

```

1  -- $Id: pattern_generator.vhd,v 1.4 2004/12/09 22:32:59 tofp Exp $
2  --***** PATTERN_GENERATOR.VHD : Pattern generator module.
3  --
4  --
5  --
6  --* REVISION HISTORY:
7  --* 26-Apr-2001 CS Original coding
8  --* 24-Oct-2001 CS Length input was replaced with bl_reg
9  --* Pattern input was replaced with ps_reg
10 --* 22-Mar-2002 CS Pattern codes are changed to match with pRORC params
11 --* 24-Apr-2002 CS Event ID counter is implemented
12 --* 2-May-2002 CS Counter control has been changed
13 --* 3-May-2002 CS Random length generation
14 --*
15 --*****
16
17
18 LIBRARY ieee;
19 USE ieee.std_logic_1164.ALL;
20 USE work.my_conversions.ALL;
21 USE work.my_utilities.ALL;
22
23
24 ENTITY pattern_generator IS
25   PORT (
26     clock      : IN std_logic;
27     arstn     : IN std_logic;
28     ps_reg     : IN std_logic_vector ( 7 DOWNTO 0);
29     bl_reg     : IN std_logic_vector ( 7 DOWNTO 0);
30     xx_reg     : IN std_logic_vector ( 7 DOWNTO 0);
31     tid        : IN std_logic_vector ( 3 DOWNTO 0);
32     enable      : IN std_logic;
33     suspend     : IN std_logic;
34     reset_evid  : IN std_logic;
35     fifo_q      : IN std_logic_vector (31 DOWNTO 0);
36     fifo_empty   : IN std_logic;
37     fifo_rdreq   : OUT std_logic;
38     datao       : OUT std_logic_vector (32 DOWNTO 0);
39     datao_valid  : OUT std_logic);
40 END pattern_generator;
41
42 ARCHITECTURE SYN OF pattern_generator IS
43
44   CONSTANT FIFO_OUT  : std_logic_vector := "001";
45   CONSTANT ALTER_OF   : std_logic_vector := "010";
46   CONSTANT FLYING_0   : std_logic_vector := "011";
47   CONSTANT FLYING_1   : std_logic_vector := "100";
48   CONSTANT INCREMENT  : std_logic_vector := "101";
49   CONSTANT DECREMENT  : std_logic_vector := "110";
50
51   CONSTANT FESTWEOB : std_logic_vector := "01100100";
52
53   TYPE pg_state IS (
54     IDLE,
55     INITPG1,
56     TXEVID,
57     TXDATA,
58     TXIDLE,
59     TXDTSW,
60     WAITGAP
61   );
62
63   SIGNAL s_suspend    : std_logic;
64   SIGNAL is_fifo_out  : boolean;
65
66 BEGIN

```

```

67
68     s_suspend    <= (suspend OR fifo_empty) WHEN (ps_reg(2 DOWNTO 0) = FIFO_OUT) ELSE suspend;
69     -- this following variable allows FIFO_OUT for both (ps_reg = 0) and (ps_reg = 1)
70     is_fifo_out <= (ps_reg(2 DOWNTO 0) = FIFO_OUT) OR (ps_reg(2 DOWNTO 0) = "000");
71
72     main : PROCESS (clock, arstn)
73         VARIABLE pg_present : pg_state;
74         VARIABLE pg_next      : pg_state;
75
76         VARIABLE pgdata          : std_logic_vector (31 DOWNTO 0);
77         VARIABLE dtstw           : std_logic_vector (31 DOWNTO 0);
78         VARIABLE shift_reg        : std_logic_vector (31 DOWNTO 0);
79         VARIABLE shiftrg_init     : boolean;
80         VARIABLE shiftrg_enable   : boolean;
81         VARIABLE counter_reg      : std_logic_vector (31 DOWNTO 0);
82         VARIABLE counter_init     : boolean;
83         VARIABLE counter_enable   : boolean;
84         VARIABLE alternate_reg    : std_logic_vector (31 DOWNTO 0);
85         VARIABLE alterrg_init     : boolean;
86         VARIABLE alterrg_enable   : boolean;
87         VARIABLE actual_length    : std_logic_vector (18 DOWNTO 0);
