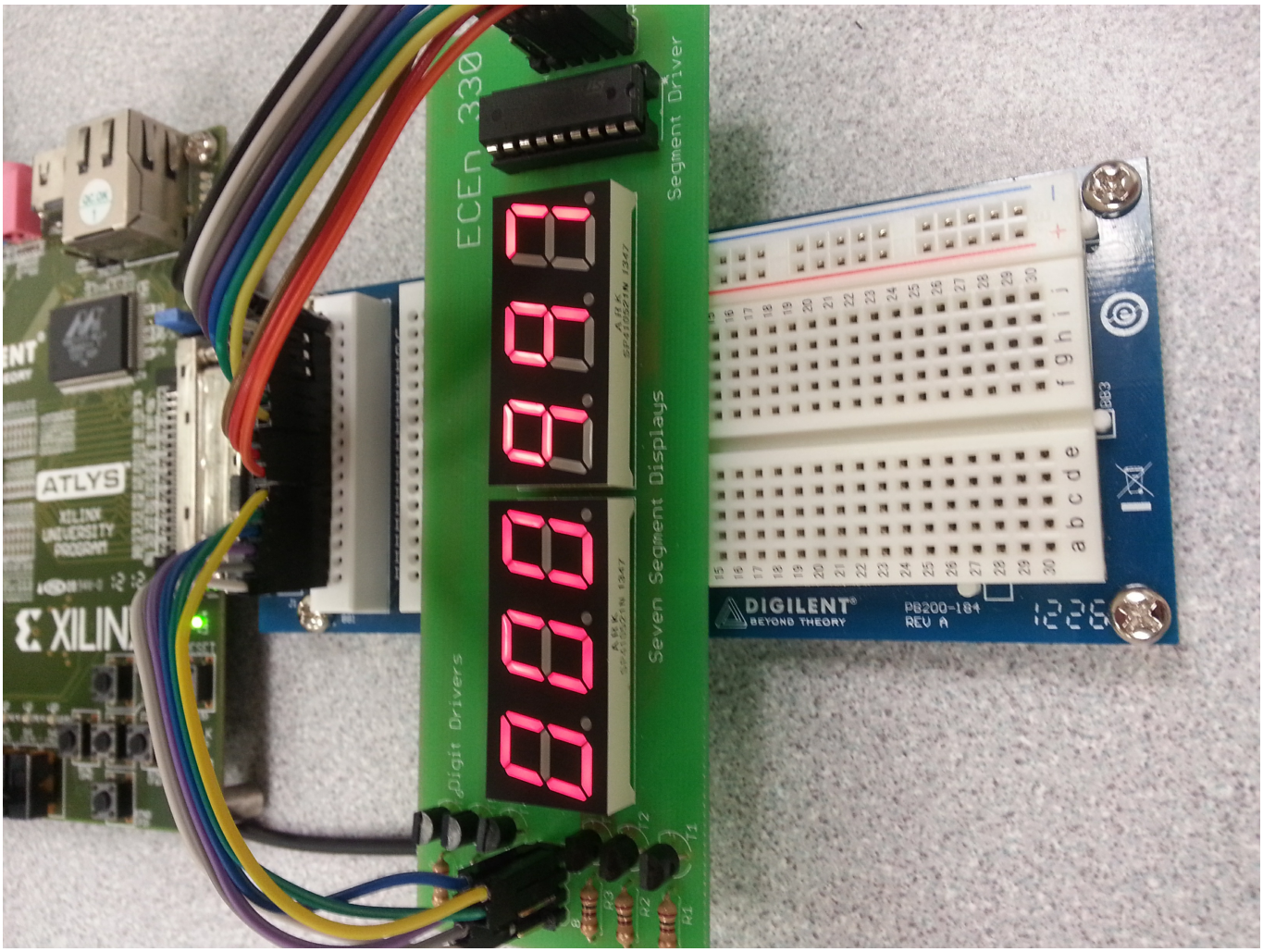
Lab 7 - Taylor Cowley and Andrew Okazaki

Adding a Score display

Hardware

An old 6 digit 7-segment LED display was found for former iterations of the ECEN 330 class. It had transistor drivers for the LEDs already on the board, which would make sending signals much easier



