY : in std_logic_vector((NUM_SWITCHES-1) downto 0);
      OUTPUT_SWITCH_ARRAY : out std_logic_vector((NUM_SWITCHES-1) downto 0);
      NEW_SWITCH_STATE : out std_logic;
      ACKNOWLEDGE : in std_logic
    );

end component;

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

SWITCH_DEBOUNCER_CORE_INSTANCE: SWITCH_DEBOUNCER_CORE

generic map
{
NUM_SWITCHES => NUM_SWITCHES,
DELAY => DELAY
}

port map
{
RESET => Bus2IP_Reset,
CLOCK => Bus2IP_Clk,
INPUT_SWITCH_ARRAY => INPUT_SWITCH_ARRAY,
OUTPUT_SWITCH_ARRAY => OUTPUT_SWITCH_ARRAY,
NEW_SWITCH_STATE => NEW_SWITCH_STATE,
ACKNOWLEDGE => NEW_SWITCH_STATE
};

-- 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 0);
slv_reg_read_sel <= Bus2IP_RdCE(0 to 0);
slv_write_ack <= Bus2IP_WrCE(0);
slv_read_ack <= Bus2IP_RdCE(0);

-- implement slave model software accessible register(s) read mux
SLAVE_REG_READ_PROC : process( slv_reg_read_sel, OUTPUT_SWITCH_ARRAY ) is
begin
case slv_reg_read_sel is
when "1" => slv_ip2bus_data(0 to (C_SLV_DWIDTH-NUM_SWITCHES-1)) <= (others => '0');
slv_ip2bus_data((C_SLV_DWIDTH-NUM_SWITCHES) to (C_SLV_DWIDTH-1)) <= OUTPUT_SWITCH_ARRAY;
when others => slv_ip2bus_data <= (others => '0');
end case;
end process SLAVE_REG_READ_PROC;

-- Example code to generate user logic interrupts
```
-- Note:
-- The example code presented here is to show you one way of generating
-- interrupts from the user logic. This code snippet infers a counter
-- and generate the interrupts whenever the counter rollover (the counter
-- will rollover `21 sec @50Mhz).

---------------------------------------------------------------------------
IP2Bus_IntrEvent(0) <= NEW_SWITCH_STATE;
---------------------------------------------------------------------------
-- 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;