```
1  ------------------------------------------------------------------------------
2  -- user_logic.vhd - entity/architecture pair
3  ------------------------------------------------------------------------------
4  --
5  -- ****************************************************************************
6  -- ** Copyright (c) 1995-2011 Xilinx, Inc.  All rights reserved.            **
7  -- **                                                                        **
8  -- ** Xilinx, Inc.                                                          **
9  -- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
10 -- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
11 -- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
12 -- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
13 -- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
14 -- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
15 -- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
16 -- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
17 -- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
18 -- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
19 -- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
20 -- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
21 -- ** FOR A PARTICULAR PURPOSE.                                            **
22 -- **                                                                        **
23 -- ****************************************************************************
24 --
25 ------------------------------------------------------------------------------
26 -- Filename:          user_logic.vhd
27 -- Version:           1.00.a
28 -- Description:       User logic.
29 -- Date:              Wed Nov 02 16:29:35 2016 (by Create and Import Peripheral Wizard)
30 -- VHDL Standard:     VHDL'93
31 ------------------------------------------------------------------------------
32 -- Naming Conventions:
33 --   active low signals:                    "*_n"
34 --   clock signals:                         "clk", "clk_div#", "clk_#x"
35 --   reset signals:                         "rst", "rst_n"
36 --   generics:                              "C_*"
37 --   user defined types:                    "*_TYPE"
38 --   state machine next state:              "*_ns"
39 --   state machine current state:           "*_cs"
40 --   combinatorial signals:                 "*_com"
41 --   pipelined or register delay signals:   "*_d#"
42 --   counter signals:                       "*cnt*"
43 --   clock enable signals:                  "*_ce"
44 --   internal version of output port:       "*_i"
45 --   device pins:                           "*_pin"
46 --   ports:                                 "- Names begin with Uppercase"
47 --   processes:                             "*_PROCESS"
48 --   component instantiations:              "<ENTITY_>I_<#|FUNC>"
49 ------------------------------------------------------------------------------
50
51 -- DO NOT EDIT BELOW THIS LINE --------------------
52 library ieee;
53 use ieee.std_logic_1164.all;
54 use ieee.std_logic_arith.all;
55 use ieee.std_logic_unsigned.all;
56
57
```

```
58  ------------------------------------------------------------------------------
    ------------------------------------
59  -- uncomment the next two files
60  --library proc_common_v3_00_a;
61  --use proc_common_v3_00_a.proc_common_pkg.all;
62
63  -- DO NOT EDIT ABOVE THIS LINE --------------------
64
65  --USER libraries added here
66
67  ------------------------------------------------------------------------------
68  -- Entity section
69  ------------------------------------------------------------------------------
70  -- Definition of Generics:
71  --   C_NUM_REG                  -- Number of software accessible registers
72  --   C_SLV_DWIDTH               -- Slave interface data bus width
73  --
74  -- Definition of Ports:
75  --   Bus2IP_Clk                 -- Bus to IP clock
76  --   Bus2IP_Resetn              -- Bus to IP reset
77  --   Bus2IP_Data                -- Bus to IP data bus
78  --   Bus2IP_BE                  -- Bus to IP byte enables
79  --   Bus2IP_RdCE                -- Bus to IP read chip enable
80  --   Bus2IP_WrCE                -- Bus to IP write chip enable
81  --   IP2Bus_Data                -- IP to Bus data bus
82  --   IP2Bus_RdAck               -- IP to Bus read transfer acknowledgement
83  --   IP2Bus_WrAck               -- IP to Bus write transfer acknowledgement
84  --   IP2Bus_Error               -- IP to Bus error response
85  ------------------------------------------------------------------------------
86
87  entity user_logic is
88    generic
89    (
90      -- ADD USER GENERICS BELOW THIS LINE ---------------
91      --USER generics added here
92      -- ADD USER GENERICS ABOVE THIS LINE ---------------
93
94      -- DO NOT EDIT BELOW THIS LINE ---------------------
95      -- Bus protocol parameters, do not add to or delete
