```
/// Verilog test fixture created to test sync signals
//  Instructions:
// 1. Add to your project as a simulation source.
// 2. Adjust instantiation of the UUT (line 44) which is your top module.
// 3. Comment out lines 112 and 137 if your outputs are not run through FFs.
// 4. Run Simulator.
// 5. Add SERRORS and RGBERRORS to the waveform viewer (these signals are under GUT
and RUT).
// 6. Reset simulation time to 0.
// 7. Set step size to 17ms.
// 8. Simulate one step (may take ?? seconds/minutes, or more if you have more
components).
// 9. If SERRORs or RGBERRORS is not 0 find where they changed: there's a problem
there or earlier.
//    Check for transitions on "oops" and rgb_oops" to find out where.
//10. Otherwise simulate one more step and check them again.
//
//   Martine  May 2023 (version with btnR as GSR)
//   greset is now the global reset signal that should be connected to whatever
//   input of the top level design source will be used as the greset

`timescale 1ns / 1ps

module testSyncs();

// Inputs
   reg btnC,btnD,btnU,btnL,btnR;
   reg [15:0] sw;
   reg clkin;
   reg greset;  //use as input to whatever will be connected to the global reset pin

// Output
   wire [3:0] an;
   wire dp;
   wire [6:0] seg;
   wire [15:0] led;
   wire HS;
   wire [3:0] vgaBlue;
   wire [3:0] vgaGreen;
   wire [3:0] vgaRed;
   wire VS;
   wire oops;
   wire rgb_oops;


// You may need to replace the instantiation below
```

```
// to match your top level module and its port names.

// Instantiate the UUT
   lab6_top UUT (
              .btnU(btnU),
              .btnD(greset), //btnD is the global reset in S24 Lab6
              .btnC(btnC),
              .btnR(btnR),
              .btnL(btnL),
              .clkin(clkin),
              .seg(seg),
              .an(an),
              .vgaBlue(vgaBlue),
              .vgaRed(vgaRed),
              .vgaGreen(vgaGreen),
              .Vsync(VS),
              .Hsync(HS),
              .sw(sw),
              .led(led)
   );

       reg good_HS;
       reg good_VS;
       reg activeH;
       reg activeV;

// Clock parameters
   parameter PERIOD = 10;
   parameter real DUTY_CYCLE = 0.5;
   parameter OFFSET = 2;

       initial
       begin
              clkin = 1'b0;
          #OFFSET
              clkin = 1'b1;
      forever
         begin
            #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
         end
       end

     initial
     begin
       sw = 16'b0;
       btnU = 1'b0;
```

```
      btnC = 1'b0;
      btnD = 1'b0;
      btnL = 1'b0;
      btnR = 1'b0;
      greset = 1'b0;
      #600 greset = 1'b1;
      #80 greset = 1'b0;
    end

  // process to generate correct values for HS and activeH
  initial
      begin
          #OFFSET;
          #0.1;

              activeH = 1'b1;
       good_HS = 1'b1;
       #880;
       #700; // to get past gsr pulse and clock delay
       // comment out the line below if your HSync is not passed through a
       // D FF before leaving the FPGA
       #40;  // one more clock for FF delay output (this line may need to be
commented out)
          forever
                 begin
                      activeH = 1'b1;
                      #(640*40);
                      activeH = 1'b0;
                      #(15*40);
                      good_HS = 1'b0;
                      #((96)*40);
                      good_HS = 1'b1;
                      #(49*40);
                 end
           end

  //correct values for VS and activeV
  initial
      begin
          #OFFSET;
          #0.1;
          activeV = 1'b1;
          good_VS = 1'b1;
          #880;
          #700; // to get past gsr and DCM clock delay
          // comment out the line below if your VSync is not passed through a
```

```
          // D FF before leaving the FPGA
          #40;  // one more clock for FF delay output (this line may need to be
commented out)
             forever
                 begin
                      activeV = 1'b1;
                      #(480*800*40);
                      activeV=1'b0;
                      #(9*800*40);
                      good_VS = 1'b0;
                      #(2*800*40);
                      good_VS = 1'b1;
                      #(34*800*40);
                 end
             end



// Instantiate the module comparing good and actual sync signals
   check_the_sync_signals GUT (
             .myHS(HS),
             .myVS(VS),
             .correct_HS(good_HS),
             .correct_VS(good_VS),
             .clk(clkin),
             .greset(greset),
             .sync_error(oops)
   );

// Instantiate the module checking RGB signals are
// low outside the active region
   check_the_rgb_signals RUT (.VB(vgaBlue), .VG(vgaGreen), .VR(vgaRed),
             .activeH(activeH), .activeV(activeV),
             .clk(clkin),
             .greset(greset),
             .rgb_error(rgb_oops)
   );

endmodule

module check_the_sync_signals(
 input myHS,
 input myVS,
 input correct_HS,
 input correct_VS,
 input clk,
```

```
 input greset,
 output sync_error);

  // sync_error is high when actual and expected sync signals differ
 assign sync_error = (myHS^correct_HS)|(myVS^correct_VS);

  // SERRORS is incremented when sync_error is high at the rising edge of betterclk
 integer SERRORS;

  always @(posedge clk)
   begin
          SERRORS  <= SERRORS + sync_error;   // non-blocking assignment
      end

  // reset SERRORS on rising edge of gsr
  always @(posedge greset)
   begin
          SERRORS <= 32'b0;   //  non-blocking assignment
      end

endmodule

module check_the_rgb_signals(VB, VG, VR,
                    activeH, activeV, clk, greset, rgb_error);

 input [3:0] VB, VG, VR;
 input activeH, activeV;
 input clk, greset;
 output rgb_error;

  // rgb_error is high when any RGB output is high outside the active region
 assign rgb_error = ((|VR)|(|VB)|(|VG))&~(activeH*activeV);

  // RGBERRORS is incremented when rgb_error is high at the rising edge of betterclk
 integer RGBERRORS;

 always @(posedge clk)
   begin
          RGBERRORS  <= RGBERRORS + rgb_error;   // non-blocking assignment
      end

      // reset SERRORS on rising edge of gsr
 always @(posedge greset)
   begin
          RGBERRORS <= 32'b0;   //  non-blocking assignment
      end
```

```
endmodule
```