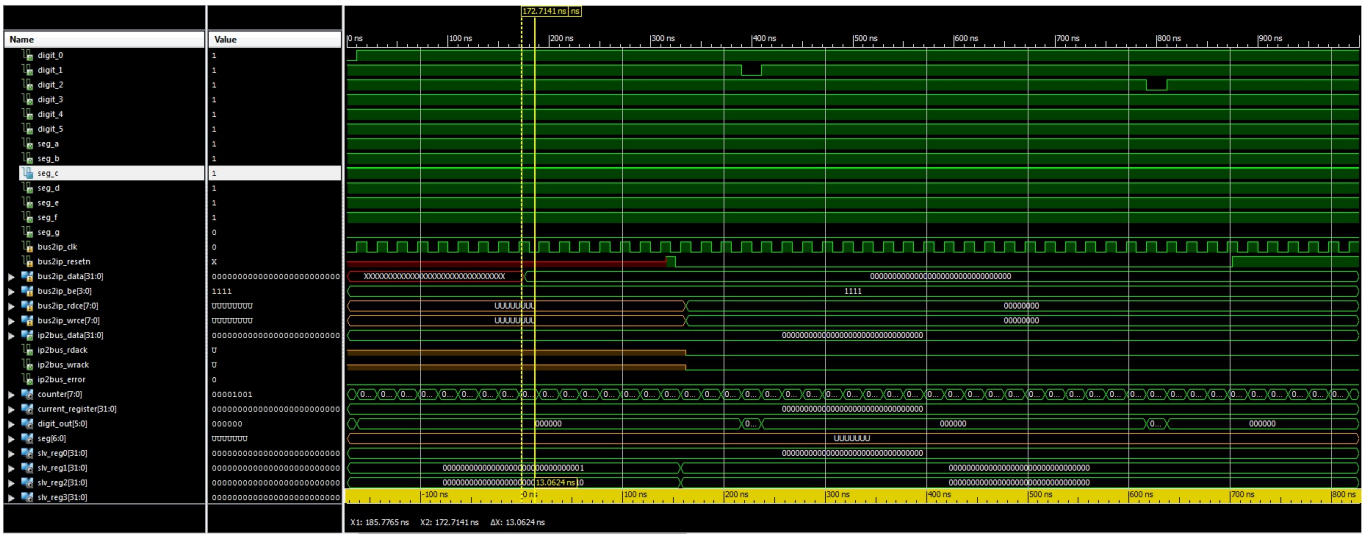
We made a custom IP in the Xilinx editor that had 8 registers- 6 to store the input for each digit, and 2 extra for debugging. This was a very similar process to making the PIT in the last lab

VHDL

VHDL code was added to make the proper 7 segment decoder- based on a certain input, the 7 outputs had to arrange to make a number. See Appendix for code. Also, a counter was added, and depending on the counter, a certain digit had its output sent. This is because the board had digit selectors and 7 segment selectors. It can display more than one digit at a time, but they will all display the same thing, depending on the 7 segment drivers. The solution to having different things displayed on each digit is to cycle faster than the human eye can distinguish through each of the 6 digits, outputting the proper data.

One problem we had was this counter was going too quickly, and the LEDs were never receiving enough power to output.

Some simulation was done to verify proper decoding by the VHDL



Software Interface

Because all logic for the display is handled in the VHDL, all the software had to do was write the value for each digit into a certain register. With the Xilinx automatically generating the software driver, all it took was

```
SCORE_DISPLAY_2_mWriteReg(XPAR_SCORE_DISPLAY_2_0_BASEADDR, i*SCORE_DISPLAY_DIGIT_OFFSET, number);
```

to make a number display on the LED display.

Bug Report

In our lab we ran into both hardware errors and software error. In our code it took us a while to find the output pin location in software. The next error we ran into in software was the when outputting a digit then moving to output the next digit we were cycling too fast. The speed of cycling caused artifacts from other digits and did not allow the number

