```
////////////////////////////////////////////////////////////////////////////////////////
// dte power discovery pld
// Rick Brooks
// 7/8/2000
// added code for detecting xfmr saturation during power on, to detect an open circuit
////////////////////////////////////////////////////////////////////////////////////////

module  dte_pld_rand5(clk, clk40n, clk160n, in1, in2, n_reset,
                drv1a, drv1b, drv1c, drv1d, drv1e ,drv1f, drv1g, drv1h,
                drv2a, drv2b, drv2c, drv2d, drv2e ,drv2f, drv2g, drv2h,
                detect_open, detect_short, pospulse_detect, negpulse_detect,
                n_discovered_LED, n_enable_disc_process, power_fault, power_open,
                dead_pospulse, dead_negpulse, n_spare_LED,
                n_turn_power_on, n_power_is_on_LED, pwr_fault_in, n_ps_fault_LED,
                pwr_undercurrent, n_rc_reset, n_pld_clear);


////////////////////////////////////////////////////////////////////////////////////////
// inputs.....
////////////////////////////////////////////////////////////////////////////////////////

// 160ns global clock input, and global reset
input clk, n_reset;

// 40ns clock input from oscillator
input clk40n;

// global output enable not used
// input output_ena_in;

// inputs from the detector, in1 monitors drv1 and in2 monitors drv2
input in1, in2;

// inputs from the power module, active high inputs
input pwr_fault_in, pwr_undercurrent;

// input to enable the discovery and power process, otherwise everything remains off
input n_enable_disc_process;

// RC reset input, need to clean up this signal
input n_rc_reset;
////////////////////////////////////////////////////////////////////////////////////////




////////////////////////////////////////////////////////////////////////////////////////
// outputs...
////////////////////////////////////////////////////////////////////////////////////////

// 160ns clock output to drive the global clock for the PLD
output clk160n;

// 8 outputs for the first pulse, staggered drive to achieve gaussian rise and fall times
output drv1a, drv1b, drv1c, drv1d, drv1e, drv1f, drv1g, drv1h;

// 8 outputs for the second pulse, staggered drive to achieve gaussian rise and fall times
output drv2a, drv2b, drv2c, drv2d, drv2e, drv2f, drv2g, drv2h;

// logic for detection of a open load or a shorted load
output detect_open, detect_short;

// these outputs drive the DTE power module, or the indicator
output n_turn_power_on, n_power_is_on_LED;

// to detect whether load current has been detected for a given single pulse
output pospulse_detect, negpulse_detect;

// to detect an input that is stuck high and therefore faulty
output dead_pospulse, dead_negpulse;

// registered version of power module fault, or power module open load
output power_fault, power_open, n_ps_fault_LED;

// says that the discovery process has successfully found a qualified device
```

```
output n_discovered_LED;

// output of cleaned up reset signal
output n_pld_clear;

// spare LED output
output n_spare_LED;
//////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////
// registers...
