```
1           ----------------------------------------------------------------------------------
2         -- Company:
3         -- Engineer:
4         --
5         -- Create Date:    08:54:15 10/30/2009
6         -- Design Name:
7         -- Module Name:    emul_top - Behavioral
8         -- Project Name:
9         -- Target Devices:
10        -- Tool versions:
11        -- Description:
12        --
13        -- Dependencies:
14        --
15        -- Revision:
16        -- Revision 0.01 - File Created
17        -- Additional Comments:
18        --
19        ----------------------------------------------------------------------------------
20        library IEEE;
21        use IEEE.STD_LOGIC_1164.ALL;
22        use IEEE.STD_LOGIC_ARITH.ALL;
23        use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25        ---- Uncomment the following library declaration if instantiating
26        ---- any Xilinx primitives in this code.
27        --library UNISIM;
28        --use UNISIM.VComponents.all;
29
30        entity emul_top is
31            Port ( BTN : in  STD_LOGIC_VECTOR (3 downto 0);
32                   SW : in  STD_LOGIC_VECTOR (7 downto 0);
33                   CLK50MHZ : in  STD_LOGIC;
34                   LED : out  STD_LOGIC_VECTOR (7 downto 0);
35                   seg_out : out  STD_LOGIC_VECTOR (7 downto 0);
36                   digit : out  STD_LOGIC_VECTOR (3 downto 0);
37                   VGA_R, VGA_G, VGA_B : out  STD_LOGIC;
38                   VS, HS : out std_logic );
39
40        end emul_top;
41
42        architecture Behavioral of emul_top is
43         constant MAXFCN : integer := 20 ;
44         constant MAXFCNINDEX   : std_logic_vector(7 downto 0) := CONV_STD_LOGIC_VECTOR(MAXFCN,
45
46         signal CLK    : std_logic ;
47         signal dispdrv_en      : std_logic ;
48         signal curr   : std_logic_vector(3 downto 0) ; --
49         signal cd     : std_logic_vector(1 downto 0) ; -- current driver
50         signal seg    : std_logic_vector(6 downto 0);           --
51         signal dp     : std_logic ;           --
52         signal DIN    : std_logic_vector(15 downto 0) ;            --
53         signal khz_count : std_logic_vector(15 downto 0);
54         signal khz_en : std_logic ;
55         signal tenhz_count  : std_logic_vector(6 downto 0);
56         signal tenhz_en  : std_logic ;
57         signal quad_count   : std_logic_vector(4 downto 0);
58         signal quad_en   : std_logic ;
59         signal fcn_index : std_logic_vector(7 downto 0);
60         signal count  :   std_logic_vector(7 downto 0);
61         signal CLK25MHZ  : std_logic ;
```

```
62          signal VGA_X: STD_LOGIC_VECTOR (9 downto 0);
63          signal VGA_Y: STD_LOGIC_VECTOR (9 downto 0);
64          signal DISP : std_logic ;
65          signal HEXVALUE : STD_LOGIC_VECTOR (3 downto 0);
66          signal CHARLINE : STD_LOGIC_VECTOR (2 downto 0);
67          signal CHARCOL : STD_LOGIC_VECTOR (2 downto 0);
68          signal HEXBEAM : STD_LOGIC;
69          signal HEXQ : STD_LOGIC;
70          signal RAMADDRB : STD_LOGIC_VECTOR (3 downto 0);
71          signal RAMDATAB : STD_LOGIC_VECTOR (3 downto 0);
72          -- ---------------------- traffic signals:
73          signal P2_GREEN, P2_RED, P3_GREEN, P3_RED, P3_YELLOW : STD_LOGIC;
74          signal L2_GREEN, L2_RED, L3_GREEN, L3_RED, L3_YELLOW : STD_LOGIC;
75          signal HEXHILIGHT : STD_LOGIC;
76          signal FRAME : STD_LOGIC;
77          -- ---------------------- stop-watch signals:
78          signal stopw_run : STD_LOGIC;
79          signal ms100_count, sec_count : STD_LOGIC_VECTOR (3 downto 0);
80          signal CE1s : STD_LOGIC;
81
82          type ram_type is array (31 downto 0) of std_logic_vector (3 downto 0);
83          signal RAM : ram_type;