88         VARIABLE fixed_length     : std_logic_vector (18 DOWNTO 0);
89         VARIABLE rand_length      : std_logic_vector (18 DOWNTO 0);
90         VARIABLE rand_mask        : std_logic_vector (18 DOWNTO 0);
91         VARIABLE rand_msb         : std_logic;
92         VARIABLE random_data      : std_logic_vector (31 DOWNTO 0);
93         VARIABLE event_id         : std_logic_vector (31 DOWNTO 0);
94         VARIABLE word_counter     : std_logic_vector (19 DOWNTO 0);
95         VARIABLE block_counter    : std_logic_vector (18 DOWNTO 0);
96         VARIABLE block_end         : std_logic;
97
98     BEGIN
99
100        IF (arstn = '0') THEN
101
102            dataao      <= (OTHERS => '0');
103            dataao_valid <= '0';
104            fifo_rdreq  <= '0';
105
106            pg_present    := IDLE;
107            pg_next       := IDLE;
108            pgdata         := (OTHERS => '0');
109            shift_reg      := (OTHERS => '0');
110            shiftrg_init    := false;
111            shiftrg_enable  := false;
112            counter_reg    := (OTHERS => '0');
113            counter_init    := false;
114            counter_enable  := false;
115            alternate_reg   := (OTHERS => '0');
116            alterrg_init    := false;
117            alterrg_enable  := false;
118            actual_length   := (6      => '1', OTHERS => '0');
119            fixed_length    := (6      => '1', OTHERS => '0');
120            rand_length     := (OTHERS => '1');
121            rand_mask       := int2slv(63, 19);
122            rand_msb        := '1';
123            random_data     := (OTHERS => '0');
124            event_id        := (0      => '1', OTHERS => '0');
125            word_counter    := (OTHERS => '0');
126            block_counter   := (0      => '1', OTHERS => '0');
127            block_end        := '0';
128
129        ELSIF (clock'event AND clock = '1') THEN
130
131            IF (is_fifo_out) THEN
132                block_end := bool2slv( fifo_q(31 DOWNTO 24) = X"EA");

```

```

133 ELSE
134     block_end := word_counter(19);
135 END IF;
136
137 IF (pg_present = IDLE OR pg_present = TXDTSW) THEN
138     word_counter := ('0' & actual_length);
139 ELSIF (pg_present = TXIDLE) THEN
140     word_counter := word_counter;
141 ELSE
142     word_counter := dec(word_counter);
143 END IF;
144
145 IF (bl_reg(4) = '0') THEN
146     actual_length := fixed_length;
147 ELSE
148     actual_length := rand_length AND rand_mask;
149 END IF;
150
151 IF (reset_evid = '1') THEN
152     rand_length := "10000000000" & xx_reg;
153 ELSIF (pg_present = INITPG1) THEN
154     rand_msб := rand_length(18);
155     rand_length(18) := rand_length(17);
156     rand_length(17) := rand_length(16);
157     rand_length(16) := rand_length(15);
158     rand_length(15) := rand_length(14);
159     rand_length(14) := rand_length(13);
160     rand_length(13) := rand_length(12);
161     rand_length(12) := rand_length(11);
162     rand_length(11) := rand_length(10);
163     rand_length(10) := rand_length(9);
164     rand_length(9) := rand_length(8);
165     rand_length(8) := rand_length(7);