//////////////////////////////////////////////////////////////////////////////////////////////////////

// clock divider to generate global clock: clk160n
wire clk160n;
reg [1:0] clkgen;

// used to generate the shaped output #1
reg drv1a, drv1b, drv1c, drv1d, drv1e, drv1f, drv1g, drv1h;

// used to generate the shaped output #2
reg drv2a, drv2b, drv2c, drv2d, drv2e, drv2f, drv2g, drv2h;

// state machines
reg  [8:0] state_main;          // the main state machine
reg [2:0] state_pulse1;         // state machine for shaped output #1
reg [2:0] state_pulse2;         // state machine for shaped output #2

// handoff signals between state machines
reg ramppulsego1, ramppulsedone1, ramppulsego2, ramppulsedone2;
reg idletimerstart, faultdelaystart, timerdone;

// sampling of the inputs at various times
reg in_t1, in_t2, in_t3, in_t4, in_t5, in_t6, in_t7, in_t8;

// used for the detection of an open cable, or a shorted cable
reg detect_open, detect_short;

// used to find an input that is stuck high
reg dead_pospulse, dead_negpulse;

// used to capture the status and control of the power module
reg n_turn_power_on, power_fault, power_open;

// the number of consectutive positive diode detections
reg [8:0] polarity_p;       // counts the detection of a straight through cable
reg [8:0] polarity_n;       // counts the detection of a crossover cable

// counter used to generate the 8 staggered outputs: drv1a through drv1h
reg [5:0] pulsecount1;          // the counter used for positive ramp and pulse

// counter used to generate the 8 staggered outputs: drv2a through drv2h
reg [5:0] pulsecount2;          // the counter used for negative ramp and pulse

// the timer value for counting up to a few hundred msec
reg [23:0] timer;

// detects a single positive polarity pulse transmission
reg pospulse_detect;

// detects a single negative polarity pulse transmission
reg negpulse_detect;

// bit that signifies that discovery has been successful
reg n_discovered_LED;

// bit that signifies that the "power on" mode has been established
reg powermode;

// registered version of the input enable, without this, everything turns off, i.e. becomes "safe"
reg enable_disc;

// this register contains the latest pseudo random number used to determine the time between pulses
```

```
reg [11:0] rand_idletime;

// this is true when a new pseudo random number has been captured
reg rand_captured;

// clean up registers to generate the global reset signal
reg n_pld_clear, n_pld_clear1;

// generate a delay for the reset
reg [7:0] reset_count;

// for driving an external LED that states "the power is turned on"
wire n_power_is_on_LED;

// spare LED
wire n_spare_LED;
////////////////////////////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////////////////////////////////
// assignments and parameters
////////////////////////////////////////////////////////////////////////////////////////////////////////////

// output for the LED
assign n_power_is_on_LED = n_turn_power_on;

// output for the LED
assign  n_spare_LED = 1'b1;

// drive an LED to say PS fault
assign n_ps_fault_LED = ~power_fault;

// this is the 160ns output clock which is fed into the global clk input
assign clk160n = clkgen[1];

// main state machine encodings
parameter start = 9'b000000000;
parameter idlephase1 = 9'b000000001, waitphase1 =   9'b000000010, idlephase2 = 9'b000000100;
parameter waitphase2 = 9'b000001000, handoffphase = 9'b000010000, poweroninit = 9'b000100000;
parameter poweron = 9'b001000000, fault = 9'b010000000, checkload = 9'b100000000;

// encodings for the pulse #1 state machine
parameter no_pulse1 = 3'b000, rampup1 = 3'b001, pulse1 = 3'b011, rampdown1 = 3'b010, wait1 = 3'b110;

// encodings for the pulse #2 state machine
parameter no_pulse2 = 3'b000, rampup2 = 3'b001, pulse2 = 3'b011, rampdown2 = 3'b010, wait2 = 3'b110;

// sets the delay before which power faults are ignored
// with a 160ns global clock, a value of 24'h00100 yields about 40.96us of delay
// with a 160ns global clock, a value of 24'h800000 yields about 1.342 seconds of delay
parameter faultdelaytime = 24'h200000;

// sets the number of consecutive detections before entering the power on state
// 9'h100 gives 256 consecutive detections
parameter consec_detect = 9'h100;

// sets the number of consecutive opens during power on state before turning off power
parameter consec_opens = 8'h40;

// sets the ramp up and down delay times
// a spacing of 6'h02 produces 320 ns between steps
// a spacing of 6'h03 produces 480 ns between steps
parameter td0 = 6'h00;
parameter td1 = 6'h03;
parameter td2 = 6'h06;
parameter td3 = 6'h09;
parameter td4 = 6'h0C;
parameter td5 = 6'h0F;
parameter td6 = 6'h12;
parameter td7 = 6'h15;
```

```
// sets the pulse duration and sample points for detection during the pulse interval
// a pulselength of 8'h20 produces a pulse flat top of approx 5.12 us
parameter pulselength = 8'h20;    // use 8'h20
parameter p_sample1 = 8'h1C;          // must be slightly less than p_sample2 (use 8'h1C)
parameter p_sample2 = 8'h1E;          // must be slightly less than the pulselength time (use 8'h1E)

///////////////////////////////////////////////////////////////////////////////////////////////////////




///////////////////////////////////////////////////////////////////////////////////////////////////////
// pseudo random number generator, adapted from the TP-PMD22 spec
///////////////////////////////////////////////////////////////////////////////////////////////////////
//
// output tnrzdout;           // the random number output
wire [11:0]s;            // descrambler key stream
