Nathan Roth
CE1911-021
2/5/2020
Dr.Meier

Lab Week 8 Lab Write Up

Abstract:
In this lab I created a game on the DE-10 LITE board. The game I created was wack-a-mole, I created it using multiple registers to control the state counter, the player score, and the random number that lights up the led. I had 40 states, the first 11 states displayed basic directions to the game. Then the from state 11 to state 15, a ready up message is displayed. After that a random led will light up for nine states and if the switch under the light is flipped up player one will get a point. Then another ready message will display for 3 states. After that a random led will light up for nine states and if the switch under the light is flipped up player two will get a point. Finally, both players scores will be displayed for 3 states. To generate a random number, I used the provided random number generator, and attached a clock switcher to a load signal that would act as load for the random number register, the switching clock speed will have the led stay light for different intervals. The number will be from 0 to 9 and that number will correspond to a led light above a switch, but this will only run during states 16 to 24 and 28 to 36. The player score register will keep track of score by using a load signal that will only go high when the same number switch and led are high. This prevents most of the cheating because if multiple switches are up the then the score will not increase. Although people could still cheat the game by flipping up one switch and leaving it, or if a light is on for too long you can score two points by leaving the switch up. A possible solution to that is by setting the switch to a clock switch and have the load only count on the rising edge of the switch. I did not implement this because I wanted to have a working game in time, and I had other labs I needed to focus on.

Instructions:
To play wack-a-mole, flip up the switch under the lit led to score a point. Do not leave the switch up because if the switch is left up and other switches are flipped up the score will not increase. If you want to reset the game press the lower button for around a second. The game will display more basic instructions, then it will tell player one to get ready. Then player one will get 9 chances to score, after that a second ready message will display alerting player two to get ready. Then player two will also get 9 chances to score. Finally, both scores will be displayed next to each other so you can determine a winner.

```
/*
*  Project:    Wack-a-mole
*  Entity:     wack
*  Author:     rothn@msoe.edu
*  Date:       27 JAN 2020
*  Purpose:
*      To create a circuit that plays wack-a-mole on a
*      DE-10 LITE board. The game will have 21 possible
*      states, allowing two players to play. The first will
*      go for 9 states, a point is scored when the user
*      flips the switch underneth the lit led when it
*      activates. After player one, the second player
*      will go for 9 states and the winner will be
*      displayed on the seven-segment displayed
*/

-- library
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.dseg7.all;
use work.ledstrip.all;
use work.clockdiv.all;


-- entity description
entity wack is
port(rst, clk: in std_logic;
     switches: in std_logic_vector(9 downto 0);
     leds: out std_logic_vector(9 downto 0);
     hex5: out std_logic_vector(7 downto 0);
     hex4: out std_logic_vector(7 downto 0);
     hex3: out std_logic_vector(7 downto 0);
     hex2: out std_logic_vector(7 downto 0);
     hex1: out std_logic_vector(7 downto 0);
     hex0: out std_logic_vector(7 downto 0));
end entity wack;


-- architecture description
architecture behavioral of wack is

  -- internal state counter
  signal q,d : integer range 0 to 39;

  -- player load and counter internal signals
  signal p1ld, p2ld: std_logic;
  signal p1scrd, p1scrq, p2scrd, p2scrq: integer range 0 to 9;

  -- random ld
  signal randld: integer range 0 to 1;
  signal less_slow_clk: std_logic;
  signal cnt2: integer range 0 to 25000000;
  signal rndCNT: std_logic_vector (25 downto 0);
