Source Code for Control Unit of Electroporation Devices

Bob Van Elst

1 Generics.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

package generator is
--Vivado doesn't support generics in pacakges
--  generic(
-- --    state_counter_length: integer := 16;
-- --    number_counter_length: integer := 8;
-- --    burst_counter_length: integer := 8
--   var:integer := 10;
--  );

  constant state_counter_length: integer := 32;
  constant state_counter_bipol_length: integer := 16;
  constant number_counter_length: integer := 8;
  constant burst_counter_length: integer := 8;

  constant out_sleep: std_logic_vector(3 downto 0):= "0000";
  constant out_pos_prep: std_logic_vector(3 downto 0):= "0010";
  constant out_pos_gen: std_logic_vector(3 downto 0):= "1010";
  constant out_pos_dis: std_logic_vector(3 downto 0):= "0011";
  constant out_neg_prep: std_logic_vector(3 downto 0):= "0001";
  constant out_neg_gen: std_logic_vector(3 downto 0):= "0101";
  constant out_neg_dis: std_logic_vector(3 downto 0):= "0011";
  constant out_ICE: std_logic_vector(3 downto 0):= "0011";



  type state_type is (
      fsm_sleep, --Do nothing here
      fsm_pos_prep, --Charge for pulsing (positive)
      fsm_pos_gen, --Start pulsing (positive)
      fsm_pos_dis, --Discharge pulsing (positive)
      fsm_pause_mono, --Pause after positive pulsing
```

```
36          fsm_neg_prep, --Charge for pulsing (negative)
37          fsm_neg_gen, --Start pulsing (negative)
38          fsm_neg_dis, --Discharge pulsing (negative)
39          fsm_pause_bipol, --Pause after negative pulsing
40          fsm_pause_burst, --Pause between two pulse funcitons
41          fsm_ICE --In Case of Emergency state, discharge everything
42      );
43
44      --Monopolar Simple------------------------------------------------------
45      type mon_simple_in_type is record
46        GEN_START: std_logic;
47        pulse_number: std_logic_vector(number_counter_length - 1 downto 0);
48        burst_number: std_logic_vector(burst_counter_length - 1 downto 0);
49
50        pulse_pos_gen_duration: std_logic_vector(state_counter_length - 1
          downto 0);
51
52        pause_mono_duration: std_logic_vector(state_counter_length - 1 downto
          0);
53        pause_burst_duration: std_logic_vector(state_counter_length - 1
          downto 0);
54      end record;
55
56      type mon_simple_out_type is record
57        state_out: state_type;
58      end record;
59
60      component Monopolar_Simple
61        port(
62          clk, reset : in std_logic;
63          sys_i : in mon_simple_in_type;
64          sys_o : out mon_simple_out_type
65        );
66      end component;
67      --Bipolar Simple---------------------------------------------------------
68      type bipol_simple_in_type is record
69        GEN_START: std_logic;
70        pulse_number: std_logic_vector(number_counter_length - 1 downto 0);
71        burst_number: std_logic_vector(burst_counter_length - 1 downto 0);
72
73        pulse_pos_gen_duration: std_logic_vector(state_counter_length - 1
          downto 0);
74        pulse_neg_gen_duration: std_logic_vector(state_counter_length - 1
          downto 0);
75
76        pause_mono_duration: std_logic_vector(state_counter_length - 1 downto
          0);
```

```
77      pause_bipol_duration: std_logic_vector(state_counter_length - 1
        ↪  downto 0);
78      pause_burst_duration: std_logic_vector(state_counter_length - 1
        ↪  downto 0);
79    end record;
80
81    type bipol_simple_out_type is record
82      state_out: state_type;
83    end record;
84
85    component Bipolar_Simple
86      port(
87        clk, reset : in std_logic;
88        sys_i : in bipol_simple_in_type;
89        sys_o : out bipol_simple_out_type
90      );
91    end component;
92    --Monopolar
      ↪  Full-----------------------------------------------------
93    type mon_full_in_type is record
94      GEN_START: std_logic;
95      pulse_number: std_logic_vector(number_counter_length - 1 downto 0);
96      burst_number: std_logic_vector(burst_counter_length - 1 downto 0);
97      pulse_pos_prep_duration: std_logic_vector(state_counter_bipol_length
        ↪  - 1 downto 0);
98      pulse_pos_gen_duration: std_logic_vector(state_counter_length - 1
        ↪  downto 0);
99      pulse_pos_dis_duration: std_logic_vector(state_counter_bipol_length -
        ↪  1 downto 0);
100     pause_mono_duration: std_logic_vector(state_counter_length - 1 downto
        ↪  0);
101     pause_burst_duration: std_logic_vector(state_counter_length - 1
        ↪  downto 0);
102   end record;
103
104   type mon_full_out_type is record
105     state_out: state_type;
106   end record;
107
108   component Monopolar_Full
109     port(
110       clk, reset : in std_logic;
111       sys_i : in mon_full_in_type;
112       sys_o : out mon_full_out_type
113     );
114   end component;
115   --Bipolar Full-----------------------------------------------------
116   type bipol_full_in_type is record
117     GEN_START: std_logic;
118     pulse_number: std_logic_vector(number_counter_length - 1 downto 0);
```

```
119       burst_number: std_logic_vector(burst_counter_length - 1 downto 0);
120       pulse_pos_prep_duration: std_logic_vector(state_counter_bipol_length
          ↪  - 1 downto 0);
121       pulse_pos_gen_duration: std_logic_vector(state_counter_length - 1
          ↪  downto 0);
122       pulse_pos_dis_duration: std_logic_vector(state_counter_bipol_length -
          ↪  1 downto 0);
123       pulse_neg_prep_duration: std_logic_vector(state_counter_bipol_length
          ↪  - 1 downto 0);
124       pulse_neg_gen_duration: std_logic_vector(state_counter_length - 1
          ↪  downto 0);
125       pulse_neg_dis_duration: std_logic_vector(state_counter_bipol_length -
          ↪  1 downto 0);
126       pause_mono_duration: std_logic_vector(state_counter_length - 1 downto
          ↪  0);
127       pause_bipol_duration: std_logic_vector(state_counter_length - 1
          ↪  downto 0);
128       pause_burst_duration: std_logic_vector(state_counter_length - 1
          ↪  downto 0);
129     end record;
130
131     type bipol_full_out_type is record
132       state_out: state_type;
133     end record;
134
135     component Bipolar_Full
136       port(
137         clk, reset : in std_logic;
138         sys_i : in bipol_full_in_type;
139         sys_o : out bipol_full_out_type
140       );
141     end component;
142
143     --One-shot and
        ↪  lock-------------------------------------------------------
144     type latch_full_type is record
145       s: std_logic_vector(1 downto 0);
146       GEN_START: std_logic;
147       pulse_number: std_logic_vector(number_counter_length - 1 downto 0);
148       burst_number: std_logic_vector(burst_counter_length - 1 downto 0);
149       pulse_pos_prep_duration: std_logic_vector(state_counter_bipol_length
          ↪  - 1 downto 0);
150       pulse_pos_gen_duration: std_logic_vector(state_counter_length - 1
          ↪  downto 0);
151       pulse_pos_dis_duration: std_logic_vector(state_counter_bipol_length -
          ↪  1 downto 0);
152       pulse_neg_prep_duration: std_logic_vector(state_counter_bipol_length
          ↪  - 1 downto 0);
153       pulse_neg_gen_duration: std_logic_vector(state_counter_length - 1
          ↪  downto 0);
```