reg [10:0]ds;                 // key stream register
reg tnrzdout;                 // ciphertext output bit
wire tnrzdin;
//
// KEY STREAM
//
assign tnrzdin = 1'b1;
assign s[11:1] = ds[10:0];                // shift previous bits
assign s[0] = s[11] ^ s[9];      // generate newest bit
//
always @(posedge clk)
if (~n_pld_clear1)               // reset the stream
     ds[10:0] = #1 11'h0;                  // reset key stream
else if ( ds == 11'h0 )
     ds[10:0] = #1 11'h1;                  // initialize key stream
else
     ds[10:0] = #1 s[10:0];                      // save current key stream
//
// CIPHERTEXT STREAM
//
always @(posedge clk)
tnrzdout = #1 s[0] ^ tnrzdin; // scramble NRZ data bit
//
//
///////////////////////////////////////////////////////////////////////////////////////////////////////




///////////////////////////////////////////////////////////////////////////////////////////////////////
// clock generation
// here, a 40ns clock comes in, and a 160ns clock is generated
// the 160ns clock is hooked externally to the global clock input (clk)
// this is done to improve timing margin in a PLD implementation
///////////////////////////////////////////////////////////////////////////////////////////////////////

always @(posedge clk40n or negedge n_reset)
     begin
          if (n_reset == 0)
               begin
                    clkgen = 2'b00;
               end
          else
               begin
                    clkgen = clkgen + 1;
               end
     end
///////////////////////////////////////////////////////////////////////////////////////////////////////




///////////////////////////////////////////////////////////////////////////////////////////////////////
// reset generation
// the input is from an RC circuit: R pullup, C to ground....
// here we must clean up the RC input and create a global reset signal
// clk40n is used here, since it comes directly from the oscillator
// n_pld_clear is hooked up externally to the global n_reset pin
```

```
////////////////////////////////////////////////////////////////////////////////////////////////

// first register the n_rc_reset input
always @(posedge clk40n)
    begin
        if (n_rc_reset == 1)
            begin
                n_pld_clear1 = 1'b1;
            end
        else
            begin
                n_pld_clear1 = 1'b0;
            end
    end

always @(posedge clk40n)
    begin
        if (n_pld_clear1 == 0)
            begin
                reset_count[7:0] = 8'h00;
                n_pld_clear = 1'b0;
            end
        else if (reset_count[7] == 1)
            begin
                n_pld_clear = 1'b1;
            end
        else
            begin
                reset_count = reset_count + 1;
            end
    end

////////////////////////////////////////////////////////////////////////////////////////////////









////////////////////////////////////////////////////////////////////////////////////////////////
// general purpose timer
// 24'h800000 is equiv to 1.342 seconds using the 160ns clock (clk)
////////////////////////////////////////////////////////////////////////////////////////////////

always @(posedge clk or negedge n_reset)
    begin
        if (n_reset == 0)       // initialize registers
            begin
                timer = 24'h000000;
                timerdone = 0;
                rand_captured = 0;
                rand_idletime[10:0] = 11'h000;
            end

        else if (enable_disc == 0)          // initialize registers
            begin
                timer = 24'h000000;
                timerdone = 0;
                rand_captured = 0;
                rand_idletime[10:0] = 11'h000;
            end

        else
            begin
                if (idletimerstart == 1 & enable_disc == 1) // this case produces a pseudo random time
                    begin
                        if (idletimerstart == 1 & rand_captured == 0)          // here the random nuber is captured
                            begin
// the random idle time between pulses
// with a set minimum of 40.96us
//                                      rand_idletime[10:0] =      { 1'b0 , ds[0] , 1'b1 , ds[2] , ds[5] , ds[10] ,
```

```
//                                                              ds[1] , ds[4] , ds[7] , ds[3] , ds[6]};

// the random idle time between pulses
// with a set minimum of ??us
// used for testing
//                              rand_idletime[10:0] =   { 1'b0 , 1'b0 , 1'b0 , 1'b1 , ds[5] , ds[10] ,
//                                                      ds[1] , ds[4] , ds[7] , ds[3] , ds[6]};


// this is the right one, I think
//
//                              rand_idletime[10:0] =   { ds[0] , ds[1] , 1'b1 , ds[2] , ds[3] ,
//                                                      ds[4] , ds[5] , ds[6] , ds[7] , ds[8] , ds[9]};

// this is the right one, I think
//
                                rand_idletime[11:0] =   { ds[0] , ds[1] , ds[2] , 1'b1 , ds[3] , ds[4] ,
                                                        ds[5] , ds[6] , ds[7] , ds[8] , ds[9] , ds[10]};


                                    rand_captured = 1;
                                end

                            else if (timer == rand_idletime & rand_captured == 1)       //the timer is done
                                begin
                                    timerdone = 1;
                                    rand_captured = 0;
                                end

                            else
                                timer = timer + 1;