96      C_NUM_REG                     : integer              := 4;
97      C_SLV_DWIDTH                  : integer              := 32
98      -- DO NOT EDIT ABOVE THIS LINE ---------------------
99    );
100   port
101   (
102     -- ADD USER PORTS BELOW THIS LINE ------------------
103     --USER ports added here
104     myinterrupt : out std_logic;
105     -- ADD USER PORTS ABOVE THIS LINE ------------------
106
107     -- DO NOT EDIT BELOW THIS LINE ---------------------
108     -- Bus protocol ports, do not add to or delete
109     Bus2IP_Clk                    : in  std_logic;
110     Bus2IP_Resetn                 : in  std_logic;
111     Bus2IP_Data                   : in  std_logic_vector(C_SLV_DWIDTH-1 downto 0);
112     Bus2IP_BE                     : in  std_logic_vector(C_SLV_DWIDTH/8-1 downto 0);
113     Bus2IP_RdCE                   : in  std_logic_vector(C_NUM_REG-1 downto 0);
```

```
114       Bus2IP_WrCE                    : in  std_logic_vector(C_NUM_REG-1 downto 0);
115       IP2Bus_Data                    : out std_logic_vector(C_SLV_DWIDTH-1 downto 0);
116       IP2Bus_RdAck                   : out std_logic;
117       IP2Bus_WrAck                   : out std_logic;
118       IP2Bus_Error                   : out std_logic
119       -- DO NOT EDIT ABOVE THIS LINE ---------------------
120     );
121
122     attribute MAX_FANOUT : string;
123     attribute SIGIS : string;
124
125     attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
126     attribute SIGIS of Bus2IP_Resetn : signal is "RST";
127
128   end entity user_logic;
129
130   ------------------------------------------------------------------------------
131   -- Architecture section
132   ------------------------------------------------------------------------------
133
134   architecture IMP of user_logic is
135
136     --USER signal declarations added here, as needed for user logic
137
138     ---------------------------------------
139     -- Signals for user logic slave model s/w accessible register example
140     ---------------------------------------
141     signal slv_reg0                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
142     signal slv_reg1                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
143     signal slv_reg2                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
144     signal slv_reg3                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
145
146     signal slv_reg_write_sel              : std_logic_vector(3 downto 0);
147     signal slv_reg_read_sel               : std_logic_vector(3 downto 0);
148     signal slv_ip2bus_data                : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
149     signal slv_read_ack                   : std_logic;
150     signal slv_write_ack                  : std_logic;
151
152   begin
153
154     --USER logic implementation added here
155
156     -- slv_reg0 = counter
157     -- slv_reg1 = delay_number
158     -- slv_reg2 = control register
159     -- slv_reg3 = unused for now
160
161
162
163
164
165     ---------------------------------------
166     -- Example code to read/write user logic slave model s/w accessible registers
167     --
168     -- Note:
169     -- The example code presented here is to show you one way of reading/writing
170     -- software accessible registers implemented in the user logic slave model.
```

```
171     -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
172     -- to one software accessible register by the top level template. For example,
173     -- if you have four 32 bit software accessible registers in the user logic,
174     -- you are basically operating on the following memory mapped registers:
175     --
176     --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
177     --                     "1000"   C_BASEADDR + 0x0
178     --                     "0100"   C_BASEADDR + 0x4
179     --                     "0010"   C_BASEADDR + 0x8
180     --                     "0001"   C_BASEADDR + 0xC
181     --
182     ---------------------------------------------
183     slv_reg_write_sel <= Bus2IP_WrCE(3 downto 0);
184     slv_reg_read_sel  <= Bus2IP_RdCE(3 downto 0);
185     slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or
        Bus2IP_WrCE(3);
186     slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or
        Bus2IP_RdCE(3);
187
188     -- implement slave model software accessible register(s)
189     SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
190     begin
191
192       if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
193         if Bus2IP_Resetn = '0' then
194           --slv_reg0 <= (others => '0');
195           slv_reg0 <= (others => '1');  -- counter resets to FF FF FF FF
196
197           slv_reg1 <= (others => '0');  -- delay resets to 00 00 00 00
198           slv_reg2 <= (others => '0');  -- control disables interrupts, does not load
        delay, and no decrement.