to fully register, in addition to making the display very dim. Once we slowed the timer down it fixed the error that we were experiencing. The hardware error that we ran into was missing connecting the wires. This took a while to debug because at first we thought it was in the software but finally looked at the hardware and found our error.

```
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:              Tue Nov 15 17:51:08 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  library proc_common_v3_00_a;
```

```
  58  use proc_common_v3_00_a.proc_common_pkg.all;
  59
  60  -- DO NOT EDIT ABOVE THIS LINE --------------------
  61
  62  --USER libraries added here
  63
  64  ------------------------------------------------------------------------------
  65  -- Entity section
  66  ------------------------------------------------------------------------------
  67  -- Definition of Generics:
  68  --   C_NUM_REG                    -- Number of software accessible registers
  69  --   C_SLV_DWIDTH                 -- Slave interface data bus width
  70  --
  71  -- Definition of Ports:
  72  --   Bus2IP_Clk                   -- Bus to IP clock
  73  --   Bus2IP_Resetn                -- Bus to IP reset
  74  --   Bus2IP_Data                  -- Bus to IP data bus
  75  --   Bus2IP_BE                    -- Bus to IP byte enables
  76  --   Bus2IP_RdCE                  -- Bus to IP read chip enable
  77  --   Bus2IP_WrCE                  -- Bus to IP write chip enable
  78  --   IP2Bus_Data                  -- IP to Bus data bus
  79  --   IP2Bus_RdAck                 -- IP to Bus read transfer acknowledgement
  80  --   IP2Bus_WrAck                 -- IP to Bus write transfer acknowledgement
  81  --   IP2Bus_Error                 -- IP to Bus error response
  82  ------------------------------------------------------------------------------
  83
  84  entity user_logic is
  85    generic
  86    (
  87      -- ADD USER GENERICS BELOW THIS LINE ---------------
  88      --USER generics added here
  89      -- ADD USER GENERICS ABOVE THIS LINE ---------------
  90
  91      -- DO NOT EDIT BELOW THIS LINE ----------------------
  92      -- Bus protocol parameters, do not add to or delete
  93      C_NUM_REG                     : integer              := 8;
  94      C_SLV_DWIDTH                  : integer              := 32
  95      -- DO NOT EDIT ABOVE THIS LINE ----------------------
  96    );
  97    port
  98    (
  99      -- ADD USER PORTS BELOW THIS LINE ------------------
 100      --USER ports added here
 101      digit_0 : out std_logic;
 102      digit_1 : out std_logic;
 103      digit_2 : out std_logic;
 104      digit_3 : out std_logic;
 105      digit_4 : out std_logic;
 106      digit_5 : out std_logic;
 107
 108      seg_a : out std_logic;
 109      seg_b : out std_logic;
 110      seg_c : out std_logic;
 111      seg_d : out std_logic;
 112      seg_e : out std_logic;
 113      seg_f : out std_logic;
 114      seg_g : out std_logic;
```

```
115       -- ADD USER PORTS ABOVE THIS LINE ------------------
116
117       -- DO NOT EDIT BELOW THIS LINE ---------------------
118       -- Bus protocol ports, do not add to or delete
119       Bus2IP_Clk                     : in  std_logic;
120       Bus2IP_Resetn                  : in  std_logic;
121       Bus2IP_Data                    : in  std_logic_vector(C_SLV_DWIDTH-1 downto 0);
122       Bus2IP_BE                      : in  std_logic_vector(C_SLV_DWIDTH/8-1 downto 0);
123       Bus2IP_RdCE                    : in  std_logic_vector(C_NUM_REG-1 downto 0);
124       Bus2IP_WrCE                    : in  std_logic_vector(C_NUM_REG-1 downto 0);
125       IP2Bus_Data                    : out std_logic_vector(C_SLV_DWIDTH-1 downto 0);
126       IP2Bus_RdAck                   : out std_logic;
127       IP2Bus_WrAck                   : out std_logic;
128       IP2Bus_Error                   : out std_logic
129       -- DO NOT EDIT ABOVE THIS LINE ---------------------
130     );
131
132     attribute MAX_FANOUT : string;
133     attribute SIGIS : string;
134
135     attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
136     attribute SIGIS of Bus2IP_Resetn : signal is "RST";
137
138   end entity user_logic;
139
140   ------------------------------------------------------------------------------
141   -- Architecture section
142   ------------------------------------------------------------------------------
143
144   architecture IMP of user_logic is
145
146     --USER signal declarations added here, as needed for user logic
147
148     ---------------------------------------
149     -- Signals for user logic slave model s/w accessible register example
150     ---------------------------------------
151     signal slv_reg0                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
152     signal slv_reg1                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
153     signal slv_reg2                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
154     signal slv_reg3                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
155     signal slv_reg4                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
156     signal slv_reg5                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
157     signal slv_reg6                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
158     signal slv_reg7                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
159     signal slv_reg_write_sel              : std_logic_vector(7 downto 0);
160     signal slv_reg_read_sel               : std_logic_vector(7 downto 0);