  signal sq, sd: integer range 0 to 3;
  signal rnd_slow: std_logic;

  -- if the led should light up
  signal ledlight: integer range 0 to 10;

  -- declare other internal signals for 1-Hz clock
  signal slow_clk: std_logic;
  signal cnt: integer range 0 to 25000000;

```

```
64
65      -- random counter signals
66      signal rndmcount, randomreg: integer range 0 to 9;
67
68      -- assign DE10-Lite pins
69      -- see DE10-Lite Manual pages 24 - 29 for pins
70      attribute chip_pin : string;
71      attribute chip_pin of rst : signal is "A7"; -- key0
72      attribute chip_pin of clk : signal is "P11";
73      attribute chip_pin of leds : signal is
74        "B11,A11,D14,E14,C13,D13,B10,A10,A9,A8 ";
75      attribute chip_pin of switches: signal is
76        "F15,B14,A14,A13,B12,A12,C12,D12,C11,C10";
77      attribute chip_pin of hex5 : signal is
78        "L19,N20,N19,M20,N18,L18,K20,J20";
79      attribute chip_pin of hex4 : signal is
80        "F17,F20,F19,H19,J18,E19,E20,F18";
81      attribute chip_pin of hex3 : signal is
82        "D22,E17,D19,C20,C19,E21,E22,F21";
83      attribute chip_pin of hex2 : signal is
84        "A19,B22,C22,B21,A21,B19,A20,B20";
85      attribute chip_pin of hex1 : signal is
86        "A16,B17,A18,A17,B16,E18,D18,C18";
87      attribute chip_pin of hex0 : signal is
88        "D15,C17,D17,E16,C16,C15,E15,C14";
89
90
91    begin
92
93      -- next state logic states
94      d <= q+1 when q<39 else 0;
95
96
97      -- next state logic player 1 score
98      p1scrd <= p1scrq+1 when p1scrq < 9 else 9;
99
100     -- next state logic player 2 score
101     p2scrd <= p2scrq+1 when p2scrq < 9 else 9;
102
103
104      --register for states
105     reg: process(all) -- uses all inputs could possible affect the process
106     begin
107        if rising_edge(slow_clk) then
108           if rst = '0' then q <= 0;
109           else q <= d;
110           end if;
111        end if;
112     end process reg;
113
114
115     -- Player 1 score
116     p1: process(all) -- uses all inputs could possible affect the process
117      begin
118        if rising_edge(less_slow_clk) then
119           if rst = '0' then p1scrq <= 0;
120              elsif p1ld = '1' then p1scrq <= p1scrd;
121              elsif q = 0 then p1scrq <= 0;
122           end if;
123        end if;
124     end process p1;
125
126     -- rising edge of switch?
127
```

```
125     |
126     |   -- Player 2 score
127     |   p2: process(all) -- uses all inputs could possible affect the process
128     |    begin
129     |       if rising_edge(less_slow_clk) then
130     |          if rst = '0' then p2scrq <= 0;
131     |             elsif p2ld = '1' then p2scrq <= p2scrd;
132     |             elsif q = 0 then p2scrq <= 0;
133     |          end if;
134     |       end if;
135     |    end process p2;
136     |
137     |
138     |   -- load signal for the random number
139     |   randomLD: process(all)
140     |   begin
141     |    if rising_edge(rnd_slow) then
142     |       if rst = '0' then randld <= 0;
143     |          elsif randld = 0 then randld <= 1;
144     |          else randld <= 0;
145     |       end if;
146     |    end if;
147     |   end process randomLD;
148     |
149     |
150     |   -- random counter
151     |   rndm:process(all)
152     |   begin
153     |     if rising_edge(clk) then
154     |        if rst = '0' then rndmcount <= 0;
155     |          elsif rndmcount = 9 then rndmcount <= 0;
156     |             else rndmcount <= rndmcount + 1;
157     |         end if;
158     |     end if;
159     |   end process rndm;
160     |
161     |    -- process that grabs the random number when told by the go signal
162     |    -- note that humans are slow; if a fast clock is attached, this process will literally get
163     |    -- random values millions of times before the load is  1. Thus, the random number is really
164     |    -- captured when the load signal is 1
165     |   getrandom:process(all)
166     |   begin
167     |     if rising_edge(less_slow_clk) then
168     |       if randld = 1 then randomreg <= rndmcount;
