

```

1  LIBRARY ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use IEEE.std_logic_misc.all;
5
6  entity top_10Obs is
7
8  port (
9      CLOCK_50          :in      std_logic;
10     KEY                :in      std_logic_vector(3 downto 0) ;
11     GPIO               :out      std_logic_vector(34 downto 0) );
12
13 end entity;
14
15 -----
16 ----- ARCHITECTURE -----
17 -----
18 architecture rtl of top_10Obs is
19
20     constant tau_range :integer := 20;
21
22
23     component Altplc is
24         PORT(
25             areset      : IN STD_LOGIC  := '0';
26             inclk0      : IN STD_LOGIC  := '0';
27             c0          : OUT STD_LOGIC ; -- 50Mhz
28             c1          : OUT STD_LOGIC ; -- 50Mhz
29             c2          : OUT STD_LOGIC ; -- 100 Mhz
30             c3          : OUT STD_LOGIC  -- 100 Mhz
31         );
32     end component;
33
34
35
36     component signalgenerator is
37         port(
38             clk      :in  std_logic  := 'X';
39             reset    :in  std_logic  := 'X';   -- clk
40             output   :out std_logic -- export
41         );
42     end component signalgenerator;
43
44
45     component observer
46         generic (
47             observernumber :unsigned(15 downto 0):=x"0001" -- how many observer are
48             instantiated
49         );
50         PORT (
51             clk      :in      std_logic      := 'X';
52             reset    :in      std_logic      := 'X';
53             enable_in :in      std_logic;
54             invariance_tau :in  std_logic_vector(7 downto 0);
55             signal_phi :in      std_logic;
56             output    :out      std_logic;
57             enable_out :out      std_logic
58         );
59     end component;
60
61 -----
62 -- <BEGIN_0>
63 FOR OBS_0 : observer
64     use entity work.observer(Behavioural);
65 FOR OBS_1 : observer
66     use entity work.observer(Behavioural);
67 FOR OBS_2 : observer
68     use entity work.observer(Behavioural);
69 FOR OBS_3 : observer
70     use entity work.observer(Behavioural);
71 FOR OBS_4 : observer

```

```

71     use entity work.observer(Behavioural);
72 FOR OBS_5 : observer
73     use entity work.observer(Behavioural);
74 FOR OBS_6 : observer
75     use entity work.observer(Behavioural);
76 FOR OBS_7 : observer
77     use entity work.observer(Behavioural);
78 FOR OBS_8 : observer
79     use entity work.observer(Behavioural);
80 FOR OBS_9 : observer
81     use entity work.observer(Behavioural);
82
83 -- <END_0>
84 -----
85
86 signal clk_s          : std_logic      :='0';
87 signal clk_g          : std_logic      :='0';
88 signal reset_s        : std_logic      :='0';
89 signal enable_s       : std_logic      :='0';
90 signal phi_s          : std_logic      :='0';
91 signal next_obs_s     : std_logic      :='0';
92
93 -- <BEGIN_1>
94 signal add: std_logic_vector(9 downto 0):=(others=>'0');
95 signal en1            :std_logic:= '0';
96 signal en2            :std_logic:= '0';
97 signal en3            :std_logic:= '0';
98 signal en4            :std_logic:= '0';
99 signal en5            :std_logic:= '0';
100 signal en6            :std_logic:= '0';
101 signal en7            :std_logic:= '0';
102 signal en8            :std_logic:= '0';
103 signal en9            :std_logic:= '0';
104
105 -- <END_1>
106 -----
107 signal output_s       : std_logic      :='0';
108 signal tau_s          : std_logic_vector(7 downto 0) := (others => '0');
109
110
111 -----
112 ----- BEGIN OF ARCHITECTURE -----
113 -----
114 begin
115
116     signalgenerator_top : component signalgenerator
117         port map (
118             output => phi_s,
119             reset => reset_s,
120             clk => clk_g
121         );
122
123     PLL: component AltPLC
124     PORT MAP (areset => reset_s,inclk0 => CLOCK_50 ,c1 => clk_g,c0 =>clk_s) ;
125
126 -----
127 -- <BEGIN_2>
128 OBS_0: observer GENERIC MAP(observernumber => x"000A")
129     PORT MAP ( output=>add(0),clk=>clk_s,reset =>reset_s, enable_in =>enable_s,invariance_tau => tau_s,
130     signal_phi=> phi_s,enable_out=>en1) ;
131 OBS_1: observer GENERIC MAP(observernumber => x"000A")
132     PORT MAP ( output=>add(1),clk=>clk_s,reset =>reset_s, enable_in =>en1,invariance_tau => tau_s,signal_phi=> phi_s,enable_out=>en2) ;
133 OBS_2: observer GENERIC MAP(observernumber => x"000A")
134     PORT MAP ( output=>add(2),clk=>clk_s,reset =>reset_s, enable_in =>en2,invariance_tau => tau_s,signal_phi=> phi_s,enable_out=>en3) ;
135 OBS_3: observer GENERIC MAP(observernumber => x"000A")
136     PORT MAP ( output=>add(3),clk=>clk_s,reset =>reset_s, enable_in =>en3,invariance_tau => tau_s,signal_phi=> phi_s,enable_out=>en4) ;