```
154       pulse_neg_dis_duration: std_logic_vector(state_counter_bipol_length -
          ↪  1 downto 0);
155       pause_mono_duration: std_logic_vector(state_counter_length - 1 downto
          ↪  0);
156       pause_bipol_duration: std_logic_vector(state_counter_length - 1
          ↪  downto 0);
157       pause_burst_duration: std_logic_vector(state_counter_length - 1
          ↪  downto 0);
158     end record;
159
160     component One_Shot_Latch
161       port(
162         clk, reset : in std_logic;
163         sys_i : in latch_full_type;
164         gen_mon_simple: out mon_simple_in_type;
165         gen_mon_full: out mon_full_in_type;
166         gen_bipol_simple: out bipol_simple_in_type;
167         gen_bipol_full: out bipol_full_in_type;
168         mux_select: out std_logic_vector(1 downto 0)
169       );
170     end component;
171
172     --State_MUX-------------------------------------------------------
173     component State_MUX
174       port(
175         mon_simple_in: in state_type;
176         mon_full_in: in state_type;
177         bi_simple_in: in state_type;
178         bi_full_in: in state_type;
179         s: in std_logic_vector(1 downto 0);
180         state_out: out state_type
181       );
182     end component;
183
184     --State
          ↪  Decoder------------------------------------------------------
185     type decoder_in_type is record
186       state_load: state_type;
187       ICE: std_logic;
188     end record;
189     type decoder_out_type is record
190       GEN_END_OUT: std_logic;
191       GEN_POS_PULSE: std_logic;
192       GEN_NEG_PULSE: std_logic;
193       MOS_DRIVER_OUT: std_logic_vector(3 downto 0);
194     end record;
195     component State_Decoder
196       port(
197         clk, reset : in std_logic;
198         sys_i: in decoder_in_type;
```

```
199         sys_o: out decoder_out_type
200       );
201     end component;
202   end package;
```

2 Generator_Wrapper.vhd

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4   use work.generator.all;
5
6
7
8   entity Gen_wrapper is
9   --  generic(
10  --    state_counter_length: integer := 16;
11  --    number_counter_length: integer := 8;
12  --    burst_counter_length: integer := 8
13  --  );
14
15    port(
16      Clk, Reset : in std_logic;
17      ICE_in: in std_logic;
18      GEN_START: in std_logic;
19      bipol_enable: in std_logic;
20      pre_post_enable: in std_logic;
21      pulse_number: in std_logic_vector(number_counter_length - 1 downto
        ↪  0);
22      burst_number: in std_logic_vector(burst_counter_length - 1 downto 0);
23
24      pause_mono_duration: in std_logic_vector(state_counter_length - 1
        ↪  downto 0);
25      pause_bipol_duration: in std_logic_vector(state_counter_length - 1
        ↪  downto 0);
26      pause_burst_duration: in std_logic_vector(state_counter_length - 1
        ↪  downto 0);
27
28      pulse_pos_gen_duration: in std_logic_vector(state_counter_length - 1
        ↪  downto 0);
29      pulse_neg_gen_duration: in std_logic_vector(state_counter_length - 1
        ↪  downto 0);
30
31      pulse_pos_prep_duration: in
        ↪  std_logic_vector(state_counter_bipol_length - 1 downto 0);
32      pulse_neg_prep_duration: in
        ↪  std_logic_vector(state_counter_bipol_length - 1 downto 0);
```

```
33
34      pulse_pos_dis_duration: in
        ↪  std_logic_vector(state_counter_bipol_length - 1 downto 0);
35      pulse_neg_dis_duration: in
        ↪  std_logic_vector(state_counter_bipol_length - 1 downto 0);
36
37
38      GEN_END_OUT: out std_logic;
39      GEN_POS_PULSE: out std_logic;
40      GEN_NEG_PULSE: out std_logic;
41      MOS_DRIVER_OUT: out std_logic_vector(3 downto 0)
42    );
43  end Gen_wrapper;
44
45  architecture rtl of Gen_wrapper is
46
47    signal connect_latch1_gen1: mon_simple_in_type;
48    signal connect_latch1_gen2: mon_full_in_type;
49    signal connect_latch1_gen3: bipol_simple_in_type;
50    signal connect_latch1_gen4: bipol_full_in_type;
51    signal connect_gen1_mux1: state_type;
52    signal connect_gen2_mux1: state_type;
53    signal connect_gen3_mux1: state_type;
54    signal connect_gen4_mux1: state_type;
55    signal connect_mux1_decoder1: state_type;
56    signal sleeping: state_type := fsm_sleep;
57  --  signal latched_params: latch_full_type;
58    signal latched_mux_select: std_logic_vector(1 downto 0);
59
60  begin
61    latch1: One_Shot_Latch
62      port map (
63        --inputs
64        clk => Clk,
65        reset => Reset,
66        sys_i.s(1) => bipol_enable,
67        sys_i.s(0) => pre_post_enable,
68        sys_i.GEN_START => GEN_START,
69        sys_i.pulse_number => pulse_number,
70        sys_i.burst_number => burst_number,
71        sys_i.pulse_pos_prep_duration => pulse_pos_prep_duration,
72        sys_i.pulse_pos_gen_duration => pulse_pos_gen_duration,
73        sys_i.pulse_pos_dis_duration => pulse_pos_dis_duration,
74        sys_i.pulse_neg_prep_duration => pulse_neg_prep_duration,
75        sys_i.pulse_neg_gen_duration => pulse_neg_gen_duration,
76        sys_i.pulse_neg_dis_duration => pulse_neg_dis_duration,
77        sys_i.pause_mono_duration => pause_mono_duration,
78        sys_i.pause_bipol_duration => pause_bipol_duration,
79        sys_i.pause_burst_duration => pause_burst_duration,
80        --putputs
```

```
81        gen_mon_simple => connect_latch1_gen1,
82        gen_mon_full => connect_latch1_gen2,
83        gen_bipol_simple => connect_latch1_gen3,
84        gen_bipol_full => connect_latch1_gen4,
85        mux_select => latched_mux_select
86      );
87
88    gen1: Monopolar_Simple
89      port map (
90        --inputs
91        clk => Clk,
92        reset => Reset,
93        sys_i => connect_latch1_gen1,
94  --      sys_i.GEN_START => latched_params.GEN_START,
95  --      sys_i.pulse_number => latched_params.pulse_number,
96  --      sys_i.burst_number => latched_params.burst_number,
97  --      sys_i.pulse_pos_gen_duration =>
    ↪  latched_params.pulse_pos_gen_duration,
98  --      sys_i.pause_mono_duration => latched_params.pause_mono_duration,
99  --      sys_i.pause_burst_duration =>
    ↪  latched_params.pause_burst_duration,
100       --outputs
101       sys_o.state_out => connect_gen1_mux1
102     );
103   gen2: Monopolar_Full
104     port map (
105       --inputs
106       clk => Clk,
107       reset => Reset,
108       sys_i => connect_latch1_gen2,
109 --      sys_i.GEN_START => latched_params.GEN_START,
110 --      sys_i.pulse_number => latched_params.pulse_number,
111 --      sys_i.burst_number => latched_params.burst_number,
112 --      sys_i.pulse_pos_prep_duration =>
    ↪  latched_params.pulse_pos_prep_duration,
113 --      sys_i.pulse_pos_gen_duration =>
    ↪  latched_params.pulse_pos_gen_duration,
114 --      sys_i.pulse_pos_dis_duration =>
    ↪  latched_params.pulse_pos_dis_duration,
115 --      sys_i.pause_mono_duration => latched_params.pause_mono_duration,
116 --      sys_i.pause_burst_duration =>
    ↪  latched_params.pause_burst_duration,
117       --outputs
118       sys_o.state_out => connect_gen2_mux1
119     );
120   gen3: Bipolar_Simple
121     port map (
122       --inputs
123       clk => Clk,
124       reset => Reset,
```

```
125         sys_i => connect_latch1_gen3,
126   --      sys_i.GEN_START => latched_params.GEN_START,
127   --      sys_i.pulse_number => latched_params.pulse_number,
128   --      sys_i.burst_number => latched_params.burst_number,
129   --      sys_i.pulse_pos_gen_duration =>
      ↪  latched_params.pulse_pos_gen_duration,
130   --      sys_i.pulse_neg_gen_duration =>
      ↪  latched_params.pulse_neg_gen_duration,
131   --      sys_i.pause_mono_duration => latched_params.pause_mono_duration,
132   --      sys_i.pause_bipol_duration =>
      ↪  latched_params.pause_bipol_duration,
133   --      sys_i.pause_burst_duration =>
      ↪  latched_params.pause_burst_duration,
134         --outputs
135         sys_o.state_out => connect_gen3_mux1
136       );
137     gen4: Bipolar_Full
138       port map (
139         --inputs
140         clk => Clk,
141         reset => Reset,
142         sys_i => connect_latch1_gen4,
143   --      sys_i.GEN_START => latched_params.GEN_START,
144   --      sys_i.pulse_number => latched_params.pulse_number,
145   --      sys_i.burst_number => latched_params.burst_number,
146   --      sys_i.pulse_pos_prep_duration =>
      ↪  latched_params.pulse_pos_prep_duration,
147   --      sys_i.pulse_pos_gen_duration =>
      ↪  latched_params.pulse_pos_gen_duration,
148   --      sys_i.pulse_pos_dis_duration =>
      ↪  latched_params.pulse_pos_dis_duration,
149   --      sys_i.pulse_neg_prep_duration =>
      ↪  latched_params.pulse_neg_prep_duration,
150   --      sys_i.pulse_neg_gen_duration =>
      ↪  latched_params.pulse_neg_gen_duration,
151   --      sys_i.pulse_neg_dis_duration =>
      ↪  latched_params.pulse_neg_dis_duration,
152   --      sys_i.pause_mono_duration => latched_params.pause_mono_duration,
153   --      sys_i.pause_bipol_duration =>
      ↪  latched_params.pause_bipol_duration,
154   --      sys_i.pause_burst_duration =>
      ↪  latched_params.pause_burst_duration,
155         --outputs
156         sys_o.state_out => connect_gen4_mux1
157       );
158     mux1: State_MUX
159       port map(
160         --inputs
161         mon_simple_in => connect_gen1_mux1,
162         mon_full_in => connect_gen2_mux1,
```

```
163         bi_simple_in => connect_gen3_mux1,
164         bi_full_in => connect_gen4_mux1,
165         s => latched_mux_select,
166         --outputs
167         state_out => connect_mux1_decoder1
168       );
169
170     decoder1: State_Decoder
171       port map(
172         --inputs
173         clk => Clk,
174         reset => Reset,
175         sys_i.state_load => connect_mux1_decoder1,
176         sys_i.ICE => ICE_in,
177         --outputs
178         sys_o.GEN_END_OUT => GEN_END_OUT,
179         sys_o.GEN_POS_PULSE => GEN_POS_PULSE,
180         sys_o.GEN_NEG_PULSE => GEN_NEG_PULSE,
181         sys_o.MOS_DRIVER_OUT => MOS_DRIVER_OUT
182       );
183
184 end rtl;
```