169     |       end if;
170     |     end if;
171     |   end process;
172     |
173     |
174     |   -- multiplexer to check the leds should light up
175     |   with q select
176     |   ledlight <= randomreg when 16 to 24 | 28 to 36,
177     |               10 when others;
178     |
179     |
180     |   -- output logic for random number
181     |   with ledlight select
182     |   leds <= active_high_bit0 when 0,
183     |           active_high_bit1 when 1,
184     |           active_high_bit2 when 2,
185     |           active_high_bit3 when 3,
186     |           active_high_bit4 when 4,
187     |           active_high_bit5 when 5,
188     |           active_high_bit6 when 6,
```

```
188             active_high_bit6 when 6,
189             active_high_bit7 when 7,
190             active_high_bit8 when 8,
191             active_high_bit9 when 9,
192             active_high_blank when others;
193
194
195         -- p1 enable score enable
196         p1ld <= '1' when leds = active_high_bit0 and switches = active_high_sw0 and q > 15 and q < 25 else
197                 '1' when leds = active_high_bit1 and switches = active_high_sw1 and q > 15 and q < 25 else
198                 '1' when leds = active_high_bit2 and switches = active_high_sw2 and q > 15 and q < 25 else
199                 '1' when leds = active_high_bit3 and switches = active_high_sw3 and q > 15 and q < 25 else
200                 '1' when leds = active_high_bit4 and switches = active_high_sw4 and q > 15 and q < 25 else
201                 '1' when leds = active_high_bit5 and switches = active_high_sw5 and q > 15 and q < 25 else
202                 '1' when leds = active_high_bit6 and switches = active_high_sw6 and q > 15 and q < 25 else
203                 '1' when leds = active_high_bit7 and switches = active_high_sw7 and q > 15 and q < 25 else
204                 '1' when leds = active_high_bit8 and switches = active_high_sw8 and q > 15 and q < 25 else
205                 '1' when leds = active_high_bit9 and switches = active_high_sw9 and q > 15 and q < 25 else
206                 '0';
207
208
209         -- p2 enable score enable
210         p2ld <= '1' when leds = active_high_bit0 and switches = active_high_sw0 and q > 27 and q < 37 else
211                 '1' when leds = active_high_bit1 and switches = active_high_sw1 and q > 27 and q < 37 else
212                 '1' when leds = active_high_bit2 and switches = active_high_sw2 and q > 27 and q < 37 else
213                 '1' when leds = active_high_bit3 and switches = active_high_sw3 and q > 27 and q < 37 else
214                 '1' when leds = active_high_bit4 and switches = active_high_sw4 and q > 27 and q < 37 else
215                 '1' when leds = active_high_bit5 and switches = active_high_sw5 and q > 27 and q < 37 else
216                 '1' when leds = active_high_bit6 and switches = active_high_sw6 and q > 27 and q < 37 else
217                 '1' when leds = active_high_bit7 and switches = active_high_sw7 and q > 27 and q < 37 else
218                 '1' when leds = active_high_bit8 and switches = active_high_sw8 and q > 27 and q < 37 else
219                 '1' when leds = active_high_bit9 and switches = active_high_sw9 and q > 27 and q < 37 else
220                 '0';
221
222
223
224         -- output logic
225         hex0 <= active_low_m when q = 0 else
226                 active_low_w when q = 1 else
227                 active_low_dash when q = 2 else
228                 active_low_f when q = 3 else
229                 active_low_h when q = 4 else
230                 active_low_t when q = 5 else
231                 active_low_d when q = 6 else
232                 active_low_e when q = 7 else
233                 active_low_L when q = 8 else
234                 active_low_blank when q = 9 else
235                 active_low_blank when q = 10 else
236                 active_low_y when q = 11 else
237                 active_low_y when q = 12 else
238                 active_low_y when q = 13 else
239                 active_low_y when q = 14 else
240                 active_low_y when q = 15 else
