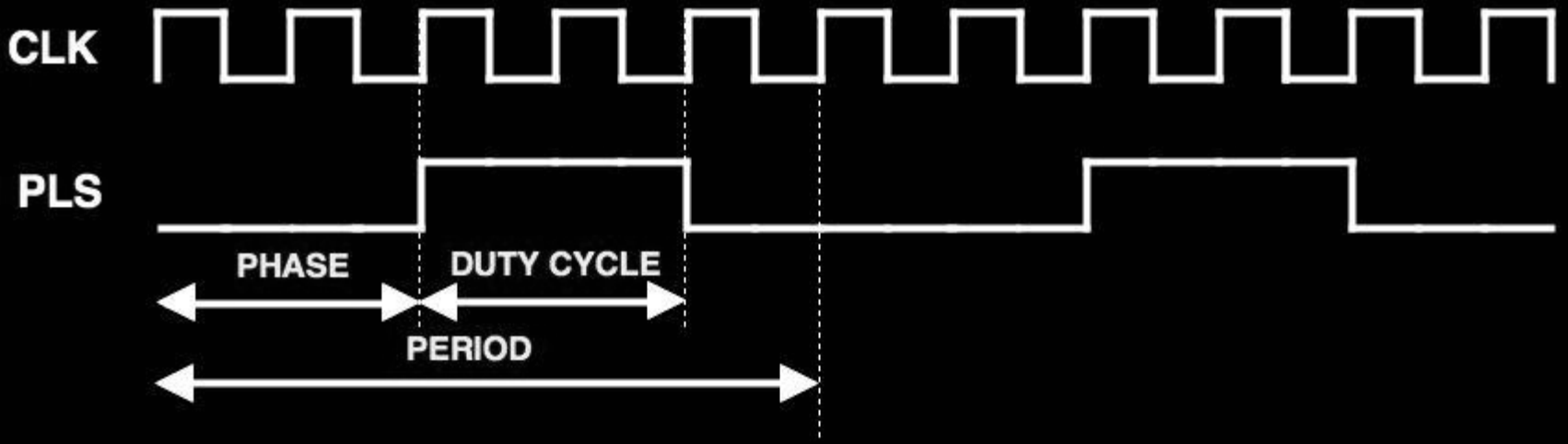
Pulse generator module

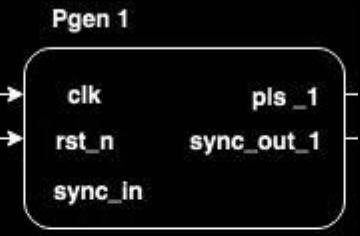


D. Pankova

VERILOG CODE

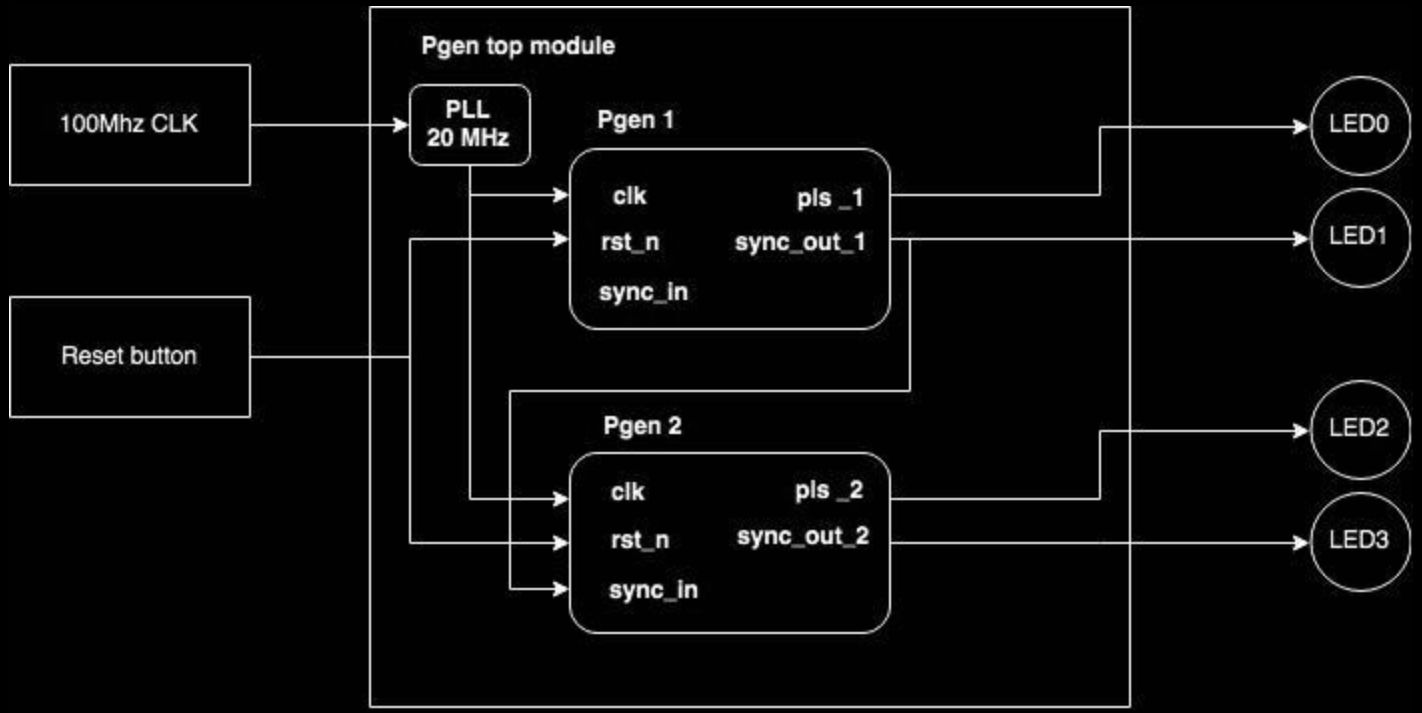
```
////////////////////////////////////////////////////////////////////
// Daria Pankova Mon Jun  8 14:07:54 EDT 2015
// pgen.v
//
// "Pulse generator"
// A custom Verilog HDL module.
// Creates a custom waveform from a clock signal.
// (with user set phase, period and length)
// WARNING: if the parameters are wrong they are going
// to be set to default
////////////////////////////////////////////////////////////////////

module pgen
  (
   input  clk,      // clock
   input  rst_n,    // reset
   input  sync_in,  // incoming syncronization
   output pls,      // the resulting wave
   output sync_out  // the outgoing syncronization
   );

   // Parameters
   parameter P_CLK_FREQ_HZ = 20000000; // 20MHz system clock
   parameter P_FREQ_HZ = 2000000; // 2MHz default output waveform
   parameter P_DUTY_CYCLE_PERCENT = 50; // 50% default duty cycle
   parameter P_PHASE_DEG = 0; // 0 degree phase by default

   //Function for setting period count value
   function integer set_period;
      input integer P_CLK_FREQ_HZ;
      input integer P_FREQ_HZ;
      //if clock frequency is higher then
      //module outputs calculated frequncy value
      if (P_CLK_FREQ_HZ > P_FREQ_HZ) set_period = P_CLK_FREQ_HZ/P_FREQ_HZ;
      //if not sets the default
      else set_period = 10;
   endfunction // if

   //Setting perion count value
   localparam PERIOD_CNT = set_period(P_CLK_FREQ_HZ,P_FREQ_HZ);
```

```
//Function for setting phase count value
function integer set_phase;
   input integer PERIOD_CNT;