166     rand_length(7) := rand_length(6);
167     rand_length(6) := rand_length(5);
168     rand_length(5) := rand_length(4) XOR rand_msб;
169     rand_length(4) := rand_length(3);
170     rand_length(3) := rand_length(2);
171     rand_length(2) := rand_length(1) XOR rand_msб;
172     rand_length(1) := rand_length(0) XOR rand_msб;
173     rand_length(0) := rand_msб;
174 END IF;
175
176 CASE bl_reg(3 DOWNTO 0) IS
177     WHEN "0001" =>
178         fixed_length := int2slv(15, 19);
179         rand_mask := int2slv(15, 19);
180     WHEN "0010" =>
181         fixed_length := int2slv(31, 19);
182         rand_mask := int2slv(31, 19);
183     WHEN "0011" =>
184         fixed_length := int2slv(63, 19);
185         rand_mask := int2slv(63, 19);
186     WHEN "0100" =>
187         fixed_length := int2slv(127, 19);
188         rand_mask := int2slv(127, 19);
189     WHEN "0101" =>
190         fixed_length := int2slv(255, 19);
191         rand_mask := int2slv(255, 19);
192     WHEN "0110" =>
193         fixed_length := int2slv(511, 19);
194         rand_mask := int2slv(511, 19);
195     WHEN "0111" =>
196         fixed_length := int2slv(1023, 19);
197         rand_mask := int2slv(1023, 19);
198     WHEN "1000" =>

```

```

199     fixed_length := int2slv(2047, 19);
200     rand_mask    := int2slv(2047, 19);
201 WHEN "1001" =>
202     fixed_length := int2slv(4095, 19);
203     rand_mask    := int2slv(4095, 19);
204 WHEN "1010" =>
205     fixed_length := int2slv(8191, 19);
206     rand_mask    := int2slv(8191, 19);
207 WHEN "1011" =>
208     fixed_length := int2slv(16383, 19);
209     rand_mask    := int2slv(16383, 19);
210 WHEN "1100" =>
211     fixed_length := int2slv(32767, 19);
212     rand_mask    := int2slv(32767, 19);
213 WHEN "1101" =>
214     fixed_length := int2slv(65535, 19);
215     rand_mask    := int2slv(65535, 19);
216 WHEN "1110" =>
217     fixed_length := int2slv(131071, 19);
218     rand_mask    := int2slv(131071, 19);
219 WHEN "1111" =>
220     fixed_length := int2slv(262143, 19);
221     rand_mask    := int2slv(262143, 19);
222 WHEN OTHERS =>
223     fixed_length := int2slv(15, 19);
224     rand_mask    := int2slv(15, 19);
225 END CASE;
226
227 CASE ps_reg(2 DOWNTO 0) IS
228 WHEN FIFO_OUT =>
229     pgdata := fifo_q;
230 WHEN ALTER_OF =>
231     pgdata := alternate_reg;
232 WHEN FLYING_0 =>
233     pgdata := shift_reg;
234 WHEN FLYING_1 =>
235     pgdata := shift_reg;
236 WHEN INCREMENT =>
237     pgdata := counter_reg;
238 WHEN DECREMENT =>
239     pgdata := counter_reg;
240 WHEN OTHERS =>
241     pgdata := fifo_q; -- default is FIFO_OUT
242 END CASE;
243
244 IF (counter_init) THEN
245     counter_reg := (OTHERS => '0');
246 ELSIF (counter_enable) THEN
247     IF (ps_reg(2 DOWNTO 0) = DECREMENT) THEN
248         counter_reg := dec(counter_reg);
249     ELSE
250         counter_reg := inc(counter_reg);
251     END IF;
252 END IF;
253
254 IF (shiftrg_init) THEN
255     IF (ps_reg(2 DOWNTO 0) = FLYING_1) THEN
256         shift_reg := ( 0 => '1', OTHERS => '0' );
257     ELSE
258         shift_reg := ( 0 => '0', OTHERS => '1' );
259     END IF;
260 ELSIF (shiftrg_enable) THEN