241                 active_low_y when q = 25 else
242                 active_low_y when q = 26 else
243                 active_low_y when q = 27 else
244                 active_low_0 when q > 15 and q < 25 and p1scrq = 0 else
245                 active_low_1 when q > 15 and q < 25 and p1scrq = 1 else
246                 active_low_2 when q > 15 and q < 25 and p1scrq = 2 else
247                 active_low_3 when q > 15 and q < 25 and p1scrq = 3 else
248                 active_low_4 when q > 15 and q < 25 and p1scrq = 4 else
249                 active_low_5 when q > 15 and q < 25 and p1scrq = 5 else
250                 active_low_6 when q > 15 and q < 25 and p1scrq = 6 else
251                 active_low_7 when q > 15 and q < 25 and p1scrq = 7 else
```

```
252             active_low_8 when q > 15 and q < 25 and p1scrq = 8 else
253             active_low_9 when q > 15 and q < 25 and p1scrq = 9 else
254             active_low_0 when q > 27 and q < 37 and p2scrq = 0 else
255             active_low_1 when q > 27 and q < 37 and p2scrq = 1 else
256             active_low_2 when q > 27 and q < 37 and p2scrq = 2 else
257             active_low_3 when q > 27 and q < 37 and p2scrq = 3 else
258             active_low_4 when q > 27 and q < 37 and p2scrq = 4 else
259             active_low_5 when q > 27 and q < 37 and p2scrq = 5 else
260             active_low_6 when q > 27 and q < 37 and p2scrq = 6 else
261             active_low_7 when q > 27 and q < 37 and p2scrq = 7 else
262             active_low_8 when q > 27 and q < 37 and p2scrq = 8 else
263             active_low_9 when q > 27 and q < 37 and p2scrq = 9 else
264             active_low_0 when q > 36 and q < 40 and p2scrq = 0 else
265             active_low_1 when q > 36 and q < 40 and p2scrq = 1 else
266             active_low_2 when q > 36 and q < 40 and p2scrq = 2 else
267             active_low_3 when q > 36 and q < 40 and p2scrq = 3 else
268             active_low_4 when q > 36 and q < 40 and p2scrq = 4 else
269             active_low_5 when q > 36 and q < 40 and p2scrq = 5 else
270             active_low_6 when q > 36 and q < 40 and p2scrq = 6 else
271             active_low_7 when q > 36 and q < 40 and p2scrq = 7 else
272             active_low_8 when q > 36 and q < 40 and p2scrq = 8 else
273             active_low_9 when q > 36 and q < 40 and p2scrq = 9 else
274             active_low_blank;
275     -- output logic
276     hex1 <= active_low_o when q = 0 else
277             active_low_blank when q = 1 else
278             active_low_a when q = 2 else
279             active_low_blank when q = 3 else
280             active_low_t when q = 4 else
281             active_low_i when q = 5 else
282             active_low_n when q = 6 else
283             active_low_h when q = 7 else
284             active_low_blank when q = 8 else
285             active_low_o when q = 9 else
286             active_low_ed when q = 10 else
287             active_low_d when q = 11 else
288             active_low_d when q = 12 else
289             active_low_d when q = 13 else
290             active_low_d when q = 14 else
291             active_low_d when q = 15 else
292             active_low_d when q = 25 else
293             active_low_d when q = 26 else
294             active_low_d when q = 27 else
295             active_low_dash when q > 15 and q < 25 else
296             active_low_dash when q > 27 and q < 37 else
297             active_low_2 when q > 36 and q < 40 else
298             active_low_blank;
299
300
301
302     -- output logic
303     hex2 <= active_low_c when q = 0 else
304             active_low_o when q = 1 else
305             active_low_dash when q = 2 else
306             active_low_ed when q = 3 else
307             active_low_blank when q = 4 else
308             active_low_w when q = 5 else
309             active_low_u when q = 6 else
310             active_low_t when q = 7 else
311             active_low_t when q = 8 else
312             active_low_t when q = 9 else
313             active_low_r when q = 10 else
314             active_low_r when q = 11 else