84
85          type f_LED_type is array (0 to MAXFCN - 1) of STD_LOGIC_VECTOR (7 downto 0);
86          signal f_LED : f_LED_type ;
87
88          component vga_time is
89           port (CLK: in STD_LOGIC;
90                 RESET: in STD_LOGIC;
91                 BLANK: out STD_LOGIC;
92                 DISP: out STD_LOGIC;
93                 DISP_HO: out STD_LOGIC;
94                 DISP_VO: out STD_LOGIC;
95                 HS: out STD_LOGIC;
96                 LFO: out STD_LOGIC;
97                 VS: out STD_LOGIC;
98                 X: buffer STD_LOGIC_VECTOR (9 downto 0);
99                 Y: buffer STD_LOGIC_VECTOR (9 downto 0));
100         end component vga_time;
101
102         component hexchar is
103            Port ( HEXVALUE : in  STD_LOGIC_VECTOR (3 downto 0);
104                   CHARLINE : in  STD_LOGIC_VECTOR (2 downto 0);
105                   CHARCOL : in  STD_LOGIC_VECTOR (2 downto 0);
106                   DOUT : out  STD_LOGIC);
107         end component hexchar;
108
109         component traffic is
110          port (CLK: in STD_LOGIC;
111                RST: in STD_LOGIC;
112                WALKRQ: in STD_LOGIC;
113                FSM: out STD_LOGIC_VECTOR (2 downto 0);
114                L2_GREEN: out STD_LOGIC;
115                L2_RED: out STD_LOGIC;
116                L3_GREEN: out STD_LOGIC;
117                L3_RED: out STD_LOGIC;
118                L3_YELLOW: out STD_LOGIC);
119         end component traffic;
120
121
122        begin
```

```
123
124         -- generates a 1 kHz signal from a 50 MHz signal
125         process (CLK50MHZ)
126         begin
127         if CLK50MHZ'event and CLK50MHZ = '1' then
128            khz_count <= khz_count + 1 ;
129            if khz_count = 49999 then
130               khz_en <= '1' ;
131               khz_count <= (others => '0') ;
132            else
133               khz_en <= '0' ;
134            end if ;
135         end if ;
136         end process ;
137
138         -- generates a 10 Hz signal from a 1kHz signal
139         process (CLK50MHZ)
140         begin
141         if CLK50MHZ'event and CLK50MHZ = '1' then
142            if khz_en = '1' then
143               tenhz_count <= tenhz_count + 1 ;
144               if tenhz_count = 99 then
145                  tenhz_en <= '1' ;
146                  tenhz_count <= (others => '0') ;
147               else
148                  tenhz_en <= '0' ;
149               end if ;
150            else
151               tenhz_en <= '0' ;
152            end if ;
153         end if ;
154         end process ;
155
156         -- generates a 4 Hz signal from a 10Hz signal
157         process (CLK50MHZ)
158         begin
159         if CLK50MHZ'event and CLK50MHZ = '1' then
160            if tenhz_en = '1' then
161               quad_count <= quad_count + 1 ;
162         --    if quad_count = 3 then
163         --       quad_en <= '1' ;
164         --       quad_count <= (others => '0') ;
165         --    else
166         --       quad_en <= '0' ;
167         --    end if ;
168         -- else
169         --    quad_en <= '0' ;
170            end if ;
171         end if ;
172         end process ;
173
174         -- Function counter
175         process (BTN(0))
176         begin
177         if BTN(0)'event and BTN(0) = '1' then
178           if fcn_index < MAXFCNINDEX then
179             fcn_index <= fcn_index + 1 ;
180           else
181             fcn_index <= (others => '0');
182           end if;
183         end if;
```

```
184         end process;
185
186         CLK <= CLK50MHZ ;
187         dispdrv_en <= khz_en ;
188         DIN(7 downto 0) <= fcn_index;
189         --DIN(11 downto 8) <= tenhz_count (6 downto 3);
190         --DIN(15 downto 12) <= SW (3 downto 0);
191         -- LED <= SW;
192