                        end

// here the timer is used for a delay
// so the PS faults are ignored until after the delay

                    else if (faultdelaystart == 1 & enable_disc == 1)
                        begin
                            if (timer == faultdelaytime)
                                timerdone = 1;

                            else
                                timer = timer + 1;
                        end

                    else
                        begin
                            timer = 24'h000000;
                            timerdone = 0;
                        end
                end
        end
////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////
// the main discovery and power control state machine
////////////////////////////////////////////////////////////////////////////////////////////

always @(posedge clk or negedge n_reset)
        begin
            if (n_reset == 0)
                begin
                    ramppulsego1 = 0;
                    ramppulsego2 = 0;
                    idletimerstart = 0;
                    faultdelaystart = 0;
                    polarity_p = 9'h00;
                    polarity_n = 9'h00;
                    detect_open = 0;
                    detect_short = 0;
                    n_turn_power_on = 1;
                    power_fault = 0;
                    power_open = 0;
                    dead_pospulse = 0;
                    dead_negpulse = 0;
```

```
            enable_disc = 0;
            n_discovered_LED = 1;
            state_main = 7'b0000000;
            pospulse_detect = 0;
            negpulse_detect = 0;
            powermode = 0;
        end
    else
        begin

            pospulse_detect = ~in_t1 & in_t2 & in_t3 & ~in_t4;      // detected current flow pos direction
            dead_pospulse = in_t1 & in_t2 & in_t3 & in_t4;          // the signal for pulse 1 is stuck high
            negpulse_detect = ~in_t5 & in_t6 & in_t7 & ~in_t8;      // detected current flow neg direction
            dead_negpulse = in_t5 & in_t6 & in_t7 & in_t8;          // the signal for pulse 2 is stuck high
            power_fault = pwr_fault_in;             // registered version of a power fault
            power_open = pwr_undercurrent; // registered version of a power open (undercurrent)
            enable_disc = ~n_enable_disc_process;   // registered version of the enable input

            case (state_main)

                start:                  // the initial safe starting state
                    begin
                        ramppulsego1 = 0;
                        ramppulsego2 = 0;
                        idletimerstart = 0;
                        faultdelaystart = 0;
                        polarity_p = 9'h00;
                        polarity_n = 9'h00;
                        detect_open = 0;
                        detect_short = 0;
                        n_turn_power_on = 1;
                        n_discovered_LED = 1;
                        powermode = 0;

                        if (enable_disc == 0)
                            state_main = start;             // wait until the enable is true
                        else
                            state_main = idlephase1;

                    end

                idlephase1:             // waits for the idle timer to count
                    begin
                        ramppulsego1 = 0;
                        idletimerstart = 1;         // handoff to the idle timer state machine

                        if (enable_disc == 0) // process is not enabled anymore, go to the start state
                            begin
                                state_main = start;
                            end

                        else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
                            begin
                                state_main = fault;
                            end

                        else if (timerdone == 1)        // the timer is done, go to the next step
                            begin
                                idletimerstart = 0;
                                state_main = waitphase1;
                            end

                        else
                            state_main = idlephase1;
                    end

                waitphase1:
                    begin
                        ramppulsego1 = 1;   // handoff to the ramp1/pulse1 state machine

                        if (enable_disc == 0) // process is not enabled anymore, go to the start state
                            begin
                                state_main = start;
                            end

                        else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
```

```
                    begin
                        state_main = fault;
                    end

                else if (ramppulsedone1 == 1)           // the ramp1/pulse1 state machine is done
                    begin
                        state_main = idlephase2;
                        ramppulsego1 = 0;
                    end

                else
                    begin
                        state_main = waitphase1;
                    end
            end

        idlephase2:             // waits for the idle timer to count
            begin
                ramppulsego2 = 0;
                idletimerstart = 1;         // handoff to the idle timer state machine

                if (enable_disc == 0) // process is not enabled anymore, go to the start state
                    begin
                        state_main = start;
