

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use IEEE.NUMERIC_STD.ALL;
4
5
6  architecture Behavioural of observer is
7
8
9  signal inc_tau                : unsigned(8 downto 0) := "000000000";
10 signal count,count_next      : unsigned(15 downto 0) := x"0001";
11 signal count_p,count_p_next  : unsigned(15 downto 0) := x"0002";
12 signal cycle,cycle_next     : unsigned(15 downto 0) := x"0000";
13 signal direction,direction_next : std_logic := '1';
14
15
16 signal enable_logic          : std_logic := '0';
17 signal output_next          : std_logic := '0';
18 begin --BEGIN ARCHITECTURE
19
20 -- parallel logic
21 inc_tau <= unsigned(invariance_tau) + to_unsigned(1,9) ;
22 enable_logic <= enable_in and not reset;
23
24 -- changes cycle up from 0 to observernumber and down back to 0
25 comb_cycle: process(cycle,direction,enable_logic)
26 begin --changes cycle_next, direction, changeDirection
27   if(direction = '0' and enable_logic = '1') then
28     if(cycle = 0)then
29       direction_next <= '1';
30       cycle_next <= cycle + 1;
31     else
32       direction_next <= '0';
33       cycle_next <= cycle - 1;
34     end if;
35   elsif(direction = '1' and enable_logic = '1') then
36     if(cycle = observernumber)then
37       direction_next <= '0';
38       cycle_next <= cycle - 1;
39     else
40       direction_next <= '1';
41       cycle_next <= cycle + 1;
42     end if;
43   else
44     direction_next <= direction;
45     cycle_next <= cycle;
46   end if;
47 end process comb_cycle;
48
49
50 -- main logic of the observer
51 comb_logic: process(inc_tau,count,count_p,cycle,signal_phi,enable_logic)
52 begin
53   if ( (cycle = observernumber or cycle = 0) and enable_logic = '1') then -- m
54     cycles passed
55     if(signal_phi = '0') then -- w(phi) = 0
56       count_next <= x"0001";
57       count_p_next <= x"0002";
58       output_next <= '0';
59     elsif(count_p <= inc_tau) then
60       count_next <= count + 1; --every clock cycle
61       count_p_next <= count_p + 1 ;
62       output_next <= '0';
63     else
64       count_next <= count;
65       count_p_next <= count_p;
66       output_next <= '1';
67     end if;
68   elsif(count_p <= inc_tau and enable_logic = '1') then
69     count_next <= count + 1; --every clock cycle
70     count_p_next <= count_p + 1 ;
71     output_next <= '0';

```

```

71   elsif(count_p > inc_tau and enable_logic = '1') then
72     count_next <= count;
73     count_p_next <= count_p;
74     output_next <= '1';
75   else
76     count_next <= count;
77     count_p_next <= count_p;
78     output_next <= '0';
79   end if;
80 end process comb_logic;
81
82 --the synchronisation logic
83 sync: process(clk,enable_logic)
84 begin
85   if(clk'event and clk = '0')then
86     if(enable_logic = '1') then
87       cycle <= cycle_next;
88       direction <= direction_next;
89       count <= count_next;
90       count_p <= count_p_next;
91       output <= output_next;
92       enable_out <= '1';
93     else
94       cycle <= x"0000";
95       direction <= '1';
96       count <= x"0001";
97       count_p <= x"0002";
98       output <= '0';
99       enable_out <= '0';
100    end if;
101  end if;
102 end process sync;
103 end architecture;--END ARCHITECTURE

```