193         process (CLK)
194         begin
195         if CLK'event and CLK = '1' then
196            if dispdrv_en = '1' then
197               cd(1 downto 0) <= cd(1 downto 0) + 1 ;
198            end if ;
199            case cd(1 downto 0) is
200               when "00" =>   curr <= DIN(3 downto 0) ;   digit <= "1110" ;
201               when "01" =>   curr <= DIN(7 downto 4) ;   digit <= "1101" ;
202               when "10" =>   curr <= DIN(11 downto 8) ;  digit <= "1011" ;
203               when others => curr <= DIN(15 downto 12) ; digit <= "0111" ;
204            end case ;
205         end if ;
206         end process ;
207
208         --      0
209         --     ---
210         --  5 |   | 1
211         --     ---   <- 6
212         --  4 |   | 2
213         --     ---
214         --      3
215
216           with curr SELect
217           seg <= "1111001" when "0001",   --1
218                  "0100100" when "0010",   --2
219                  "0110000" when "0011",   --3
220                  "0011001" when "0100",   --4
221                  "0010010" when "0101",   --5
222                  "0000010" when "0110",   --6
223                  "1111000" when "0111",   --7
224                  "0000000" when "1000",   --8
225                  "0010000" when "1001",   --9
226                  "0001000" when "1010",   --A
227                  "0000011" when "1011",   --b
228                  "1000110" when "1100",   --C
229                  "0100001" when "1101",   --d
230                  "0000110" when "1110",   --E
231                  "0001110" when "1111",   --F
232                  "1000000" when others;   --0
233
234           seg_out <= DP & seg;
235
236         -- XXXXXXXXXXXXXX FUNCTION 1 - Simple gates
237           f_LED(1)(7) <= SW(7) xor SW(6) xor SW(5);
238           f_LED(1)(6) <= SW(7) and SW(6) and SW(5);
239           f_LED(1)(5) <= SW(7) or  SW(6) or  SW(5);
240           f_LED(1)(4) <= quad_count(1);
241           f_LED(1)(3) <= SW(7) xor quad_count(1);
242           f_LED(1)(2) <= SW(7) and quad_count(1);
243           f_LED(1)(1) <= SW(7) or  quad_count(1);
244           f_LED(1)(0) <= quad_count(1);
```

```
245
246
247         -- XXXXXXXXXXXXXX FUNCTION 2 - Mux
248           f_LED(2)(7) <= SW(7) when SW(3 downto 2) = "11" else
249                          SW(6) when SW(3 downto 2) = "10" else
250                          SW(5) when SW(3 downto 2) = "01" else
251                          SW(4) ;
252
253           f_LED(2)(6) <= quad_count(1) when SW(3 downto 2) = "11" else
254                          quad_count(0) when SW(3 downto 2) = "10" else
255                          tenhz_count(6) when SW(3 downto 2) = "01" else
256                          tenhz_count(5) ;
257
258           f_LED(2)(5) <= '0';
259           f_LED(2)(4) <= '0';
260           f_LED(2)(3) <= quad_count(1);
261           f_LED(2)(2) <= quad_count(0);
262           f_LED(2)(1) <= tenhz_count(6);
263           f_LED(2)(0) <= tenhz_count(5);
264
265         -- XXXXXXXXXXXXXX FUNCTION 3 - Decoder
266         -- SW(7..5) - DIN
267         -- SW(4) -  CLR
268         -- LED(7..0)
269           process(SW(7 downto 4))
270           begin
271               if ( SW(4) = '0') then
272                  f_LED(3) <= "00000000";
273               else
274                  case SW(7 downto 5) is
275                     when "000" => f_LED(3) <= "00000001";
276                     when "001" => f_LED(3) <= "00000010";
277                     when "010" => f_LED(3) <= "00000100";
278                     when "011" => f_LED(3) <= "00001000";
279                     when "100" => f_LED(3) <= "00010000";
280                     when "101" => f_LED(3) <= "00100000";
281                     when "110" => f_LED(3) <= "01000000";
282                     when "111" => f_LED(3) <= "10000000";