261     shift_reg := (shift_reg(30 DOWNTO 0) & shift_reg(31));
262 END IF;
263
264 IF (alterrge_init) THEN

```

```

265     alternate_reg := (OTHERS => '0');
266 ELSIF (alterrg_enable) THEN
267     alternate_reg := NOT alternate_reg;
268 END IF;
269
270 CASE pg_present IS
271     WHEN IDLE =>
272         dataao      <= ('0' & pgdata);
273         dataao_valid <= '0';
274     WHEN INITPG1 =>
275         dataao      <= ('0' & pgdata);
276         dataao_valid <= '0';
277     WHEN TXEVID =>
278         dataao      <= ('0' & event_id);
279         dataao_valid <= '1';
280     WHEN TXDATA =>
281         dataao <= ('0' & pgdata);
282         IF (is_fifo_out) THEN
283             dataao_valid <= NOT fifo_empty;
284         ELSE
285             dataao_valid <= '1';
286         END IF;
287     WHEN TXIDLE =>
288         dataao      <= ('0' & pgdata);
289         dataao_valid <= '0';
290     WHEN TXDTSW =>
291         dataao      <= ('1' & dtstw);
292         dataao_valid <= '1';
293     WHEN WAITGAP =>
294         dataao      <= ('0' & pgdata);
295         dataao_valid <= '0';
296 END CASE;
297 dtstw := ('0' & block_counter & tid & FESTWEBO);
298
299 IF (counter_init) THEN
300     block_counter := (0 => '1', OTHERS => '0');
301 ELSIF (counter_enable) THEN
302     block_counter := inc(block_counter);
303 END IF;
304
305 IF (reset_evid = '1') THEN
306     event_id := (0 => '1', OTHERS => '0');
307 ELSIF (pg_present = TXEVID) THEN
308     event_id := inc(event_id);
309 END IF;
310
311 CASE pg_present IS
312     WHEN IDLE =>
313         IF (enable = '1' AND s_suspend = '0') THEN
314             pg_next := INITPG1;
315         ELSE
316             pg_next := IDLE;
317         END IF;
318     WHEN INITPG1 =>
319         IF (is_fifo_out) THEN
320             pg_next := TXDATA;
321         ELSE
322             pg_next := TXEVID;
323         END IF;
324     WHEN TXEVID =>                                -- 24.04.2002
325         IF (block_end = '1') THEN
326             pg_next := TXDTSW;
327         ELSE
328             pg_next := TXDATA;
329         END IF;
330     WHEN TXDATA =>

```

```

331   IF (block_end = '1') THEN
332     pg_next := TXDTSW;
333   ELSIF (s_suspend = '1') THEN
334     pg_next := TXIDLE;
335   ELSE
336     pg_next := TXDATA;
337   END IF;
338 WHEN TXIDLE =>
339   IF (s_suspend = '0') THEN
340     pg_next := TXDATA;
341   ELSIF (enable = '0') THEN      -- 22.03.2002
342     pg_next := IDLE;           -- 22.03.2002
343   ELSE
344     pg_next := TXIDLE;
345   END IF;
346 WHEN TXDTSW =>
347   pg_next := WAITGAP;
348 WHEN WAITGAP =>
349   IF (enable = '0') THEN
350     pg_next := IDLE;
351   ELSE
352     pg_next := WAITGAP;
353   END IF;
354 END CASE;
355 pg_present := pg_next;
356
357   fifo_rdreq <= bool2sl(pg_next = TXDATA);
358
359   counter_init := (pg_next = INITPG1 OR pg_next = TXDTSW);
360   counter_enable := (pg_next = TXDATA);
361   shiftrg_init := (pg_next = INITPG1 OR pg_next = TXDTSW);
362   shiftrg_enable := (pg_next = TXDATA);
363   alterrg_init := (pg_next = INITPG1 OR pg_next = TXDTSW);
364   alterrg_enable := (pg_next = TXDATA);
365
366   END IF;
367 END PROCESS main;
368
369 END SYN;
370

```