199           slv_reg3 <= (others => '0');  -- register to store what the interrupt should
        be. (only use LSB)
200         else
201           case slv_reg_write_sel is
202             when "1000" =>
203               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
204                 if ( Bus2IP_BE(byte_index) = '1' ) then
205                   slv_reg0(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
        *8+7 downto byte_index*8);
206                 end if;
207               end loop;
208             when "0100" =>
209               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
210                 if ( Bus2IP_BE(byte_index) = '1' ) then
211                   slv_reg1(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
        *8+7 downto byte_index*8);
212                 end if;
213               end loop;
214             when "0010" =>
215               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
216                 if ( Bus2IP_BE(byte_index) = '1' ) then
217                   slv_reg2(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
        *8+7 downto byte_index*8);
218                 end if;
219               end loop;
220             when "0001" =>
```

```
221               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
222                 if ( Bus2IP_BE(byte_index) = '1' ) then
223                   slv_reg3(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
      *8+7 downto byte_index*8);
224                 end if;
225               end loop;
226             when others =>
227
228
      ------------------------------------------------------------------------------
      -----------
229             -- Begin custom code for our PIT Timer
230
231             -- decrement? or no? This is based on bit 0 of our control register
232               if(slv_reg2(0) = '1') then
233                 -- We allow it to decrement
234                 slv_reg0 <= slv_reg0 - 1;
235               else
236                 -- no decrementing allowed
237                 slv_reg0 <= slv_reg0;
238               end if;
239
240             -- What happens when we hit 0? This is based on but 2 of our control register
241               if(slv_reg0 = "00000000000000000000000000000000") then
242                 -- we either reload or nothing
243                 if(slv_reg2(2) = '1') then
244                   -- we reload!
245                   slv_reg0 <= slv_reg1;
246                 else
247                   -- we do NOT reload, nor do we continue ticking
248                   slv_reg0 <= (others=>'0');
249                 end if;
250               end if;
251
252             -- make an interrupt if we ever hit 1. This way we can hold at zero without
      generating interrupts
253             -- This is based on but 1 of our control register
254             -- Notice the interrupt is stored in the LSB of reg3. The real interrupt
      signal will map to this at the end.
255               if((slv_reg0 = "00000000000000000000000000000001") and (slv_reg2(1) = '1'
      )) then
256                 slv_reg3(0) <= '1';
257               else
258                 slv_reg3(0) <= '0';
259               end if;
260
261             -- A custom value that reg3 should always be held at for debugging purposes
      (except the LSB; that is the interrupt)
262               slv_reg3(31 downto 1) <= "1010101010101010101010101010101";
263
264               -- End custom code for our PIT Timer
265
      ------------------------------------------------------------------------------
      ---------------
266
267           end case;
268         end if;
```

```
269       end if;
270     end process SLAVE_REG_WRITE_PROC;
271
272     -- implement slave model software accessible register(s) read mux
273     SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1, slv_reg2,
      slv_reg3 ) is
274     begin
275
276       case slv_reg_read_sel is
277         when "1000" => slv_ip2bus_data <= slv_reg0;
278         when "0100" => slv_ip2bus_data <= slv_reg1;
279         when "0010" => slv_ip2bus_data <= slv_reg2;
280         when "0001" => slv_ip2bus_data <= slv_reg3;
281         when others => slv_ip2bus_data <= (others => '0');
282       end case;
283
284     end process SLAVE_REG_READ_PROC;
285
286     ------------------------------------------
287     -- Example code to drive IP to Bus signals
288     ------------------------------------------
289     IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
290                     (others => '0');
291
292     IP2Bus_WrAck <= slv_write_ack;
293     IP2Bus_RdAck <= slv_read_ack;
294     IP2Bus_Error <= '0';
295
296
      ------------------------------------------------------------------------------
      --
297     -- The interrupt port is always the LSB of slv reg 3 :)
298     myinterrupt <= slv_reg3(0);
299     --
300
      ------------------------------------------------------------------------------
      -----
301
302   end IMP;
303
```