283                     when others => f_LED(3) <= "00000000";
284                  end case;
285               end if;
286           end process;
287
288         -- XXXXXXXXXXXXXX FUNCTION 4 - priority encoder
289         -- SW(7..0) - DIN
290         -- LED(2..0) - Q
291
292            f_LED(4)(2 downto 0) <= "000" when SW(0) = '1' else
293           "001" when SW(1) = '1' else
294           "010" when SW(2) = '1' else
295           "011" when SW(3) = '1' else
296           "100" when SW(4) = '1' else
297           "101" when SW(5) = '1' else
298           "110" when SW(6) = '1' else
299           "111"  ;
300
301
302         -- XXXXXXXXXXXXXX FUNCTION 5 - Accumulator
303
304         -- BTN(3) - CLR
305         -- BTN(2) - CLK
```

```
306         -- SW(7) - CE
307         -- SW(3..0) - DIN
308         -- LED(7..0)- Q
309            process (BTN(3), BTN(2))
310            begin
311            if BTN(3) = '1' then
312             f_LED(5)(7 downto 0) <= (others => '0');
313            elsif BTN(2)'event and BTN(2) = '1' then  -- Rising edeg
314             if SW(7) = '1' then
315              f_LED(5)(7 downto 0) <= f_LED(5)(7 downto 0) + sw(3 downto 0);
316             end if;
317            end if;
318            end process;
319
320
321         -- XXXXXXXXXXXXXX FUNCTION 6 - Barrel Shifter
322         -- SW(5..4)- SHIFT
323         -- SW(3..0)- DIN
324         -- LED(3..0)- Q
325          process (SW(5 downto 4),SW(3 downto 0))
326           begin case SW(5 downto 4) is
327            when "00" =>       -- Shift by 0
328             f_LED(6)(3 downto 0) <= SW(3 downto 0);
329            when "01" =>       -- Shift by 1
330            f_LED(6)(1) <= SW(0);
331            f_LED(6)(2) <= SW(1);
332            f_LED(6)(3) <= SW(2);
333            f_LED(6)(0) <= SW(3);
334            when "10" =>       -- Shift by 2
335            f_LED(6)(2) <= SW(0);
336            f_LED(6)(3) <= SW(1);
337            f_LED(6)(0) <= SW(2);
338            f_LED(6)(1) <= SW(3);
339            when "11" =>       -- Shift by 3
340            f_LED(6)(3) <= SW(0);
341            f_LED(6)(0) <= SW(1);
342            f_LED(6)(1) <= SW(2);
343            f_LED(6)(2) <= SW(3);
344            when others =>
345            f_LED(6)(3 downto 0) <= SW(3 downto 0);
346           end case;
347          end process;
348
349
350         -- XXXXXXXXXXXXXX FUNCTION 7 - Comparator
351         -- SW(7..4) = NUMBER A
352         -- SW(3..0) = NUMBER B
353         -- LED(7) -- A > B
354         -- LED(6) -- A = B
355         -- LED(5) -- A < B
356
357         f_LED(7)(7) <= '1' when SW(7 downto 4) > SW(3 downto 0) else -- GT (Greater Than)
358                        '0';
359         f_LED(7)(6) <= '1' when SW(7 downto 4) = SW(3 downto 0) else -- EQU (EQUal to)
360                        '0';
361         f_LED(7)(5) <= '1' when SW(7 downto 4) < SW(3 downto 0) else  -- LT (Less Than)
362                        '0';
363
364         -- XXXXXXXXXXXXXX FUNCTION 8 - FF, FF_1
365         -- BTN(3) - CLR
366         -- BTN(2) - CLK
```

```
367         -- SW(7)  - CE
368         -- SW(3..0) - DIN
369         -- LED(3..0)- Q
370
371           process (BTN(3), BTN(2))
372           begin
373             if BTN(3) = '1' then
374               f_LED(8)(3 downto 0) <= (others => '0');
375             elsif BTN(2)'event and BTN(2) = '1' then    -- Rising edge
376               if SW(7) = '1' then
377                  f_LED(8)(3 downto 0) <= SW(3 downto 0);
378               end if;
379             end if;
380           end process;
381
382           process (BTN(3), BTN(2))
383           begin
384             if BTN(3) = '1' then
385               f_LED(8)(7 downto 4) <= (others => '0');
386             elsif BTN(2)'event and BTN(2) = '0' then   -- Falling edge