161     signal slv_ip2bus_data                : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
162     signal slv_read_ack                   : std_logic;
163     signal slv_write_ack                  : std_logic;
164
165     signal counter: std_logic_vector(15 downto 0) := "0000000000000000";  -- should work
166     signal current_register: std_logic_vector(31 downto 0);
167     signal digit_out: std_logic_vector(5 downto 0);
168     signal seg: std_logic_vector(6 downto 0);
169   begin
170
171     ------------------------------------------------------------------------
```

```
172     -- Begin logic required for 7 segment display
173
174       digit_0 <= digit_out(0);
175       digit_1 <= digit_out(1);
176       digit_2 <= digit_out(2);
177       digit_3 <= digit_out(3);
178       digit_4 <= digit_out(4);
179       digit_5 <= digit_out(5);
180
181       seg_a <=  seg(6);
182       seg_b <=  seg(5);
183       seg_c <=  seg(4);
184       seg_d <=  seg(3);
185       seg_e <=  seg(2);
186       seg_f <=  seg(1);
187       seg_g <=  seg(0);
188
189   --current_register <= slv_reg0;
190     with counter(15 downto 13) select current_register <=
191       slv_reg0 when "000",
192       slv_reg1 when "001",
193       slv_reg2 when "010",
194       slv_reg3 when "011",
195       slv_reg4 when "100",
196       slv_reg5 when "101",
197       slv_reg6 when others; -- Error, should never happen
198
199     -- Digit out is 1-hot encoded
200     -- (well, not really, but if the digits are all different it is)
201     with counter(15 downto 13) select digit_out <=
202       "000001" when "000",  -- Digit 0
203       "000010" when "001",  -- Digit 1
204       "000100" when "010",  -- Digit 2
205       "001000" when "011",  -- Digit 3
206       "010000" when "100",  -- Digit 4
207       "100000" when "101",  -- Digit 5
208       "000000" when others; -- This should never happen
209   --    digit_out <= "111000";
210
211
212     -- The binary is arranged in
213     -- A B C D E F G in seg
214     -- In current_register it is whatever is in the register.
215     --seg <= current_register(6 downto 0);  -- with 2^7, 64 different combinations.
216   with current_register select seg <=
217     "1111110" when "00000000000000000000000000000000",  -- 0
218     "0110000" when "00000000000000000000000000000001",  -- 1
219     "1101101" when "00000000000000000000000000000010",  -- 2
220     "1111001" when "00000000000000000000000000000011",  -- 3
221     "0110011" when "00000000000000000000000000000100",  -- 4
222     "1011011" when "00000000000000000000000000000101",  -- 5
223     "1011111" when "00000000000000000000000000000110",  -- 6
224     "1110000" when "00000000000000000000000000000111",  -- 7
225     "1111111" when "00000000000000000000000000001000",  -- 8
226     "1110011" when "00000000000000000000000000001001",  -- 9
227
228     "1110111" when "00000000000000000000000000001010",  -- a
```

```
229     "0011111" when "00000000000000000000000000001011",  -- b
230     "0001101" when "00000000000000000000000000001100",  -- c
231     "0111101" when "00000000000000000000000000001101",  -- d
232     "1001111" when "00000000000000000000000000001110",  -- e
233     "1000111" when "00000000000000000000000000001111",  -- f
234     "1111011" when "00000000000000000000000000010000",  -- g
235   ---------------------- 11 is missing
236     "0010111" when "00000000000000000000000000010010",  -- h
237     "0110000" when "00000000000000000000000000010011",  -- i
238     "0111100" when "00000000000000000000000000010100",  -- j
239     "0010111" when "00000000000000000000000000010101",  -- k
240     "0001110" when "00000000000000000000000000010110",  -- l
241     "0010101" when "00000000000000000000000000010111",  -- m
242     "0010110" when "00000000000000000000000000011000",  -- n
243     "1111110" when "00000000000000000000000000011001",  -- o
244     "1100111" when "00000000000000000000000000011010",  -- p
245     "0011101" when "00000000000000000000000000011011",  -- q
246     "0000101" when "00000000000000000000000000011100",  -- r
247     "1011011" when "00000000000000000000000000011101",  -- s
248     "1000110" when "00000000000000000000000000011110",  -- t
249     "0011100" when "00000000000000000000000000011111",  -- u
250     "0011100" when "00000000000000000000000000100000",  -- v
251     "0011100" when "00000000000000000000000000100001",  -- w
252     "0110111" when "00000000000000000000000000100010",  -- x
253     "0110011" when "00000000000000000000000000100011",  -- y
254     "1101101" when "00000000000000000000000000100100",  -- z
255
256     "0000001" when "00000000000000000000000000100101", -- _
257     "0000000" when "11111111111111111111111111111111", -- ' '
258     "0111111" when others; -- displays a '-' on error
259
260
261     process(Bus2IP_Clk)
262     begin
263     --finish the clock ticking.
264       if(Bus2IP_Clk'EVENT and Bus2IP_Clk = '1') then
265         --if(counter = "101") then
266           --counter <= "000";
267         --else
268           --counter <= counter + 1;
269         --end if;
270         counter <= counter + 1;
271       end if;
272     end process;
273
274
275     -- End logic required for 7 segment display
276     ------------------------------------------------------------------------
277
278
279
280     ------------------------------------------
281     -- Example code to read/write user logic slave model s/w accessible registers
282     --
283     -- Note:
284     -- The example code presented here is to show you one way of reading/writing
285     -- software accessible registers implemented in the user logic slave model.
```

```