315             active_low_r when q = 12 else
```

```
316             active_low_r when q = 13 else
317             active_low_r when q = 14 else
318             active_low_r when q = 15 else
319             active_low_r when q = 25 else
320             active_low_r when q = 26 else
321             active_low_r when q = 27 else
322             active_low_1 when q > 15 and q < 25 else
323             active_low_2 when q > 27 and q < 37 else
324             active_low_p when q > 36 and q < 40 else
325             active_low_blank;
326
327     -- output logic
328     hex3 <= active_low_L when q = 0 else
329             active_low_t when q = 1 else
330             active_low_k when q = 2 else
331             active_low_L when q = 3 else
332             active_low_p when q = 4 else
333             active_low_s when q = 5 else
334             active_low_blank when q = 6 else
335             active_low_blank when q = 7 else
336             active_low_i when q = 8 else
337             active_low_blank when q = 9 else
338             active_low_c when q = 10 else
339             active_low_blank when q = 11 else
340             active_low_blank when q = 12 else
341             active_low_blank when q = 13 else
342             active_low_blank when q = 14 else
343             active_low_blank when q = 15 else
344             active_low_blank when q = 25 else
345             active_low_blank when q = 26 else
346             active_low_blank when q = 27 else
347             active_low_p when q > 15 and q < 25 else
348             active_low_p when q > 27 and q < 37 else
349             active_low_0 when q > 36 and q < 40 and p1scrq = 0 else
350             active_low_1 when q > 36 and q < 40 and p1scrq = 1 else
351             active_low_2 when q > 36 and q < 40 and p1scrq = 2 else
352             active_low_3 when q > 36 and q < 40 and p1scrq = 3 else
353             active_low_4 when q > 36 and q < 40 and p1scrq = 4 else
354             active_low_5 when q > 36 and q < 40 and p1scrq = 5 else
355             active_low_6 when q > 36 and q < 40 and p1scrq = 6 else
356             active_low_7 when q > 36 and q < 40 and p1scrq = 7 else
357             active_low_8 when q > 36 and q < 40 and p1scrq = 8 else
358             active_low_9 when q > 36 and q < 40 and p1scrq = 9 else
359             active_low_blank;
360
361
362     -- output logic
363     hex4 <= active_low_e when q = 0 else
364             active_low_blank when q = 1 else
365             active_low_c when q = 2 else
366             active_low_o when q = 3 else
367             active_low_i when q = 4 else
368             active_low_blank when q = 5 else
369             active_low_h when q = 6 else
370             active_low_r when q = 7 else
371             active_low_L when q = 8 else
372             active_low_d when q = 9 else
373             active_low_c when q = 10 else
374             active_low_1 when q = 11 else
375             active_low_1 when q = 12 else
376             active_low_1 when q = 13 else
377             active_low_1 when q = 14 else
378             active_low_1 when q = 15 else
379             active_low_2 when q = 25 else
```

```
380             active_low_2 when q = 26 else
381             active_low_2 when q = 27 else
382             active_low_blank when q > 15 and q < 25 else
383             active_low_blank when q > 27 and q < 37 else
384             active_low_1 when q > 36 and q < 40 else
385             active_low_blank;
386
387      -- output logic
388      hex5 <= active_low_w when q = 0 else
389             active_low_e when q = 1 else
390             active_low_a when q = 2 else
391             active_low_m when q = 3 else
392             active_low_L when q = 4 else
393             active_low_e when q = 5 else
394             active_low_c when q = 6 else
395             active_low_e when q = 7 else
396             active_low_blank when q = 8 else
397             active_low_e when q = 9 else
398             active_low_s when q = 10 else
399             active_low_p when q = 11 else
400             active_low_p when q = 12 else
401             active_low_p when q = 13 else