387               if SW(7) = '1' then
388                  f_LED(8)(7 downto 4) <= SW(3 downto 0);
389               end if;
390             end if;
391           end process;
392         -- XXXXXXXXXXXXXX FUNCTION 9 - Latch
393         -- SW(7)-- GATE
394         -- SW(3..0)-- DIN
395         -- Q -- LED(3..0)
396
397          process (SW(7), SW(3 downto 0))
398         begin
399            if SW(7)='1' then  --GATE active High
400               f_LED(9)(3 downto 0) <= SW(3 downto 0);
401            end if;
402         end process;
403
404
405         -- XXXXXXXXXXXXXX FUNCTION 10 - Shift register
406         -- BTN(3)- CLR
407         -- BTN(2)- CLK
408         -- SW(7) - CE
409         -- SW(6) - SLI
410         -- SW(5) - LOAD
411         -- SW(3..0)- DIN
412         -- LED(7..0)- Q
413
414          process (BTN(3), BTN(2))
415          begin
416           if BTN(3) = '1' then
417            f_LED(10)(7 downto 0) <= (others => '0');
418             elsif BTN(2)'event and BTN(2) = '1' then  -- Rising edge
419              if SW(7) = '1' then
420               if SW(5) = '1' then
421               f_LED(10)(7 downto 0) <= f_LED(10)(7 downto 4) & SW(3 downto 0);
422               else
423               f_LED(10)(7 downto 0) <= SW(6) & f_LED(10)(7 downto 1);
424               end if;
425              end if;
426           end if;
427          end process;
```

```
428         -- XXXXXXXXXXXXXX FUNCTION 11 - COUNTER
429
430         -- BTN(3)- CLR
431         -- BTN(2)- CLK
432         -- SW(7) - CE
433         -- SW(6) - DIR
434         -- SW(5) - LOAD
435         -- LED(7..0)- Q
436            process (BTN(3), BTN(2))
437            begin
438            if BTN(3)= '1' then  -- Active high reset
439             count <=(others => '0');
440            elsif BTN(2)'event and BTN(2) = '1' then  -- Rising edge
441             if SW(7)= '1' then  -- CE
442              if SW(5) = '1' then  -- LOAD
443               count(3 downto 0) <= SW(3 downto 0);
444              else
445               if SW(6) = '1' then  -- DIR
446                count <= count + 1 ;
447               else
448                count <= count - 1 ;
449               end if;
450              end if;
451             end if ;
452            end if ;
453            end process ;
454
455             f_LED(11) <= count;
456
457         -- XXXXXXXXXXXXXX FUNCTION 12 - Memory
458         -- (fcn_index = 12) - WRITE ENABLE
459         -- BTN(2)- CLK
460         -- SW(7 .. 4)- Port A ADDRESS
461         -- SW(3 .. 0)- Port A DATA Input
462         -- LED(3.. 0)- Port A DATA Output
463         --------------------------------
464         -- RAMADDRB(3..0) - Port B ADDRESS
465         -- RAMDATAB(3..0) - Port B DATA
466
467          process (BTN(2))    -- Port A
468          begin
469            if (BTN(2)'event and BTN(2)= '1') then
470             if (fcn_index = X"C") then   -- (fcn_index = 12) - WRITE ENABLE
471                RAM(conv_integer(SW(7 downto 4))) <= SW(3 downto 0);
472             end if;
473            end if;
474          end process;
475          f_LED(12)(3 downto 0) <= RAM(conv_integer(SW(7 downto 4)));
476          RAMDATAB <= RAM(conv_integer(RAMADDRB));
477
478          process (CLK50MHZ)  -- Generates 25MHz clock for vga_time
479          begin
480           if CLK50MHZ'event and CLK50MHZ = '1' then
481            CLK25MHZ <= not CLK25MHZ;
482           end if;
483          end process;
484
485          vga1: vga_time
486           port map (CLK => CLK25MHZ,
487                     RESET => BTN(3),
488                     BLANK => open,
```

```
489                     DISP => DISP,
490                     DISP_HO => open,
491                     DISP_VO => open,
492                     HS => HS,
493                     LFO => open,