3 One_shot_latch.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.generator.all;
5
6
7  entity One_Shot_Latch is
8    port(
9      clk, reset : in std_logic;
10     sys_i : in latch_full_type;
11     gen_mon_simple: out mon_simple_in_type;
12     gen_mon_full: out mon_full_in_type;
13     gen_bipol_simple: out bipol_simple_in_type;
14     gen_bipol_full: out bipol_full_in_type;
15     mux_select: out std_logic_vector(1 downto 0)
16   );
17 end One_Shot_Latch;
18
19 architecture Behavioral of One_Shot_Latch is
20
21   type reg_type is record
22     -- load: std_logic;
```

```
    latched_outputs: latch_full_type;
    gen_mon_simple: mon_simple_in_type;
    gen_mon_full: mon_full_in_type;
    gen_bipol_simple: bipol_simple_in_type;
    gen_bipol_full: bipol_full_in_type;
    trigger_flag: std_logic;
    trigger_start: std_logic;
  end record;

  constant init_mon_simple: mon_simple_in_type := (
    GEN_START => '0',
    pulse_number => (others => '0'),
    burst_number => (others => '0'),
    pulse_pos_gen_duration => (others => '0'),
    pause_mono_duration => (others => '0'),
    pause_burst_duration => (others => '0')
  );
  constant init_mon_full: mon_full_in_type := (
    GEN_START => '0',
    pulse_number => (others => '0'),
    burst_number => (others => '0'),
    pulse_pos_prep_duration => (others => '0'),
    pulse_pos_gen_duration => (others => '0'),
    pulse_pos_dis_duration => (others => '0'),
    pause_mono_duration => (others => '0'),
    pause_burst_duration => (others => '0')
  );
  constant init_bipol_simple: bipol_simple_in_type := (
    GEN_START => '0',
    pulse_number => (others => '0'),
    burst_number => (others => '0'),
    pulse_pos_gen_duration => (others => '0'),
    pulse_neg_gen_duration => (others => '0'),
    pause_mono_duration => (others => '0'),
    pause_bipol_duration => (others => '0'),
    pause_burst_duration => (others => '0')
  );
  constant init_bipol_full: bipol_full_in_type := (
    GEN_START => '0',
    pulse_number => (others => '0'),
    burst_number => (others => '0'),
    pulse_pos_prep_duration => (others => '0'),
    pulse_pos_gen_duration => (others => '0'),
    pulse_pos_dis_duration => (others => '0'),
    pulse_neg_prep_duration => (others => '0'),
    pulse_neg_gen_duration => (others => '0'),
    pulse_neg_dis_duration => (others => '0'),
    pause_mono_duration => (others => '0'),
    pause_bipol_duration => (others => '0'),
    pause_burst_duration => (others => '0')
```

```
73    );
74
75    constant init: reg_type := (
76      latched_outputs => (
77        s => "00",
78        GEN_START => '0',
79        pulse_number => (others => '0'),
80        burst_number => (others => '0'),
81        pulse_pos_prep_duration => (others => '0'),
82        pulse_pos_gen_duration => (others => '0'),
83        pulse_pos_dis_duration => (others => '0'),
84        pulse_neg_prep_duration => (others => '0'),
85        pulse_neg_gen_duration => (others => '0'),
86        pulse_neg_dis_duration => (others => '0'),
87        pause_mono_duration => (others => '0'),
88        pause_bipol_duration => (others => '0'),
89        pause_burst_duration => (others => '0')
90      ),
91      gen_mon_simple => init_mon_simple,
92      gen_mon_full => init_mon_full,
93      gen_bipol_simple => init_bipol_simple,
94      gen_bipol_full => init_bipol_full,
95      trigger_flag => '0',
96      trigger_start => '0'
97    );
98
99    signal r, rin : reg_type := init;
100
101 begin
102   comb: process(sys_i, r, reset)
103   variable v : reg_type;
104   begin
105     v := r;
106
      ↪  --=============================================================================================
107     --One-shot trigger
108     if r.trigger_flag = '0' AND sys_i.GEN_START = '1' then
109       v.trigger_start := '1';
110     else
111       v.trigger_start := '0';
112     end if;
113
      ↪  --=============================================================================================
114     --latch the values
115
116     if r.trigger_start = '1' then
117       v.latched_outputs := sys_i;
118       v.latched_outputs.GEN_START := '1'; --for redundancy in case
          ↪  GEN_START goes low very quickly.
119     else
```