402             active_low_p when q = 14 else
403             active_low_p when q = 15 else
404             active_low_p when q = 25 else
405             active_low_p when q = 26 else
406             active_low_p when q = 27 else
407             active_low_blank when q > 15 and q < 25 else
408             active_low_blank when q > 27 and q < 37 else
409             active_low_p when q > 36 and q < 40 else
410             active_low_blank;
411
412
413     --selector for random clock
414     sd <= sq+1 when sq < 3 else 0;
415
416     -- random selector reg
417     rndReg: process(all)
418     begin
419         if rising_edge(slow_clk) then sq <= sd;
420         end if;
421     end process rndReg;
422
423     -- select output
424     with sq select
425     rnd_slow <= less_slow_clk when 3,
426                 slow_clk when 2,
427                 less_slow_clk when 1,
428                 slow_clk when others;
429
430
431      -- clock process
432      slowTwo: process(all)
433      begin
434         if rising_edge(clk) then
435               cnt2 <= cnt2+1;
436            if cnt2 = clk2hz_value then
437               less_slow_clk <= not less_slow_clk;
438               cnt2 <= 0;
439            end if;
440         end if;
441      end process slowTwo;
442
443
```

```
428             slow_clk when others;
429
430
431     -- clock process
432     slowTwo: process(all)
433     begin
434        if rising_edge(clk) then
435              cnt2 <= cnt2+1;
436           if cnt2 = clk2hz_value then
437              less_slow_clk <= not less_slow_clk;
438              cnt2 <= 0;
439           end if;
440        end if;
441     end process slowTwo;
442
443
444     -- clock process
445     counter: process(all)
446     begin
447        if rising_edge(clk) then
448              cnt <= cnt+1;
449           if cnt = clk1hz_value then
450              slow_clk <= not slow_clk;
451              cnt <= 0;
452           end if;
453        end if;
454     end process;
455
456  end architecture behavioral;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   package dseg7 is
5
6       constant active_low_blank : std_logic_vector(7 downto 0) := B"11111111";
7       constant active_low_dash : std_logic_vector(7 downto 0) := B"10111111";
8
9       -- THese numbers don't have the decimal
10      constant active_low_0 : std_logic_vector(7 downto 0) := B"11000000";
11      constant active_low_1 : std_logic_vector(7 downto 0) := B"11111001";
12      constant active_low_2 : std_logic_vector(7 downto 0) := B"10100100";
13      constant active_low_3 : std_logic_vector(7 downto 0) := B"10110000";
14      constant active_low_4 : std_logic_vector(7 downto 0) := B"10011001";
15      constant active_low_5 : std_logic_vector(7 downto 0) := B"10010010";
16      constant active_low_6 : std_logic_vector(7 downto 0) := B"10000010";
17      constant active_low_7 : std_logic_vector(7 downto 0) := B"11011000";
18      constant active_low_8 : std_logic_vector(7 downto 0) := B"10000000";
19      constant active_low_9 : std_logic_vector(7 downto 0) := B"10010000";
20
21      -- These numbers have the decimal after them
22      constant active_low_0d : std_logic_vector(7 downto 0) := B"01000000";
23      constant active_low_1d : std_logic_vector(7 downto 0) := B"01111001";
24      constant active_low_2d : std_logic_vector(7 downto 0) := B"00100100";
25      constant active_low_3d : std_logic_vector(7 downto 0) := B"00110000";
26      constant active_low_4d : std_logic_vector(7 downto 0) := B"00011001";
27      constant active_low_5d : std_logic_vector(7 downto 0) := B"00010010";
28      constant active_low_6d : std_logic_vector(7 downto 0) := B"00000010";
29      constant active_low_7d : std_logic_vector(7 downto 0) := B"01011000";
30      constant active_low_8d : std_logic_vector(7 downto 0) := B"00000000";
31      constant active_low_9d : std_logic_vector(7 downto 0) := B"00010000";
32
33      -- The alphabet without a decimal
34      constant active_low_a : std_logic_vector(7 downto 0) := B"10001000";
35      constant active_low_b : std_logic_vector(7 downto 0) := B"10000011";