494                     VS => VS,
495                     X => VGA_X,
496                     Y => VGA_Y);
497
498          hexrom: hexchar
499             port map(  HEXVALUE => HEXVALUE,
500                        CHARLINE => CHARLINE,
501                        CHARCOL => CHARCOL,
502                        DOUT => HEXQ);
503
504           CHARLINE <= VGA_Y(3 downto 1);
505           CHARCOL  <= VGA_X(3 downto 1);
506           RAMADDRB <= VGA_Y(7 downto 4);
507           HEXBEAM  <= '1' when HEXQ = '1' and VGA_X(9 downto 5) = "00011" and VGA_Y(9 downto 8)
        "00" else
508                       '0';
509           HEXVALUE <= RAMDATAB when VGA_X(4) = '1' else
510                       RAMADDRB;
511           HEXHILIGHT <= '1' when RAMADDRB = SW(7 downto 4) and VGA_X(9 downto 5) = "00011" and V
        (9 downto 8) = "00" else
512                         '0';
513
514           VGA_R <= DISP and ( HEXBEAM or (P3_RED and L3_RED) or (P2_RED and L2_RED) or (P3_YELLO
        and L3_YELLOW));
515           VGA_G <= DISP and ( ( HEXBEAM and not VGA_X(4)) or (P3_GREEN and L3_GREEN) or (P2_GREE
        and L2_GREEN) or (P3_YELLOW and L3_YELLOW));
516           VGA_B <= DISP and ( ( HEXBEAM xor HEXHILIGHT ) or FRAME);
517
518         -- XXXXXXXXXXXXXX FUNCTION 13 - DDR Register
519         -- BTN(3)- CLR
520         -- BTN(2)- CLK
521         -- SW(7) - CE
522         -- SW(3..0) - DIN
523         -- LED(3..0)- Q
524           f_LED(13)(3 downto 0) <= f_LED(8)(3 downto 0) when BTN(2)= '1' else
525                                    f_LED(8)(7 downto 4);
526
527         -- XXXXXXXXXXXXXX FUNCTION 14 - Debounce circuit
528         -- SW(0) - DIN
529         -- BTN(3) - CLR
530         -- BTN(2) - CLK
531         -- LED(0) - Q
532
533          process(BTN(2), BTN(3))
534         begin
535           if (BTN(3) = '1') then
536             f_LED(14)(1) <= '0';
537             f_LED(14)(2) <= '0';
538             f_LED(14)(3) <= '0';
539           elsif (BTN(2)'event and BTN(2) = '1') then
540             f_LED(14)(1) <= SW(0);
541             f_LED(14)(2) <= f_LED(14)(1) ;
542             f_LED(14)(3) <=f_LED(14)(2) ;
543           end if;
544         end process;
545
```

```
546         f_LED(14)(0) <= f_LED(14)(1) and f_LED(14)(2) and (not f_LED(14)(3) );
547
548         -- XXXXXXXXXXXXXX FUNCTION 15 - Traffic lights
549
550         -- BTN(3)   -  RESET
551         -- BTN(1)   -  WALK Request
552         -- LED(7)   -  road RED
553         -- LED(6)   -  road YELLOW
554         -- LED(5)   -  road GREEN
555         -- LED(4)   -  walker RED
556         -- LED(3)   -  walker GREEN
557         -- LED(2)   -  FSM(2)
558         -- LED(1)   -  FSM(1)
559         -- LED(0)   -  FSM(0)
560
561          trafficFSM: traffic
562           port map(CLK => quad_count(3),
563                 RST    => BTN(3),
564                 WALKRQ => BTN(1),
565                 FSM    => f_LED(15)(2 downto 0),
566                 L2_GREEN  => L2_GREEN ,
567                 L2_RED    => L2_RED   ,
568                 L3_GREEN  => L3_GREEN ,
569                 L3_RED    => L3_RED   ,
570                 L3_YELLOW => L3_YELLOW );
571
572         f_LED(15)(3) <= L2_GREEN;
573         f_LED(15)(4) <= L2_RED;
574         f_LED(15)(5) <= L3_GREEN;
575         f_LED(15)(7) <= L3_RED;
576         f_LED(15)(6) <= L3_YELLOW;
577
578          P3_RED     <= '1' when VGA_X = "010000----" and VGA_Y = "000100----" else
579                    '0';
580          P3_YELLOW  <= '1' when VGA_X = "010000----" and VGA_Y = "000101----" else
581                    '0';
582          P3_GREEN   <= '1' when VGA_X = "010000----" and VGA_Y = "000110----" else
583                    '0';
584          P2_RED     <= '1' when VGA_X = "010010----" and VGA_Y = "000100----" else
585                    '0';