```
120         v.latched_outputs := v.latched_outputs;
121         v.latched_outputs.GEN_START := '0'; --need this for consecutive
            ↪ Generations.
122       end if;
123
        ↪ --=============================================================================================
124       --reset system
125       if reset = '1' then
126         v := init;
127       end if;
128
        ↪ --=============================================================================================
129       mux_select <= r.latched_outputs.s;
130       case( r.latched_outputs.s) is
131         when "00" => --Monopolar simple
132           --set gen values
133           v.gen_mon_simple.GEN_START := r.latched_outputs.GEN_START;
134           v.gen_mon_simple.pulse_number := r.latched_outputs.pulse_number;
135           v.gen_mon_simple.burst_number := r.latched_outputs.burst_number;
136           v.gen_mon_simple.pulse_pos_gen_duration :=
              ↪ r.latched_outputs.pulse_pos_gen_duration;
137           v.gen_mon_simple.pause_mono_duration :=
              ↪ r.latched_outputs.pause_mono_duration;
138           v.gen_mon_simple.pause_burst_duration :=
              ↪ r.latched_outputs.pause_burst_duration;
139           --zero others
140           v.gen_mon_full := init_mon_full;
141           v.gen_bipol_simple := init_bipol_simple;
142           v.gen_bipol_full := init_bipol_full;
143         when "01" => --Monopolar full
144           --set gen values
145           v.gen_mon_full.GEN_START := r.latched_outputs.GEN_START;
146           v.gen_mon_full.pulse_number := r.latched_outputs.pulse_number;
147           v.gen_mon_full.burst_number := r.latched_outputs.burst_number;
148           v.gen_mon_full.pulse_pos_prep_duration :=
              ↪ r.latched_outputs.pulse_pos_prep_duration;
149           v.gen_mon_full.pulse_pos_gen_duration :=
              ↪ r.latched_outputs.pulse_pos_gen_duration;
150           v.gen_mon_full.pulse_pos_dis_duration :=
              ↪ r.latched_outputs.pulse_pos_dis_duration;
151           v.gen_mon_full.pause_mono_duration :=
              ↪ r.latched_outputs.pause_mono_duration;
152           v.gen_mon_full.pause_burst_duration :=
              ↪ r.latched_outputs.pause_burst_duration;
153           --zero others
154           v.gen_mon_simple := init_mon_simple;
155           v.gen_bipol_simple := init_bipol_simple;
156           v.gen_bipol_full := init_bipol_full;
157         when "10" => --Bipolar simple
158           --set gen values
```

```
159           v.gen_bipol_simple.GEN_START := r.latched_outputs.GEN_START;
160           v.gen_bipol_simple.pulse_number :=
              ↪  r.latched_outputs.pulse_number;
161           v.gen_bipol_simple.burst_number :=
              ↪  r.latched_outputs.burst_number;
162           v.gen_bipol_simple.pulse_pos_gen_duration :=
              ↪  r.latched_outputs.pulse_pos_gen_duration;
163           v.gen_bipol_simple.pulse_neg_gen_duration :=
              ↪  r.latched_outputs.pulse_neg_gen_duration;
164           v.gen_bipol_simple.pause_mono_duration :=
              ↪  r.latched_outputs.pause_mono_duration;
165           v.gen_bipol_simple.pause_bipol_duration :=
              ↪  r.latched_outputs.pause_bipol_duration;
166           v.gen_bipol_simple.pause_burst_duration :=
              ↪  r.latched_outputs.pause_burst_duration;
167           --zero others
168           v.gen_mon_simple := init_mon_simple;
169           v.gen_mon_full := init_mon_full;
170           v.gen_bipol_full := init_bipol_full;
171         when "11" => --Bipolar full
172           --set gen values
173           v.gen_bipol_full.GEN_START := r.latched_outputs.GEN_START;
174           v.gen_bipol_full.pulse_number := r.latched_outputs.pulse_number;
175           v.gen_bipol_full.burst_number := r.latched_outputs.burst_number;
176           v.gen_bipol_full.pulse_pos_prep_duration :=
              ↪  r.latched_outputs.pulse_pos_prep_duration;
177           v.gen_bipol_full.pulse_pos_gen_duration :=
              ↪  r.latched_outputs.pulse_pos_gen_duration;
178           v.gen_bipol_full.pulse_pos_dis_duration :=
              ↪  r.latched_outputs.pulse_pos_dis_duration;
179           v.gen_bipol_full.pulse_neg_prep_duration :=
              ↪  r.latched_outputs.pulse_neg_prep_duration;
180           v.gen_bipol_full.pulse_neg_gen_duration :=
              ↪  r.latched_outputs.pulse_neg_gen_duration;
181           v.gen_bipol_full.pulse_neg_dis_duration :=
              ↪  r.latched_outputs.pulse_neg_dis_duration;
182           v.gen_bipol_full.pause_mono_duration :=
              ↪  r.latched_outputs.pause_mono_duration;
183           v.gen_bipol_full.pause_bipol_duration :=
              ↪  r.latched_outputs.pause_bipol_duration;
184           v.gen_bipol_full.pause_burst_duration :=
              ↪  r.latched_outputs.pause_burst_duration;
185           --zero others
186           v.gen_mon_simple := init_mon_simple;
187           v.gen_mon_full := init_mon_full;
188           v.gen_bipol_simple := init_bipol_simple;
189         when others =>
190           v.gen_mon_simple := init_mon_simple;
191           v.gen_mon_full := init_mon_full;
192           v.gen_bipol_simple := init_bipol_simple;
```