                    end

                else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
                    begin
                        state_main = fault;
                    end

                else if (timerdone == 1 & powermode == 0)           // the timer is done, go to the next step
                    begin
                        idletimerstart = 0;
                        state_main = waitphase2;
                    end

                else if (timerdone == 1)        // the timer is done, go to the next step
                    begin
                        idletimerstart = 0;
                        state_main = checkload;
                    end

                else
                    state_main = idlephase2;
            end

        waitphase2:
            begin
                ramppulsego2 = 1;   // handoff to the ramp2/pulse2 state machine

                if (enable_disc == 0) // process is not enabled anymore, go to the start state
                    begin
                        state_main = start;
                    end

// here is where the determination is made as to whether we have detected the AC coupled diode load

                else if (ramppulsedone2 == 1)           // the ramp1/pulse2 state machine is done
                    begin

                        ramppulsego2 = 0;
                        if (pospulse_detect == 1 & negpulse_detect == 0
                            & ~dead_negpulse)       // detects a proper pos. direction current in the load
                            begin
                                polarity_p = polarity_p + 1;    // increments the consecutive pos detect counter
                                polarity_n = 9'h00;
                                detect_open = 0;
                                detect_short = 0;
                            end

                        else if (negpulse_detect == 1 & pospulse_detect == 0
                                & ~dead_pospulse)       // detects a proper neg. direction current in the load
                            begin
                                polarity_n = polarity_n + 1;    // increments the consecutive neg detect counter
                                polarity_p = 9'h00;
```

```
                    detect_open = 0;
                    detect_short = 0;
                end

            else if (negpulse_detect == 0 & pospulse_detect == 0
                    & ~dead_pospulse & ~dead_negpulse)
                begin
                    detect_open = 1;        // detects an open cable, zero the counters
                    detect_short = 0;
                    polarity_n = 9'h00;
                    polarity_p = 9'h00;
                end

            else if (negpulse_detect == 1 & pospulse_detect == 1)
                begin
                    detect_short = 1;       // detected a shorted cable, zero the counters
                    detect_open = 0;
                    polarity_n = 9'h00;
                    polarity_p = 9'h00;
                end

            else                        // a default case, zero everything
                begin
                    detect_short = 0;
                    detect_open = 0;
                    polarity_n = 9'h00;
                    polarity_p = 9'h00;
                end

            state_main = handoffphase;
        end

    else
        begin
            state_main = waitphase2;
        end

end


checkload:          // get here only when the power is on, check for open load
    begin

    ramppulsego2 = 1;       // handoff to the ramp2/pulse2 state machine

    if (enable_disc == 0) // no longer enabled; go to the start state
        begin
            state_main = start;
        end

    else if (power_fault | power_open)      // fault; go to the start state
        begin
            state_main = fault;
        end

    else if (ramppulsedone2 == 1)           // the ramp1/pulse2 state machine is done
        begin
            ramppulsego2 = 0;

            if ((~in_t2 & ~in_t3 & ~in_t6 & ~in_t7) | ~power_open)      // load current is flowing
                begin
                    state_main = poweron;           // keep the power on
                end

            else                        // detecting an open load
                begin
                    if (polarity_p == consec_opens)
                        begin
                            state_main = fault;         // open load; turn the power off
                        end
                    else
                        begin
                            polarity_p = polarity_p + 1;
                            state_main = poweron;       // keep the power on for now
                        end
                end
```

```
                end

            else
                begin
                    state_main = checkload;
                end

        end


handoffphase:               // checks for conditions to go to power on state
        begin
            if (enable_disc == 0) // process is not enabled anymore, go to the start state
                begin
                    state_main = start;
                end

            else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
                begin
                    state_main = fault;
                end

            else if (polarity_p == consec_detect)         // checks the number of consecutive counts
                    begin
                        n_discovered_LED = 0;
                        state_main = poweroninit;      // enough positive counts to enable power!!!!