```

```

136 l_phi=> phi_s,enable_out=>en4);
137 OBS_4: observer GENERIC MAP(observernumber => x"000A")
138   PORT MAP ( output=>add(4),clk=>clk_s,reset =>reset_s, enable_in =>en4,invariance_tau => tau_s,signa
139   l_phi=> phi_s,enable_out=>en5);
138 OBS_5: observer GENERIC MAP(observernumber => x"000A")
139   PORT MAP ( output=>add(5),clk=>clk_s,reset =>reset_s, enable_in =>en5,invariance_tau => tau_s,signa
140   l_phi=> phi_s,enable_out=>en6);
140 OBS_6: observer GENERIC MAP(observernumber => x"000A")
141   PORT MAP ( output=>add(6),clk=>clk_s,reset =>reset_s, enable_in =>en6,invariance_tau => tau_s,signa
142   l_phi=> phi_s,enable_out=>en7);
142 OBS_7: observer GENERIC MAP(observernumber => x"000A")
143   PORT MAP ( output=>add(7),clk=>clk_s,reset =>reset_s, enable_in =>en7,invariance_tau => tau_s,signa
144   l_phi=> phi_s,enable_out=>en8);
144 OBS_8: observer GENERIC MAP(observernumber => x"000A")
145   PORT MAP ( output=>add(8),clk=>clk_s,reset =>reset_s, enable_in =>en8,invariance_tau => tau_s,signa
146   l_phi=> phi_s,enable_out=>en9);
146 OBS_9: observer GENERIC MAP(observernumber => x"000A")
147   PORT MAP ( output=>add(9),clk=>clk_s,reset =>reset_s, enable_in =>en9,invariance_tau => tau_s,signa
148   l_phi=> phi_s,enable_out=> next_obs_s);
149
150 --- <END_2>
-----
151
152
153
154 --- <BEGIN_3>
155 output_s <= and_reduce(add);
156 --- <END_3>
-----
157
158
159
160 -----FPGA OUTPUTS-----
-----
161
162
163 reset_s <= not KEY(0);
164
165 GPIO(0) <= reset_s;
166 GPIO(1) <= enable_s;
167 GPIO(2) <= en1;
168 GPIO(3) <= en2;
169 GPIO(4) <= en3;
170 GPIO(5) <= en4;
171 GPIO(6) <= en5;
172 GPIO(7) <= en6;
173 GPIO(8) <= en7;
174 GPIO(9) <= en8;
175 GPIO(10) <= en9;
176 GPIO(11) <= next_obs_s;
177 GPIO(12) <= clk_g;
178 GPIO(13) <= phi_s;
179 GPIO(14) <= clk_s;
180
181 GPIO(15) <= add(0);
182 GPIO(16) <= add(1);
183 GPIO(17) <= add(2);
184 GPIO(18) <= add(3);
185 GPIO(19) <= add(4);
186 GPIO(20) <= add(5);
187 GPIO(21) <= add(6);
188 GPIO(22) <= add(7);
189 GPIO(23) <= add(8);
190 GPIO(24) <= add(9);
191 GPIO(25) <= output_s;
192
193 tau_s          <= std_logic_vector(to_unsigned(tau_range,8));
194
195 -----SYNCHRONIZED-----
-----
196 sync:process(clk_s)

```

```

197 begin
198   if(clk_s'event and clk_s='1') then
199     if reset_s ='0' then
200       enable_s <= '1';
201     else
202       enable_s <= '0';
203     end if;
204   end if;
205 end process;
206
207 end architecture;

```