```
193           v.gen_bipol_full := init_bipol_full;
194       end case;
195
196       gen_mon_simple <= r.gen_mon_simple;
197       gen_mon_full <= r.gen_mon_full;
198       gen_bipol_simple <= r.gen_bipol_simple;
199       gen_bipol_full <= r.gen_bipol_full;
200       rin <= v;
201     end process;
202
203     reg: process(clk)
204     begin
205       if rising_edge(clk) then
206         r <= rin;
207         r.trigger_flag <= sys_i.GEN_START; --update the init trigger
208
209       end if;
210     end process;
211
212 end Behavioral;
```

4 Monopolar_Simple.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use work.generator.all;
5
6  entity Monopolar_Simple is
7    port (
8      clk, reset : in std_logic;
9      sys_i : in mon_simple_in_type;
10     sys_o : out mon_simple_out_type
11   );
12 end Monopolar_Simple;
13
14 architecture Behavioral of Monopolar_Simple is
15
16   type reg_type is record
17     state: state_type;
18     counter_pos_gen:
       ↪  std_logic_vector(sys_i.pulse_pos_gen_duration'range);
19     counter_pause_mono:
       ↪  std_logic_vector(sys_i.pause_mono_duration'range);
20     counter_pause_burst:
       ↪  std_logic_vector(sys_i.pause_burst_duration'range);
21     number_counter: std_logic_vector(sys_i.pulse_number'range);
```

```
22      burst_counter : std_logic_vector(sys_i.burst_number'range);
23      trigger_flag, trigger_start: std_logic;
24    end record;
25
26    constant init : reg_type := (
27      state => fsm_sleep,
28      counter_pos_gen => (others => '0'),
29      counter_pause_mono => (others => '0'),
30      counter_pause_burst => (others => '0'),
31      number_counter => (others => '0'),
32      burst_counter => (others => '0'),
33      trigger_flag => '0',
34      trigger_start => '0'
35    );
36
37    --main signals
38    signal r, rin: reg_type := init;
39
40    subtype slv is std_logic_vector; -- abbreviation
41
42  begin
43  ------------------------------------------------------------------------------------------------------------
44    comb: process(reset, r, sys_i)
45    variable v: reg_type;
46    begin
47
      ↪  --=============================================================================================
48      --Copy current state to variable
49      v := r;
50
      ↪  --=============================================================================================
51      --State Machine
52      case r.state is
53        --Sleep State
54        when fsm_sleep =>
55          if r.trigger_start = '1' then
56            v.state := fsm_pos_gen;
57          else
58           v := init; --stay sleeping
59          end if;
60        --Positive Generation
61        when fsm_pos_gen =>
62          if signed(r.counter_pos_gen) >=
            ↪  signed(sys_i.pulse_pos_gen_duration) - 1 then
63            v.counter_pos_gen := (others => '0');
64            v.number_counter := slv(signed(v.number_counter) + 1);
              ↪  --increment counter in monopolar mode
65            if signed(v.number_counter) >= signed(sys_i.pulse_number) then
66              if signed(r.burst_counter) >= signed(sys_i.burst_number) - 1
                ↪  then
```

```
67                v.state := fsm_sleep;
68              else
69                v.state := fsm_pause_burst; --Don't like this------
70              end if;
71            else
72              v.state := fsm_pause_mono;
73            end if;
74          else
75            v.counter_pos_gen := slv(signed(v.counter_pos_gen) + 1);
              ↪  --increment state counter
76          end if;
77        --Pause Monopolar
78        when fsm_pause_mono =>
79          if signed(r.counter_pause_mono) >=
            ↪  signed(sys_i.pause_mono_duration) - 1 then
80            v.counter_pause_mono := (others => '0');
81            v.state := fsm_pos_gen;
82          else
83            v.counter_pause_mono := slv(signed(v.counter_pause_mono) + 1);
              ↪  --increment state counter
84          end if;
85        --Pause Burst
86        when fsm_pause_burst =>
87          if signed(r.counter_pause_burst) >=
            ↪  signed(sys_i.pause_burst_duration) - 1 then
88            v.counter_pause_burst := (others => '0');
89            v.number_counter := (others => '0');
90            v.burst_counter := slv(signed(v.burst_counter) + 1);
              ↪  --increment burst counter
91            v.state := fsm_pos_gen;
92          else
93            v.counter_pause_burst := slv(signed(v.counter_pause_burst) +
              ↪  1); --increment state counter
94          end if;
95        --Should never come here.
96        when others =>
97          null;
98      end case;
99
    ↪  --=============================================================================================
100     --One-shot trigger
101     if r.trigger_flag = '0' AND sys_i.GEN_START = '1' then
102       v.trigger_start := '1';
103     else
104       v.trigger_start := '0';
105     end if;
106
    ↪  --=============================================================================================
107     --reset system
108     if reset = '1' then
```

```
109         v := init;
110       end if;
111
    ↪   --=============================================================================================
112       sys_o.state_out <= r.state; --set output
113       rin <= v; --update next state
114
    ↪   --=============================================================================================
115     end process comb;
116   ---------------------------------------------------------------------------------------------------
117     regs: process(clk)
118     begin
119       if rising_edge(clk) then
120         r <= rin;
121         r.trigger_flag <= sys_i.GEN_START;
122       end if;
123     end process regs;
124   ---------------------------------------------------------------------------------------------------
125
126   end Behavioral;
```

5 Monopolar_full.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4   use work.generator.all;
5
6   entity Monopolar_Full is
7     port(
8       clk, reset : in std_logic;
9       sys_i : in mon_full_in_type;
10      sys_o : out mon_full_out_type
11    );
12
13  end Monopolar_Full;
14
15  architecture Behavioral of Monopolar_Full is
16    type reg_type is record
17      state: state_type;
18      counter_pos_prep:
        ↪  std_logic_vector(sys_i.pulse_pos_prep_duration'range);
19      counter_pos_gen:
        ↪  std_logic_vector(sys_i.pulse_pos_gen_duration'range);
20      counter_pos_dis:
        ↪  std_logic_vector(sys_i.pulse_pos_dis_duration'range);
21      counter_pause_mono:
        ↪  std_logic_vector(sys_i.pause_mono_duration'range);
```

```
22      counter_pause_burst:
        ↪  std_logic_vector(sys_i.pause_burst_duration'range);
23      number_counter: std_logic_vector(sys_i.pulse_number'range);
24      burst_counter : std_logic_vector(sys_i.burst_number'range);
25      trigger_flag, trigger_start: std_logic;
26    end record;
27
28    constant init : reg_type := (
29      state => fsm_sleep,
30      counter_pos_gen => (others => '0'),
31      counter_pos_prep => (others => '0'),
32      counter_pos_dis => (others => '0'),
33      counter_pause_mono => (others => '0'),
34      counter_pause_burst => (others => '0'),
35      number_counter => (others => '0'),
36      burst_counter => (others => '0'),
37      trigger_flag => '0',
38      trigger_start => '0'
39    );
40
41    --main signals
42    signal r, rin: reg_type := init;
43
44    subtype slv is std_logic_vector; -- abbreviation
45
46    begin
47
      ↪  ------------------------------------------------------------------------------------------
48      comb: process(reset, r, sys_i)
49      variable v: reg_type;
50      begin
51
        ↪  --=======================================================================================
52        --Copy current state to variable
53        v := r;
54
        ↪  --=======================================================================================
55        --State Machine
56        case r.state is
57          --Sleep State
58          when fsm_sleep =>
59            if r.trigger_start = '1' then
60              v.state := fsm_pos_prep;
61            else
62             v := init; --stay sleeping
63            end if;
64          --Positive Preparation
65          when fsm_pos_prep =>
66            if unsigned(r.counter_pos_prep) >=
              ↪  unsigned(sys_i.pulse_pos_prep_duration) - 1 then
```

```
67              v.counter_pos_prep := (others => '0');
68              v.state := fsm_pos_gen;
69            else
70              v.counter_pos_prep := slv(signed(v.counter_pos_prep) + 1);
                ↪  --increment state counter
71            end if;
72          --Positive Generation
73          when fsm_pos_gen =>
74            if signed(r.counter_pos_gen) >=
              ↪  signed(sys_i.pulse_pos_gen_duration) - 1 then
75              v.counter_pos_gen := (others => '0');
76              v.state := fsm_pos_dis;
77            else
78              v.counter_pos_gen := slv(signed(v.counter_pos_gen) + 1);
                ↪  --increment state counter
79            end if;
80          --Positive Discharge
81          when fsm_pos_dis =>
82            if signed(r.counter_pos_dis) >=
              ↪  signed(sys_i.pulse_pos_dis_duration) - 1 then
83              v.counter_pos_dis := (others => '0');
84              v.number_counter := slv(signed(v.number_counter) + 1);
                ↪  --increment counter in monopolar mode
85              if signed(v.number_counter) >= signed(sys_i.pulse_number)
                ↪  then
86                if signed(r.burst_counter) >= signed(sys_i.burst_number) -
                  ↪  1 then
87                  v.state := fsm_sleep;
88                else
89                  v.state := fsm_pause_burst; --Don't like this------
90                end if;
91              else
92                v.state := fsm_pause_mono;
93              end if;
94            else
95              v.counter_pos_dis := slv(signed(v.counter_pos_dis) + 1);
                ↪  --increment state counter
96            end if;
97          --Pause Monopolar
98          when fsm_pause_mono =>
99            if signed(r.counter_pause_mono) >=
              ↪  signed(sys_i.pause_mono_duration) - 1 then
100             v.counter_pause_mono := (others => '0');
101             v.state := fsm_pos_prep;
102           else
103             v.counter_pause_mono := slv(signed(v.counter_pause_mono) +
                ↪  1); --increment state counter
104           end if;
105         --Pause Burst
106         when fsm_pause_burst =>
```