                    end

            else if (polarity_n == consec_detect)         // checks the number of consecutive counts
                    begin
                        n_discovered_LED = 0;
                        state_main = poweroninit;      // enough negative counts to enable power!!!!
                    end

            elsestate_main = idlephase1;// not enough counts, so repeat the idle/pulse process...
        end

poweroninit:
        begin
            faultdelaystart = 1;
            n_turn_power_on = 0;
            polarity_p = 9'h00;
            polarity_n = 9'h00;

            if (enable_disc == 0) // process is not enabled anymore, go to the start state
                begin
                    state_main = start;
                end

            else if (timerdone == 1)    // timer is done, now start looking at the PS faults
                begin
                    faultdelaystart = 0;          // turn off the timer
                    state_main = poweron;
                end

            else
                begin
                    state_main = poweroninit;
                end
        end

poweron:
        begin
            if (enable_disc == 0) // process is not enabled anymore, go to the start state
                begin
                    state_main = start;
                end

            else if (power_fault == 1 | power_open == 1)    // look for PS faults
                begin
                    n_discovered_LED = 1;
                    n_turn_power_on = 1;
                    powermode = 0;
                    state_main = fault;
                end
```

```
                        else
                            begin
                                state_main = idlephase1;
                                powermode = 1;
                            end
                    end

                fault:
                    begin
                        n_turn_power_on = 1;                    // turn off the power module
                        n_discovered_LED = 1;          // "discovered" is false now
                        powermode = 0;

                        if (enable_disc == 0) // process is not enabled anymore, go to the start state
                            begin
                                state_main = start;
                            end

                        else        // start the timer to keep power off for at least at fault delay time
                            begin
                                faultdelaystart = 1;

                            if (timerdone == 1)     // timer is done, now return to start
                                begin
                                    faultdelaystart = 0;            // turn off the timer
                                    state_main = start;
                                end

                            else            // stay in fault state until timer is done
                                begin
                                    state_main = fault;
                                end
                            end

                    end

                default: state_main = start;

            endcase
        end
    end
////////////////////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////////////////////////
// the pulse #1 state machine
// creates the staggered outputs to produce shaped output #1
////////////////////////////////////////////////////////////////////////////////////////////////////

always @(posedge clk or negedge n_reset)
  begin
    if (n_reset == 0)
      begin
        drv1a = 0; drv1b = 0; drv1c = 0; drv1d = 0;
        drv1e = 0; drv1f = 0; drv1g = 0; drv1h = 0;
        ramppulsedone1 = 0;
        pulsecount1 = 8'h00;
        in_t1 = 0; in_t2 = 0; in_t3 = 0; in_t4 = 0;
        state_pulse1 = 3'b000;
      end
    else
      begin
        case (state_pulse1)
          no_pulse1:
            if (ramppulsego1 == 1 & ~ramppulsedone1 == 1)
              begin
                state_pulse1 = rampup1;
              end
            else
              begin
                drv1a = 0; drv1b = 0; drv1c = 0; drv1d = 0;
                drv1e = 0; drv1f = 0; drv1g = 0; drv1h = 0;
                ramppulsedone1 = 0;
                pulsecount1 = 8'h00;
```

```
            end

rampup1:
        begin
            if (enable_disc == 0) // process is not enabled anymore, go to no_pulse1 state
                begin
                    state_pulse1 = no_pulse1;
                end

            else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
                begin
                    state_pulse1 = no_pulse1;
                end

            else

            case (pulsecount1)


                td7:
                    begin
                        state_pulse1 = pulse1;
                        pulsecount1 = 8'hFF;
                        drv1h = 1;
                    end

                td6: drv1g = 1;

                td5: drv1f = 1;

                td4: drv1e = 1;

                td3: drv1d = 1;

                td2: drv1c = 1;

                td1: drv1b = 1;

                td0:
                    begin
                        drv1a = 1;
                            if (powermode == 0)
                                begin