586          P2_GREEN   <= '1' when VGA_X = "010010----" and VGA_Y = "000101----" else
587                    '0';
588          FRAME   <= '1' when (P3_RED or P3_YELLOW or P3_GREEN or P2_RED or P2_GREEN) = '1' and (
            VGA_X = "------0000" or VGA_Y = "------0000") else
589                     '0';
590
591         -- XXXXXXXXXXXXXX FUNCTION 16 - Stop watch timer
592         -- BTN(3)   -  RESET
593         -- BTN(2)   -  START
594         -- BTN(1)   -  STOP
595
596         -- Stop-watch control
597         process (CLK50MHZ)
598         begin
599         if CLK50MHZ'event and CLK50MHZ = '1' then
600          if BTN(1) = '1' or BTN(0) = '1' then  -- STOP button or "change function" button
601           stopw_run <= '0';
602          elsif BTN(2) = '1' then   -- RUN button
603           stopw_run <= '1';
604          end if;
605         end if ;
```

```
606         end process ;
607
608
609         -- 100 Miliseconds counter
610         process (CLK50MHZ)
611         begin
612         if CLK50MHZ'event and CLK50MHZ = '1' then
613          if BTN(3) = '1' then   -- RESET
614           ms100_count <= (others => '0') ;
615          elsif tenhz_en = '1' and stopw_run = '1' then
616               if ms100_count < 9 then
617                 ms100_count <= ms100_count + 1 ;
618               else
619                 ms100_count <= (others => '0') ;
620               end if;
621          end if ;
622         end if ;
623         end process ;
624
625         CE1s <= '1' when ms100_count = 9 and tenhz_en = '1' else
626                 '0';
627
628         -- Seconds counter
629         process (CLK50MHZ)
630         begin
631         if CLK50MHZ'event and CLK50MHZ = '1' then
632          if BTN(3) = '1' then   -- RESET
633           sec_count <= (others => '0') ;
634          elsif CE1s = '1' and stopw_run = '1' then
635               if sec_count < 9 then
636                 sec_count <= sec_count + 1 ;
637               else
638                 sec_count <= (others => '0') ;
639               end if;
640          end if ;
641         end if ;
642         end process ;
643
644         -- >>>>>>>>>>>>>>> DISPLAY DIGITS 3,2 :
645
646           process (fcn_index)
647           begin
648             case (CONV_INTEGER(fcn_index)) is
649               when 1 => DIN(15 downto 8) <= SW(7 downto 4) + SW(3 downto 0);
650               when 2 => DIN(15 downto 8) <= SW(7 downto 4) - SW(3 downto 0);
651               when 3 => DIN(15 downto 8) <= SW(7 downto 4) * SW(3 downto 0);
652               when 4 => DIN(11 downto 8) <= "0" & f_LED(4)(2 downto 0);
653               when 5 => DIN(15 downto 8) <= f_LED(5);
654               when 9 => DIN(11 downto 8) <= f_LED(9)(3 downto 0);
655               when 11 => DIN(15 downto 8) <= f_LED(11);
656               when 12 => DIN(15 downto 8) <=  x"0" & RAM(conv_integer(SW(7 downto 4)));
657               when 13 => DIN(15 downto 8) <=  x"0" & f_LED(13)(3 downto 0);
658               when 15 => DIN(15 downto 8) <=  "00000" & f_LED(15)(2 downto 0);
659               when 16 => DIN(15 downto 8) <= sec_count & ms100_count;
660               when others => DIN(15 downto 8) <= (others => '0');
661             end case;
662           end process;
663
664         -- -------- LATCH -------------------
665          process (f_LED, fcn_index)
666          begin
```

```
667           for I in 0 to 7 loop
668             LED(I) <= f_LED(CONV_INTEGER(fcn_index))(I);
669           end loop;
670         end process;
671
672
673       -- ::::: MULTIPLEXER ::::::::::::::::
674         process (f_LED, fcn_index)
675         begin
676           for I in 0 to 7 loop
677             LED(I) <= f_LED(CONV_INTEGER(fcn_index))(I);
678           end loop;
679         end process;
680
681
682       end Behavioral;
683
684
```