```
107             if signed(r.counter_pause_burst) >=
                ↪  signed(sys_i.pause_burst_duration) - 1 then
108               v.counter_pause_burst := (others => '0');
109               v.number_counter := (others => '0');
110               v.burst_counter := slv(signed(v.burst_counter) + 1);
                  ↪  --increment burst counter
111               v.state := fsm_pos_prep;
112             else
113               v.counter_pause_burst := slv(signed(v.counter_pause_burst) +
                  ↪  1); --increment state counter
114             end if;
115           --Should never come here.
116           when others =>
117             null;
118         end case;
119
          ↪  --=============================================================================================
120         --One-shot trigger
121         if r.trigger_flag = '0' AND sys_i.GEN_START = '1' then
122           v.trigger_start := '1';
123         else
124           v.trigger_start := '0';
125         end if;
126
          ↪  --=============================================================================================
127         --reset system
128         if reset = '1' then
129           v := init;
130         end if;
131
          ↪  --=============================================================================================
132         sys_o.state_out <= r.state; --set output
133         rin <= v; --update next state
134
          ↪  --=============================================================================================
135       end process comb;
136
        ↪  -----------------------------------------------------------------------------------------------
137       regs: process(clk)
138       begin
139
          ↪  --=============================================================================================
140         if rising_edge(clk) then
141           r <= rin;
142           r.trigger_flag <= sys_i.GEN_START;
143         end if;
144
          ↪  --=============================================================================================
145       end process regs;
```

```
146
    ↪  ----------------------------------------------------------------------------------------------------
147   end Behavioral;
```

6 Bipolar_simple.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4   use work.generator.ALL;
5
6   entity Bipolar_Simple is
7     port(
8       clk, reset : in std_logic;
9       sys_i : in bipol_simple_in_type;
10      sys_o : out bipol_simple_out_type
11    );
12  end Bipolar_Simple;
13
14  architecture Behavioral of Bipolar_Simple is
15
16    type reg_type is record
17      state: state_type;
18      counter_pos_gen:
        ↪  std_logic_vector(sys_i.pulse_pos_gen_duration'range);
19      counter_neg_gen:
        ↪  std_logic_vector(sys_i.pulse_pos_gen_duration'range);
20      counter_pause_mono:
        ↪  std_logic_vector(sys_i.pause_mono_duration'range);
21      counter_pause_bipol:
        ↪  std_logic_vector(sys_i.pause_mono_duration'range);
22      counter_pause_burst:
        ↪  std_logic_vector(sys_i.pause_burst_duration'range);
23      number_counter: std_logic_vector(sys_i.pulse_number'range);
24      burst_counter : std_logic_vector(sys_i.burst_number'range);
25      trigger_flag, trigger_start: std_logic;
26    end record;
27
28    constant init : reg_type := (
29      state => fsm_sleep,
30      counter_pos_gen => (others => '0'),
31      counter_neg_gen => (others => '0'),
32      counter_pause_mono => (others => '0'),
33      counter_pause_bipol => (others => '0'),
34      counter_pause_burst => (others => '0'),
35      number_counter => (others => '0'),
36      burst_counter => (others => '0'),
```

```
37      trigger_flag => '0',
38      trigger_start => '0'
39    );
40
41    --main signals
42    signal r, rin: reg_type := init;
43
44    subtype slv is std_logic_vector; -- abbreviation
45
46  begin
47  ------------------------------------------------------------------------------------------------------------
48    comb: process(reset, r, sys_i)
49    variable v: reg_type;
50    begin
51
      ↪  --=========================================================================================================
52      --Copy current state to variable
53      v := r;
54
      ↪  --=========================================================================================================
55      --State Machine
56      case r.state is
57        --Sleep State
58        when fsm_sleep =>
59          if r.trigger_start = '1' then
60            v.state := fsm_pos_gen;
61          else
62           v := init; --stay sleeping
63          end if;
64        --Positive Generation
65        when fsm_pos_gen =>
66          if signed(r.counter_pos_gen) >=
            ↪  signed(sys_i.pulse_pos_gen_duration) - 1 then
67            v.counter_pos_gen := (others => '0');
68            v.state := fsm_pause_mono;
69          else
70            v.counter_pos_gen := slv(signed(v.counter_pos_gen) + 1);
              ↪  --increment state counter
71          end if;
72        --Pause Monopolar
73        when fsm_pause_mono =>
74          if signed(r.counter_pause_mono) >=
            ↪  signed(sys_i.pause_mono_duration) - 1 then
75            v.counter_pause_mono := (others => '0');
76            v.state := fsm_neg_gen;
77          else
78            v.counter_pause_mono := slv(signed(v.counter_pause_mono) + 1);
              ↪  --increment state counter
79          end if;
80        --Negative Generation
```

```
81          when fsm_neg_gen =>
82            if signed(r.counter_neg_gen) >=
              ↪  signed(sys_i.pulse_neg_gen_duration) - 1 then
83              v.counter_neg_gen := (others => '0');
84              v.number_counter := slv(signed(v.number_counter) + 1);
                ↪  --increment counter in monopolar mode
85              if signed(v.number_counter) >= signed(sys_i.pulse_number) then
86                if signed(r.burst_counter) >= signed(sys_i.burst_number) - 1
                  ↪  then
87                  v.state := fsm_sleep;
88                else
89                  v.state := fsm_pause_burst; --Don't like this------
90                end if;
91              else
92                v.state := fsm_pause_bipol;
93              end if;
94            else
95              v.counter_neg_gen := slv(signed(v.counter_neg_gen) + 1);
                ↪  --increment state counter
96            end if;
97          --Pause Bipolar
98          when fsm_pause_bipol =>
99            if signed(r.counter_pause_bipol) >=
              ↪  signed(sys_i.pause_bipol_duration) - 1 then
100             v.counter_pause_bipol := (others => '0');
101             v.state := fsm_pos_gen;
102           else
103             v.counter_pause_bipol := slv(signed(v.counter_pause_bipol) +
                ↪  1); --increment state counter
104           end if;
105         --Pause Burst
106         when fsm_pause_burst =>
107           if signed(r.counter_pause_burst) >=
              ↪  signed(sys_i.pause_burst_duration) - 1 then
108             v.counter_pause_burst := (others => '0');
109             v.number_counter := (others => '0');
110             v.burst_counter := slv(signed(v.burst_counter) + 1);
                ↪  --increment burst counter
111             v.state := fsm_pos_gen;
112           else
113             v.counter_pause_burst := slv(signed(v.counter_pause_burst) +
                ↪  1); --increment state counter
114           end if;
115         --Should never come here.
116         when others =>
117           null;
118       end case;
119
    ↪  --=============================================================================================
120       --One-shot trigger
```

```
121       if r.trigger_flag = '0' AND sys_i.GEN_START = '1' then
122         v.trigger_start := '1';
123       else
124         v.trigger_start := '0';
125       end if;
126 
    ↪   --=============================================================================================
127       --reset system
128       if reset = '1' then
129         v := init;
130       end if;
131 
    ↪   --=============================================================================================
132       sys_o.state_out <= r.state; --set output
133       rin <= v; --update next state
134 
    ↪   --=============================================================================================
135     end process comb;
136   ---------------------------------------------------------------------------------------------------
137     regs: process(clk)
138     begin
139       if rising_edge(clk) then
140         r <= rin;
141         r.trigger_flag <= sys_i.GEN_START;
142       end if;
143     end process regs;
144   ---------------------------------------------------------------------------------------------------
145 
146   end Behavioral;
```