286     -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
287     -- to one software accessible register by the top level template. For example,
288     -- if you have four 32 bit software accessible registers in the user logic,
289     -- you are basically operating on the following memory mapped registers:
290     --
291     --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
292     --                     "1000"   C_BASEADDR + 0x0
293     --                     "0100"   C_BASEADDR + 0x4
294     --                     "0010"   C_BASEADDR + 0x8
295     --                     "0001"   C_BASEADDR + 0xC
296     --
297     ---------------------------------------------
298     slv_reg_write_sel <= Bus2IP_WrCE(7 downto 0);
299     slv_reg_read_sel  <= Bus2IP_RdCE(7 downto 0);
300     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);
301     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);
302
303     -- implement slave model software accessible register(s)
304     SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
305     begin
306
307       if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
308         if Bus2IP_Resetn = '0' then
309           slv_reg0 <= (others => '0');
310           slv_reg1 <= (others => '0');
311           slv_reg2 <= (others => '0');
312           slv_reg3 <= (others => '0');
313           slv_reg4 <= (others => '0');
314           slv_reg5 <= (others => '0');
315           slv_reg6 <= (others => '0');
316           slv_reg7 <= (others => '0');
317         else
318           case slv_reg_write_sel is
319             when "10000000" =>
320               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
321                 if ( Bus2IP_BE(byte_index) = '1' ) then
322                   slv_reg0(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
        *8+7 downto byte_index*8);
323                 end if;
324               end loop;
325             when "01000000" =>
326               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
327                 if ( Bus2IP_BE(byte_index) = '1' ) then
328                   slv_reg1(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
        *8+7 downto byte_index*8);
329                 end if;
330               end loop;
331             when "00100000" =>
332               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
333                 if ( Bus2IP_BE(byte_index) = '1' ) then
334                   slv_reg2(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
        *8+7 downto byte_index*8);
335                 end if;
336               end loop;
337             when "00010000" =>
```

```
338               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
339                 if ( Bus2IP_BE(byte_index) = '1' ) then
340                   slv_reg3(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
      *8+7 downto byte_index*8);
341                 end if;
342               end loop;
343             when "00001000" =>
344               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
345                 if ( Bus2IP_BE(byte_index) = '1' ) then
346                   slv_reg4(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
      *8+7 downto byte_index*8);
347                 end if;
348               end loop;
349             when "00000100" =>
350               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
351                 if ( Bus2IP_BE(byte_index) = '1' ) then
352                   slv_reg5(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
      *8+7 downto byte_index*8);
353                 end if;
354               end loop;
355             when "00000010" =>
356               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
357                 if ( Bus2IP_BE(byte_index) = '1' ) then
358                   slv_reg6(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
      *8+7 downto byte_index*8);
359                 end if;
360               end loop;
361             when "00000001" =>
362               for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
363                 if ( Bus2IP_BE(byte_index) = '1' ) then
364                   slv_reg7(byte_index*8+7 downto byte_index*8) <= Bus2IP_Data(byte_index
      *8+7 downto byte_index*8);
365                 end if;
366               end loop;
367             when others => null;
368           end case;
369         end if;
370       end if;
371
372     end process SLAVE_REG_WRITE_PROC;
373
374     -- implement slave model software accessible register(s) read mux
375     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
376     begin
377
378       case slv_reg_read_sel is
379         when "10000000" => slv_ip2bus_data <= slv_reg0;
380         when "01000000" => slv_ip2bus_data <= slv_reg1;
381         when "00100000" => slv_ip2bus_data <= slv_reg2;
382         when "00010000" => slv_ip2bus_data <= slv_reg3;
383         when "00001000" => slv_ip2bus_data <= slv_reg4;
384         when "00000100" => slv_ip2bus_data <= slv_reg5;
385         when "00000010" => slv_ip2bus_data <= slv_reg6;
386         when "00000001" => slv_ip2bus_data <= slv_reg7;
387         when others => slv_ip2bus_data <= (others => '0');
388       end case;
```

```
389
390     end process SLAVE_REG_READ_PROC;
391
392     ------------------------------------------
393     -- Example code to drive IP to Bus signals
394     ------------------------------------------
395     IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
396                     (others => '0');
397
398     IP2Bus_WrAck <= slv_write_ack;
399     IP2Bus_RdAck <= slv_read_ack;
400     IP2Bus_Error <= '0';
401
402   end IMP;
403
```