   input integer P_PHASE_DEG;
   input integer P_DUTY_CYCLE_PERCENT;
   integer       duty_cycle;
   integer       phase_int;
   integer       phase;
   integer       condition;

   begin //calculate the condition
      // (period count smaller than phace and duty cycle counts)
      duty_cycle = P_DUTY_CYCLE_PERCENT*PERIOD_CNT/100;
      phase_int  = P_PHASE_DEG/360;
      phase = PERIOD_CNT*(P_PHASE_DEG - phase_int*360)/360;
      condition = (PERIOD_CNT < (phase + duty_cycle));
      //if condition is true set the default
      if (condition) set_phase = 1;
      //if not set the calculated value
      else set_phase = phase;
   end
endfunction // if
//Setting phase count value
localparam PHASE_CNT = set_phase(PERIOD_CNT,P_PHASE_DEG,P_DUTY_CYCLE_PERCENT);

//Function setting duty cycle count value
function integer set_duty_cycle;
   input integer PERIOD_CNT;
   input integer P_PHASE_DEG;
   input integer P_DUTY_CYCLE_PERCENT;
   integer       duty_cycle;
   integer       phase_int;
   integer       phase;
   integer       condition;

   begin //calculate the condition
      // (period count smaller than phace and duty cycle counts)
      duty_cycle = P_DUTY_CYCLE_PERCENT*PERIOD_CNT/100;
      phase_int  = P_PHASE_DEG/360;
      phase = PERIOD_CNT*(P_PHASE_DEG - phase_int*360)/360;
      condition = (PERIOD_CNT < (phase + duty_cycle));
      //if condition is true set the default
      if (condition) set_duty_cycle = 5;
      //if not set the calculated value
      else set_duty_cycle = duty_cycle;
   end
endfunction // if
//Setting duty cycle count value
localparam DUTY_CYCLE_CNT = set_duty_cycle(PERIOD_CNT,P_PHASE_DEG,P_DUTY_CYCLE_PERCENT);
```