                                    in_t1 = in2;        // sample the 1st value in the detection template
                                end
                            else
                                begin
                                    in_t1 = in1;        // sample the 1st value in the detection template
                                end
                    end

                default: ;
            endcase

            pulsecount1 = pulsecount1 + 1;

        end

pulse1:
        begin
            if (enable_disc == 0) // process is not enabled anymore, go to no_pulse1 state
                begin
                    state_pulse1 = no_pulse1;
                end

            else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
                begin
                    state_pulse1 = no_pulse1;
                end

            else

            case (pulsecount1)
                pulselength:        // the end of the flat top
                    begin
                        state_pulse1 = rampdown1;
```

```
                    pulsecount1 = 8'hFF;
                end

            p_sample2:
                begin
                    if (powermode == 0)
                        begin
                            in_t3 = in2;        // sample the 3rd value in the detection template
                        end
                    else
                        begin
                            in_t3 = in1;        // sample the 3rd value in the detection template
                        end
                end

            p_sample1:
                begin
                    if (powermode == 0)
                        begin
                            in_t2 = in2;        // sample the 2nd value in the detection template
                        end
                    else
                        begin
                            in_t2 = in1;        // sample the 2nd value in the detection template
                        end
                end

            default: ;
        endcase

        pulsecount1 = pulsecount1 + 1;

    end

rampdown1:
    begin
        if (enable_disc == 0) // process is not enabled anymore, go to no_pulse1 state
            begin
                state_pulse1 = no_pulse1;
            end

        else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
            begin
                state_pulse1 = no_pulse1;
            end

        else

        case (pulsecount1)

            td7:
                begin
                    state_pulse1 = wait1;
                    pulsecount1 = 8'hFF;
                    ramppulsedone1 = 1;
                    drv1h = 0;
                        if (powermode == 0)
                            begin
                                in_t4 = in2;        // sample the 4th value in the detection template
                            end
                        else
                            begin
                                in_t4 = in1;        // sample the 4th value in the detection template
                            end
                end

            td6: drv1g = 0;

            td5: drv1f = 0;

            td4: drv1e = 0;

            td3: drv1d = 0;

            td2: drv1c = 0;
```

```
                        td1: drv1b = 0;

                        td0: drv1a = 0;

                        default: ;
                    endcase

                    pulsecount1 = pulsecount1 + 1;
                end

            wait1:
                begin
                    pulsecount1 = 8'h00;

                    if (enable_disc == 0 | ramppulsego1 == 0)
                        begin
                            state_pulse1 = no_pulse1;
                        end

                    else if (powermode == 1 & (power_fault | power_open))      // power fault, not enabled anymore
                        begin
                            state_pulse1 = no_pulse1;
                        end

                    else
                        begin
                            state_pulse1 = wait1;
                        end
                end

            default: state_pulse1 = no_pulse1;
        endcase
    end
end
//////////////////////////////////////////////////////////////////////////////////////////////////////////




//////////////////////////////////////////////////////////////////////////////////////////////////////////
// the pulse #2 state machine
// creates the staggered outputs to produce shaped output #2
//////////////////////////////////////////////////////////////////////////////////////////////////////////

always @(posedge clk or negedge n_reset)
    begin
        if (n_reset == 0)
            begin
                drv2a = 0; drv2b = 0; drv2c = 0; drv2d = 0;
                drv2e = 0; drv2f = 0; drv2g = 0; drv2h = 0;
                ramppulsedone2 = 0;
                pulsecount2 = 8'h00;
                in_t5 = 0; in_t6 = 0; in_t7 = 0; in_t8 = 0;
                state_pulse2 = 3'b000;
            end
        else
            begin
                case (state_pulse2)
                    no_pulse2:
                        if (ramppulsego2 == 1 & ~ramppulsedone2 == 1)
                            begin
                                state_pulse2 = rampup2;
                                pulsecount2 = 8'h00;
                            end