7 Bipolar_full.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4   use work.generator.ALL;
5 
6   entity Bipolar_full is
7     port(
8       clk, reset : in std_logic;
9       sys_i : in bipol_full_in_type;
10      sys_o : out bipol_full_out_type
11    );
12  end Bipolar_full;
13 
14  architecture Behavioral of Bipolar_full is
15 
```

```
16    type reg_type is record
17    state: state_type;
18    counter_pos_prep:
      ↪  std_logic_vector(sys_i.pulse_pos_prep_duration'range);
19    counter_pos_gen: std_logic_vector(sys_i.pulse_pos_gen_duration'range);
20    counter_pos_dis: std_logic_vector(sys_i.pulse_pos_dis_duration'range);
21    counter_neg_prep:
      ↪  std_logic_vector(sys_i.pulse_neg_prep_duration'range);
22    counter_neg_gen: std_logic_vector(sys_i.pulse_neg_gen_duration'range);
23    counter_neg_dis: std_logic_vector(sys_i.pulse_neg_dis_duration'range);
24    counter_pause_mono: std_logic_vector(sys_i.pause_mono_duration'range);
25    counter_pause_bipol: std_logic_vector(sys_i.pause_mono_duration'range);
26    counter_pause_burst:
      ↪  std_logic_vector(sys_i.pause_burst_duration'range);
27    number_counter: std_logic_vector(sys_i.pulse_number'range);
28    burst_counter : std_logic_vector(sys_i.burst_number'range);
29    trigger_flag, trigger_start: std_logic;
30    end record;
31
32    constant init : reg_type := (
33      state => fsm_sleep,
34      counter_pos_gen => (others => '0'),
35      counter_pos_prep => (others => '0'),
36      counter_pos_dis => (others => '0'),
37      counter_neg_gen => (others => '0'),
38      counter_neg_prep => (others => '0'),
39      counter_neg_dis => (others => '0'),
40      counter_pause_mono => (others => '0'),
41      counter_pause_bipol => (others => '0'),
42      counter_pause_burst => (others => '0'),
43      number_counter => (others => '0'),
44      burst_counter => (others => '0'),
45      trigger_flag => '0',
46      trigger_start => '0'
47    );
48
49    --main signals
50    signal r, rin: reg_type := init;
51
52    subtype slv is std_logic_vector; -- abbreviation
53
54    begin
55
      ↪  ------------------------------------------------------------------------------------------------
56      comb: process(reset, r, sys_i)
57      variable v: reg_type;
58      begin
59
        ↪  --=============================================================================================
60        --Copy current state to variable
```

```
61          v := r;
62
        ↪   --=============================================================================================
63          --State Machine
64          case r.state is
65            --Sleep State
66            when fsm_sleep =>
67              if r.trigger_start = '1' then
68                v.state := fsm_pos_prep;
69              else
70               v := init; --stay sleeping
71              end if;
72            --Positive Preparation
73            when fsm_pos_prep =>
74              if unsigned(r.counter_pos_prep) >=
                ↪  unsigned(sys_i.pulse_pos_prep_duration) - 1 then
75                v.counter_pos_prep := (others => '0');
76                v.state := fsm_pos_gen;
77              else
78                v.counter_pos_prep := slv(signed(v.counter_pos_prep) + 1);
                  ↪  --increment state counter
79              end if;
80            --Positive Generation
81            when fsm_pos_gen =>
82              if signed(r.counter_pos_gen) >=
                ↪  signed(sys_i.pulse_pos_gen_duration) - 1 then
83                v.counter_pos_gen := (others => '0');
84                v.state := fsm_pos_dis;
85              else
86                v.counter_pos_gen := slv(signed(v.counter_pos_gen) + 1);
                  ↪  --increment state counter
87              end if;
88            --Positive Discharge
89            when fsm_pos_dis =>
90              if signed(r.counter_pos_dis) >=
                ↪  signed(sys_i.pulse_pos_dis_duration) - 1 then
91                v.counter_pos_dis := (others => '0');
92                v.state := fsm_pause_mono;
93              else
94                v.counter_pos_dis := slv(signed(v.counter_pos_dis) + 1);
                  ↪  --increment state counter
95              end if;
96            --Pause Monopolar
97            when fsm_pause_mono =>
98              if signed(r.counter_pause_mono) >=
                ↪  signed(sys_i.pause_mono_duration) - 1 then
99                v.counter_pause_mono := (others => '0');
100               v.state := fsm_neg_prep;
101             else
```

```
102             v.counter_pause_mono := slv(signed(v.counter_pause_mono) +
                ↪  1); --increment state counter
103           end if;
104         --Negative Preparation
105         when fsm_neg_prep =>
106           if unsigned(r.counter_neg_prep) >=
              ↪  unsigned(sys_i.pulse_neg_prep_duration) - 1 then
107             v.counter_neg_prep := (others => '0');
108             v.state := fsm_neg_gen;
109           else
110             v.counter_neg_prep := slv(signed(v.counter_neg_prep) + 1);
                ↪  --increment state counter
111           end if;
112         --Negative Generation
113         when fsm_neg_gen =>
114           if signed(r.counter_neg_gen) >=
              ↪  signed(sys_i.pulse_neg_gen_duration) - 1 then
115             v.counter_neg_gen := (others => '0');
116             v.state := fsm_neg_dis;
117           else
118             v.counter_neg_gen := slv(signed(v.counter_neg_gen) + 1);
                ↪  --increment state counter
119           end if;
120         --Negative Discharge
121         when fsm_neg_dis =>
122           if signed(r.counter_neg_dis) >=
              ↪  signed(sys_i.pulse_neg_dis_duration) - 1 then
123             v.counter_neg_dis := (others => '0');
124             v.number_counter := slv(signed(v.number_counter) + 1);
                ↪  --increment counter in monopolar mode
125             if signed(v.number_counter) >= signed(sys_i.pulse_number)
                ↪  then
126               if signed(r.burst_counter) >= signed(sys_i.burst_number) -
                  ↪  1 then
127                 v.state := fsm_sleep;
128               else
129                 v.state := fsm_pause_burst; --Don't like this------
130               end if;
131             else
132               v.state := fsm_pause_bipol;
133             end if;
134           else
135             v.counter_neg_dis := slv(signed(v.counter_neg_dis) + 1);
                ↪  --increment state counter
136           end if;
137         --Pause Bipolar
138         when fsm_pause_bipol =>
139           if signed(r.counter_pause_bipol) >=
              ↪  signed(sys_i.pause_bipol_duration) - 1 then
140             v.counter_pause_bipol := (others => '0');
```