36      constant active_low_c : std_logic_vector(7 downto 0) := B"10100111";
37      constant active_low_d : std_logic_vector(7 downto 0) := B"10100001";
38      constant active_low_e : std_logic_vector(7 downto 0) := B"10000110";
39      constant active_low_f : std_logic_vector(7 downto 0) := B"10001110";
40      constant active_low_g : std_logic_vector(7 downto 0) := B"11000010";
41      constant active_low_h : std_logic_vector(7 downto 0) := B"10001011";
42      constant active_low_i : std_logic_vector(7 downto 0) := B"11111011";
43      constant active_low_j : std_logic_vector(7 downto 0) := B"11100001";
44      constant active_low_k : std_logic_vector(7 downto 0) := B"10001010";
45      constant active_low_L : std_logic_vector(7 downto 0) := B"11000111";
46      constant active_low_m : std_logic_vector(7 downto 0) := B"11001000";
47      constant active_low_n : std_logic_vector(7 downto 0) := B"10101011";
48      constant active_low_o : std_logic_vector(7 downto 0) := B"10100011";
49      constant active_low_p : std_logic_vector(7 downto 0) := B"10001100";
50      constant active_low_q : std_logic_vector(7 downto 0) := B"10011000";
51      constant active_low_r : std_logic_vector(7 downto 0) := B"10101111";
52      constant active_low_s : std_logic_vector(7 downto 0) := B"10010011";
53      constant active_low_t : std_logic_vector(7 downto 0) := B"10000111";
54      constant active_low_u : std_logic_vector(7 downto 0) := B"11100011";
55      constant active_low_v : std_logic_vector(7 downto 0) := B"11000001";
56      constant active_low_w : std_logic_vector(7 downto 0) := B"10000001";
57      constant active_low_x : std_logic_vector(7 downto 0) := B"10001001";
58      constant active_low_y : std_logic_vector(7 downto 0) := B"10010001";
59      constant active_low_z : std_logic_vector(7 downto 0) := B"11100100";
60
61      -- The alphabet with a decimal
62      constant active_low_ad : std_logic_vector(7 downto 0) := B"00001000";
63      constant active_low_bd : std_logic_vector(7 downto 0) := B"00000011";
64      constant active_low_cd : std_logic_vector(7 downto 0) := B"00100111";
```

```
39      constant active_low_f : std_logic_vector(7 downto 0) := B"10001110";
40      constant active_low_g : std_logic_vector(7 downto 0) := B"11000010";
41      constant active_low_h : std_logic_vector(7 downto 0) := B"10001011";
42      constant active_low_i : std_logic_vector(7 downto 0) := B"11111011";
43      constant active_low_j : std_logic_vector(7 downto 0) := B"11100001";
44      constant active_low_k : std_logic_vector(7 downto 0) := B"10001010";
45      constant active_low_L : std_logic_vector(7 downto 0) := B"11000111";
46      constant active_low_m : std_logic_vector(7 downto 0) := B"11001000";
47      constant active_low_n : std_logic_vector(7 downto 0) := B"10101011";
48      constant active_low_o : std_logic_vector(7 downto 0) := B"10100011";
49      constant active_low_p : std_logic_vector(7 downto 0) := B"10001100";
50      constant active_low_q : std_logic_vector(7 downto 0) := B"10011000";
51      constant active_low_r : std_logic_vector(7 downto 0) := B"10101111";
52      constant active_low_s : std_logic_vector(7 downto 0) := B"10010011";
53      constant active_low_t : std_logic_vector(7 downto 0) := B"10000111";
54      constant active_low_u : std_logic_vector(7 downto 0) := B"11100011";
55      constant active_low_v : std_logic_vector(7 downto 0) := B"11000001";
56      constant active_low_w : std_logic_vector(7 downto 0) := B"10000001";
57      constant active_low_x : std_logic_vector(7 downto 0) := B"10001001";
58      constant active_low_y : std_logic_vector(7 downto 0) := B"10010001";
59      constant active_low_z : std_logic_vector(7 downto 0) := B"11100100";
60
61      -- The alphabet with a decimal
62      constant active_low_ad : std_logic_vector(7 downto 0) := B"00001000";
63      constant active_low_bd : std_logic_vector(7 downto 0) := B"00000011";
64      constant active_low_cd : std_logic_vector(7 downto 0) := B"00100111";
65      constant active_low_dd : std_logic_vector(7 downto 0) := B"00100001";