                        else
                            begin
                                drv2a = 0; drv2b = 0; drv2c = 0; drv2d = 0;
                                drv2e = 0; drv2f = 0; drv2g = 0; drv2h = 0;
                                ramppulsedone2 = 0;
                            end

                    rampup2:
                        begin
                            if (enable_disc == 0) // process is not enabled anymore, go to no_pulse2 state
                                begin
```

```
                state_pulse2 = no_pulse2;
            end

        else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
            begin
                state_pulse2 = no_pulse2;
            end

        else

        case (pulsecount2)

            td7:
                begin
                    state_pulse2 = pulse2;
                    pulsecount2 = 8'hFF;
                    drv2h = 1;
                end

            td6: drv2g = 1;

            td5: drv2f = 1;

            td4: drv2e = 1;

            td3: drv2d = 1;

            td2: drv2c = 1;

            td1: drv2b = 1;

            td0:
                begin
                    drv2a = 1;
                        if (powermode == 0)
                            begin
                                in_t5 = in1;        // sample the 5th value in the detection template
                            end
                        else
                            begin
                                in_t5 = in2;        // sample the 5th value in the detection template
                            end
                end

            default: ;
        endcase

        pulsecount2 = pulsecount2 + 1;
    end

pulse2:
    begin
        if (enable_disc == 0) // process is not enabled anymore, go to no_pulse2 state
            begin
                state_pulse2 = no_pulse2;
            end

        else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
            begin
                state_pulse2 = no_pulse2;
            end

        else

        case (pulsecount2)
            pulselength:        // the end of the flat top
                begin
                    state_pulse2 = rampdown2;
                    pulsecount2 = 8'hFF;
                end

            p_sample2:
                begin
                    if (powermode == 0)
                        begin
                            in_t7 = in1;        // sample the 7th value in the detection template
```

```
                    end
                else
                    begin
                        in_t7 = in2;        // sample the 7th value in the detection template
                    end
            end

        p_sample1:
            begin
                if (powermode == 0)
                    begin
                        in_t6 = in1;        // sample the 6th value in the detection template
                    end
                else
                    begin
                        in_t6 = in2;        // sample the 6th value in the detection template
                    end
            end

        default: ;

    endcase

    pulsecount2 = pulsecount2 + 1;

end

rampdown2:
    begin
        if (enable_disc == 0) // process is not enabled anymore, go to no_pulse2 state
            begin
                state_pulse2 = no_pulse2;
            end

        else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
            begin
                state_pulse2 = no_pulse2;
            end

        else

        case (pulsecount2)

            td7:
                begin
                    state_pulse2 = wait2;
                    pulsecount2 = 8'hFF;
                    ramppulsedone2 = 1;
                    drv2h = 0;
                        if (powermode == 0)
                            begin
                                in_t8 = in1;        // sample the 8th value in the detection template
                            end
                        else
                            begin
                                in_t8 = in2;        // sample the 8th value in the detection template
                            end
                end

            td6: drv2g = 0;

            td5: drv2f = 0;

            td4: drv2e = 0;

            td3: drv2d = 0;

            td2: drv2c = 0;

            td1: drv2b = 0;

            td0: drv2a = 0;

            default: ;
        endcase
```

```
                    pulsecount2 = pulsecount2 + 1;

                end

            wait2:
                begin
                    pulsecount2 = 8'h00;

                    if (enable_disc == 0 | ramppulsego2 == 0)
                        begin
                            state_pulse2 = no_pulse2;
                        end

                    else if (powermode == 1 & (power_fault | power_open))       // power fault, not enabled anymore
                        begin
                            state_pulse2 = no_pulse2;
                        end

                    else
                        begin
                            state_pulse2 = wait2;
                        end
                end

            default: state_pulse2 = no_pulse2;
        endcase
    end
end

////////////////////////////////////////////////////////////////////////////////////////////////////

endmodule
```