```
141               v.state := fsm_pos_prep;
142             else
143               v.counter_pause_bipol := slv(signed(v.counter_pause_bipol) +
                  ↪  1); --increment state counter
144             end if;
145           --Pause Burst
146           when fsm_pause_burst =>
147             if signed(r.counter_pause_burst) >=
                  ↪  signed(sys_i.pause_burst_duration) - 1 then
148               v.counter_pause_burst := (others => '0');
149               v.number_counter := (others => '0');
150               v.burst_counter := slv(signed(v.burst_counter) + 1);
                    ↪  --increment burst counter
151               v.state := fsm_pos_prep;
152             else
153               v.counter_pause_burst := slv(signed(v.counter_pause_burst) +
                    ↪  1); --increment state counter
154             end if;
155           --Should never come here.
156           when others =>
157             null;
158         end case;
159
          ↪  --=============================================================================================
160         --One-shot trigger
161         if r.trigger_flag = '0' AND sys_i.GEN_START = '1' then
162           v.trigger_start := '1';
163         else
164           v.trigger_start := '0';
165         end if;
166
          ↪  --=============================================================================================
167         --reset system
168         if reset = '1' then
169           v := init;
170         end if;
171
          ↪  --=============================================================================================
172         sys_o.state_out <= r.state;
173         rin <= v; --update next state
174
          ↪  --=============================================================================================
175       end process comb;
176
        ↪  -----------------------------------------------------------------------------------------------
177       regs: process(clk)
178       begin
179
          ↪  --=============================================================================================
180         if rising_edge(clk) then
```

```
181           r <= rin;
182           r.trigger_flag <= sys_i.GEN_START;
183         end if;
184
      ↪  --=============================================================================================
185       end process regs;
186
    ↪  ---------------------------------------------------------------------------------------------
187     end Behavioral;
```

8 State_Mux.vhd

```
1   library ieee ;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4   use work.generator.state_type;
5
6
7   entity State_MUX is
8     port(
9     --input
10      mon_simple_in: in state_type;
11      mon_full_in: in state_type;
12      bi_simple_in: in state_type;
13      bi_full_in: in state_type;
14      s: in std_logic_vector(1 downto 0);
15      --output
16      state_out: out state_type
17    );
18  end State_MUX;
19
20  architecture rtl of State_MUX is
21
22  type t_array_mux is array (0 to 3) of state_type;
23
24  signal array_mux  : t_array_mux;
25
26  begin
27    array_mux(0) <= mon_simple_in;
28    array_mux(1) <= mon_full_in;
29    array_mux(2) <= bi_simple_in;
30    array_mux(3) <= bi_full_in;
31
32    state_out <= array_mux(to_integer(unsigned(s)));
33  end rtl;
```

9 State_Decoder.vhd

```
1  library ieee ;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.generator.all;
5
6
7  entity State_Decoder is
8    -- generic(
9    --  out_sleep: std_logic_vector(3 downto 0):= "0000";
10   --  out_pos_prep: std_logic_vector(3 downto 0):= "0010";
11   --  out_pos_gen: std_logic_vector(3 downto 0):= "1010";
12   --  out_pos_dis: std_logic_vector(3 downto 0):= "0011";
13   --  out_neg_prep: std_logic_vector(3 downto 0):= "0001";
14   --  out_neg_gen: std_logic_vector(3 downto 0):= "0101";
15   --  out_neg_dis: std_logic_vector(3 downto 0):= "0011"
16   -- );
17   port(
18     clk, reset : in std_logic;
19     sys_i: in decoder_in_type;
20     sys_o: out decoder_out_type
21   );
22 end State_Decoder;
23
24 architecture Behavioral of State_Decoder is
25
26   type reg_type is record
27     state_load: state_type;
28     out_buff: decoder_out_type;
29     ICE_trigger: std_logic;
30   end record;
31
32   constant init: reg_type := (
33     state_load => fsm_sleep,
34     out_buff => (
35       GEN_END_OUT => '0',
36       GEN_POS_PULSE => '0',
37       GEN_NEG_PULSE => '0',
38       MOS_DRIVER_OUT => out_sleep
39     ),
40     ICE_trigger => '0'
41   );
42
43   signal r, rin : reg_type := init;
44 begin
45 ------------------------------------------------------------------------------------------------
46   comb: process(sys_i, r, reset)
47   variable v : reg_type;
```

```
48  begin
49
    ↪  --=============================================================================================
50    v := r;
51
    ↪  --=============================================================================================
52    if reset = '1' then
53      v := init;
54    elsif r.ICE_trigger = '1' then
55      v.state_load := fsm_ICE;
56    else
57      v.state_load := sys_i.state_load;
58    end if;
59
    ↪  --=============================================================================================
60    if sys_i.ICE = '1' then
61      v.ICE_trigger := '1';
62      v.state_load := fsm_ICE;
63    end if;
64
    ↪  --=============================================================================================
65    case r.state_load is --determine outputs
66      when fsm_sleep =>
67        v.out_buff.GEN_END_OUT := '1';
68        v.out_buff.GEN_POS_PULSE := '0';
69        v.out_buff.GEN_NEG_PULSE := '0';
70        v.out_buff.MOS_DRIVER_OUT := out_sleep;
71      when fsm_pos_prep =>
72        v.out_buff.GEN_END_OUT := '0';
73        v.out_buff.GEN_POS_PULSE := '0';
74        v.out_buff.GEN_NEG_PULSE := '0';
75        v.out_buff.MOS_DRIVER_OUT := out_pos_prep;
76      when fsm_pos_gen =>
77        v.out_buff.GEN_END_OUT := '0';
78        v.out_buff.GEN_POS_PULSE := '1';
79        v.out_buff.GEN_NEG_PULSE := '0';
80        v.out_buff.MOS_DRIVER_OUT := out_pos_gen;
81      when fsm_pos_dis =>
82        v.out_buff.GEN_END_OUT := '0';
83        v.out_buff.GEN_POS_PULSE := '0';
84        v.out_buff.GEN_NEG_PULSE := '0';
85        v.out_buff.MOS_DRIVER_OUT := out_pos_dis;
86      when fsm_neg_prep =>
87        v.out_buff.GEN_END_OUT := '0';
88        v.out_buff.GEN_POS_PULSE := '0';
89        v.out_buff.GEN_NEG_PULSE := '0';
90        v.out_buff.MOS_DRIVER_OUT := out_neg_prep;
91      when fsm_neg_gen =>
92        v.out_buff.GEN_END_OUT := '0';
93        v.out_buff.GEN_POS_PULSE := '0';
```

```
94          v.out_buff.GEN_NEG_PULSE := '1';
95          v.out_buff.MOS_DRIVER_OUT := out_neg_gen;
96        when fsm_neg_dis =>
97          v.out_buff.GEN_END_OUT := '0';
98          v.out_buff.GEN_POS_PULSE := '0';
99          v.out_buff.GEN_NEG_PULSE := '0';
100         v.out_buff.MOS_DRIVER_OUT := out_neg_dis;
101       when fsm_pause_bipol | fsm_pause_mono | fsm_pause_burst =>
102         v.out_buff.GEN_END_OUT := '0';
103         v.out_buff.GEN_POS_PULSE := '0';
104         v.out_buff.GEN_NEG_PULSE := '0';
105         v.out_buff.MOS_DRIVER_OUT := out_sleep;
106       when fsm_ICE =>
107         v.out_buff.GEN_END_OUT := '1';
108         v.out_buff.GEN_POS_PULSE := '0';
109         v.out_buff.GEN_NEG_PULSE := '0';
110         v.out_buff.MOS_DRIVER_OUT := out_ICE;
111       when others =>
112         null;
113     end case;
114
    ↪  --=============================================================================================
115     sys_o <= r.out_buff;
116     rin <= v;
117
    ↪  --=============================================================================================
118   end process;
119 ------------------------------------------------------------------------------------------------
120   reg: process(clk)
121   begin
122
    ↪  --=============================================================================================
123     if rising_edge(clk) then
124       r <= rin;
125     end if;
126
    ↪  --=============================================================================================
127   end process;
128 ------------------------------------------------------------------------------------------------
129
130 end Behavioral;
```