66      constant active_low_ed : std_logic_vector(7 downto 0) := B"00000110";
67      constant active_low_fd : std_logic_vector(7 downto 0) := B"00001110";
68      constant active_low_gd : std_logic_vector(7 downto 0) := B"01000010";
69      constant active_low_hd : std_logic_vector(7 downto 0) := B"00001011";
70      constant active_low_id : std_logic_vector(7 downto 0) := B"01111011";
71      constant active_low_jd : std_logic_vector(7 downto 0) := B"01100001";
72      constant active_low_kd : std_logic_vector(7 downto 0) := B"00001010";
73      constant active_low_Ld : std_logic_vector(7 downto 0) := B"01000111";
74      constant active_low_md : std_logic_vector(7 downto 0) := B"01001000";
75      constant active_low_nd : std_logic_vector(7 downto 0) := B"00101011";
76      constant active_low_od : std_logic_vector(7 downto 0) := B"00100011";
77      constant active_low_pd : std_logic_vector(7 downto 0) := B"00001100";
78      constant active_low_qd : std_logic_vector(7 downto 0) := B"00011000";
79      constant active_low_rd : std_logic_vector(7 downto 0) := B"00101111";
80      constant active_low_sd : std_logic_vector(7 downto 0) := B"00010011";
81      constant active_low_td : std_logic_vector(7 downto 0) := B"00000111";
82      constant active_low_ud : std_logic_vector(7 downto 0) := B"01100011";
83      constant active_low_vd : std_logic_vector(7 downto 0) := B"01000001";
84      constant active_low_wd : std_logic_vector(7 downto 0) := B"00000001";
85      constant active_low_xd : std_logic_vector(7 downto 0) := B"00001001";
86      constant active_low_yd : std_logic_vector(7 downto 0) := B"00010001";
87      constant active_low_zd : std_logic_vector(7 downto 0) := B"01100100";
88
89  end package dseg7;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;

package ledstrip is

    --Leds
    constant active_high_bit9 : std_logic_vector(9 downto 0) := B"1000000000";
    constant active_high_bit8 : std_logic_vector(9 downto 0) := B"0100000000";
    constant active_high_bit7 : std_logic_vector(9 downto 0) := B"0010000000";
    constant active_high_bit6 : std_logic_vector(9 downto 0) := B"0001000000";
    constant active_high_bit5 : std_logic_vector(9 downto 0) := B"0000100000";
    constant active_high_bit4 : std_logic_vector(9 downto 0) := B"0000010000";
    constant active_high_bit3 : std_logic_vector(9 downto 0) := B"0000001000";
    constant active_high_bit2 : std_logic_vector(9 downto 0) := B"0000000100";
    constant active_high_bit1 : std_logic_vector(9 downto 0) := B"0000000010";
    constant active_high_bit0 : std_logic_vector(9 downto 0) := B"0000000001";
    constant active_high_blank : std_logic_vector(9 downto 0) := B"0000000000";

    -- Switches
    constant active_high_sw9 : std_logic_vector(9 downto 0) := B"1000000000";
    constant active_high_sw8 : std_logic_vector(9 downto 0) := B"0100000000";
    constant active_high_sw7 : std_logic_vector(9 downto 0) := B"0010000000";
    constant active_high_sw6 : std_logic_vector(9 downto 0) := B"0001000000";
    constant active_high_sw5 : std_logic_vector(9 downto 0) := B"0000100000";
    constant active_high_sw4 : std_logic_vector(9 downto 0) := B"0000010000";
    constant active_high_sw3 : std_logic_vector(9 downto 0) := B"0000001000";
    constant active_high_sw2 : std_logic_vector(9 downto 0) := B"0000000100";
    constant active_high_sw1 : std_logic_vector(9 downto 0) := B"0000000010";
    constant active_high_sw0 : std_logic_vector(9 downto 0) := B"0000000001";
    constant active_high_Nosw : std_logic_vector(9 downto 0) := B"0000000000";

end package ledstrip;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;

package clockdiv is

    constant crystal : integer := 50000000;
    constant clk1hz_value : integer := crystal/2;   -- 1 second clock
    constant clk2hz_value : integer := crystal/4;   -- 1/2 second clock
    constant clk3hz_value : integer := crystal/6;   -- 1/3 second clock
    constant clk4hz_value : integer := crystal/8;   -- 1/4 second clock
    constant clk8hz_value : integer := crystal/16;
    constant clk16hz_value : integer := crystal/32;

end package clockdiv;
```