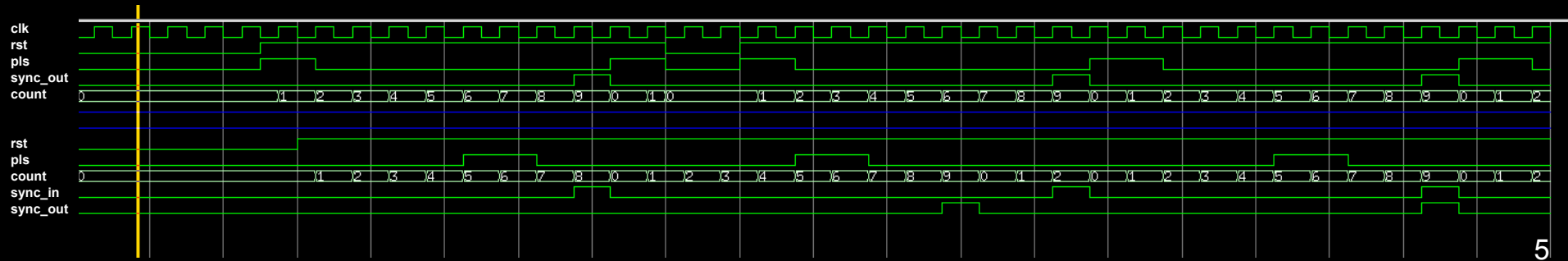
```
  // Internal counter
  // ceiling(log2()), used to figure out counter size.
  function integer clogb2;
     input integer    value;
     for(clogb2=0;value>0;clogb2=clogb2+1)
       value = value >> 1;
  endfunction // for

  // Counter
  localparam NBITS = clogb2(PERIOD_CNT);
  reg [NBITS-1:0]     count;
  always @(posedge clk or negedge rst_n)
    begin
       if (!rst_n) count <= {NBITS{1'b0}};
       else
         begin
            if      (sync_in == 1'b1)       count <= {NBITS{1'b0}};
            else if (count == PERIOD_CNT-1) count <= {NBITS{1'b0}};
            else                            count <= count + 1'b1;
         end
    end

  // Combinational
  assign sync_out = (count == PERIOD_CNT-1);
  assign pls = (count >= PHASE_CNT) && (count < PHASE_CNT+DUTY_CYCLE_CNT) && (rst_n);

ndmodule
```

Testing


Pgen top module
100Mhz CLK
Reset button
PLL
20 MHz
Pgen 1
clk
rst_n
sync_in
pls _1
sync_out_1
Pgen 2
clk
rst_n
sync_in
pls _2
sync_out_2
LED0
LED1
LED2
LED3

TEST BENCH

```
`ifdef PHASE_90DEG_2
   reg rst_n_1;
   reg rst_n_2;
   wire pls_1;
   wire pls_2;
   wire sync;
   reg  clk_1;

   pgen #(        .P_CLK_FREQ_HZ(20000000),
                  .P_FREQ_HZ(2000000),
                  .P_DUTY_CYCLE_PERCENT(20),
                  .P_PHASE_DEG(360) )
   i1 (   // port map - connection between master
       .pls(pls_1),
       .sync_out(sync),
       .rst_n(rst_n_1),
       .sync_in(sync_in),
       .clk(clk_1)
       );

   pgen #(        .P_CLK_FREQ_HZ(20000000),
                  .P_FREQ_HZ(2000000),
                  .P_DUTY_CYCLE_PERCENT(20),
                  .P_PHASE_DEG(180) )
   i2 (   // port map - connection between master por
       .pls(pls_2),
       .sync_out(sync_out),
       .rst_n(rst_n_2),
       .sync_in(sync),
       .clk(clk_1)
       );

   // Clock generator
   localparam PERIOD = 50.0;
   always #(PERIOD/2) clk_1 = ~ clk_1;

   initial
     begin
        clk_1 = 1'b0;
        rst_n_1 = 1'b0;
        rst_n_2 = 1'b0;
        #(5*PERIOD) rst_n_1 = 1'b1;
        #(1*PERIOD) rst_n_2 = 1'b1;
        #(10*PERIOD) rst_n_1 = 1'b0;
        #(2*PERIOD) rst_n_1 = 1'b1;
     end
`endif
```



FPGA Test

PLS_1

Sync_out_1

PLS_2

Sync_out_2