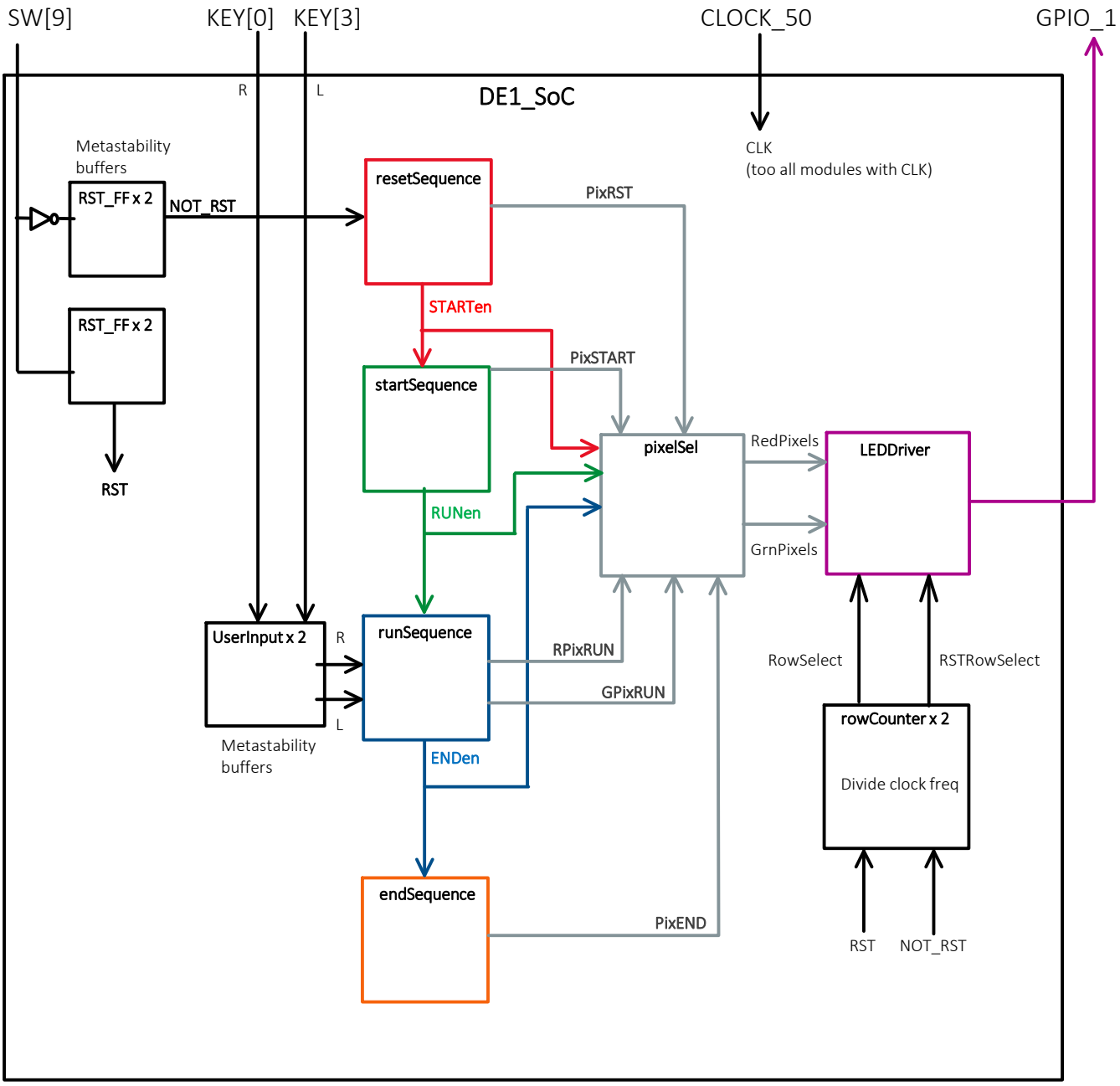
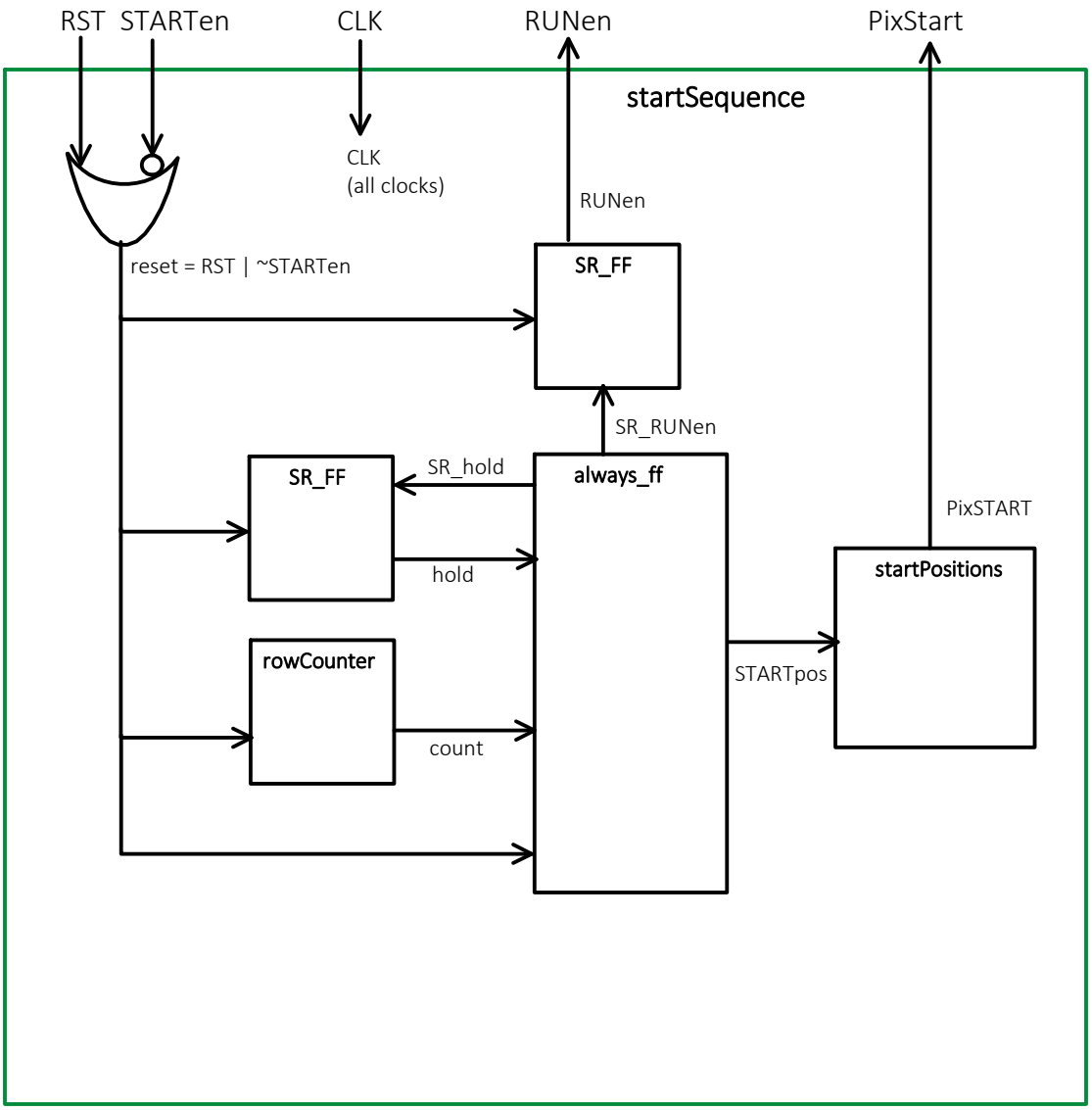
Asteroid Avoider - Module descriptions

MODULE	DESCRIPTION
DE1_SoC	• Top-level module for board
resetSequence	• Sets display pixels for reset condition • Outputs start sequence enable flag when finished
startSequence	• Sets display pixels for start condition • Outputs run sequence enable flag when finished
runSequence	• Sets display pixels for run condition • Outputs end sequence enable flag when finished
endSequence	• Sets display pixels for end condition
UserInput	• Outputs single pulse for high input • Used for setting key inputs to game
pixelSel	• Enables pixel sets depending on sequence enable flags
LEDDriver	• Maps selected pixels to GPIO_1 for 16x16 display
startPositions	• Maps given position to set of pixels for display
collisionCompare	• Maps Green and Red Pixels grids to single 16x16 "check" grid • Check grid will have logic 1 anywhere green and red pixels are logic 1 at same time. • Check grid is fed to collisionCheck to get a single logical value.
collisionCheck	• Outputs logic 1 anytime there is a logic 1 present anywhere in the check grid
countersel	• Selects rowcounter count speed based on given input • Used for adjusting game speed with switches
rowcounter[n]	• Counters 0->7 map a count with n = 27->20, where • Count frequency = Clock frequency / (2^{n+1}). • Can also adjust n by instantiating different parameter.
asteroidLocation	• Maps asteroid pixel location. • Uses LSFR10b to generate random column choice. • Uses choiceToggle to latch onto new random column at beginning of count sequence. • Uses columnChoice to pick column pixel for current row count.
choiceToggle	• Choice Toggle updates column choice and beginning of count sequence
columnChoice	• Column Choice maps a 4-bit choice to a 16-bit pixel row
shipLocation	• Maps user inputs to ship pixel location
shipPositions	• Maps 4-bit position choice to a 16x16 pixel grid
endExplode	• Maps 4-bit position choice to a 16x16 pixel grid
LSFR10b	• 10-bit Linear Feedback Shift Register for generating pseudorandom numbers.
D_FF	• D-type flip flop mainly used for buffering/metastability.
SR_FF	• SR-type flip flop for latching enable signals.
RST_FF	• D-type without reset input used for reset sequence.

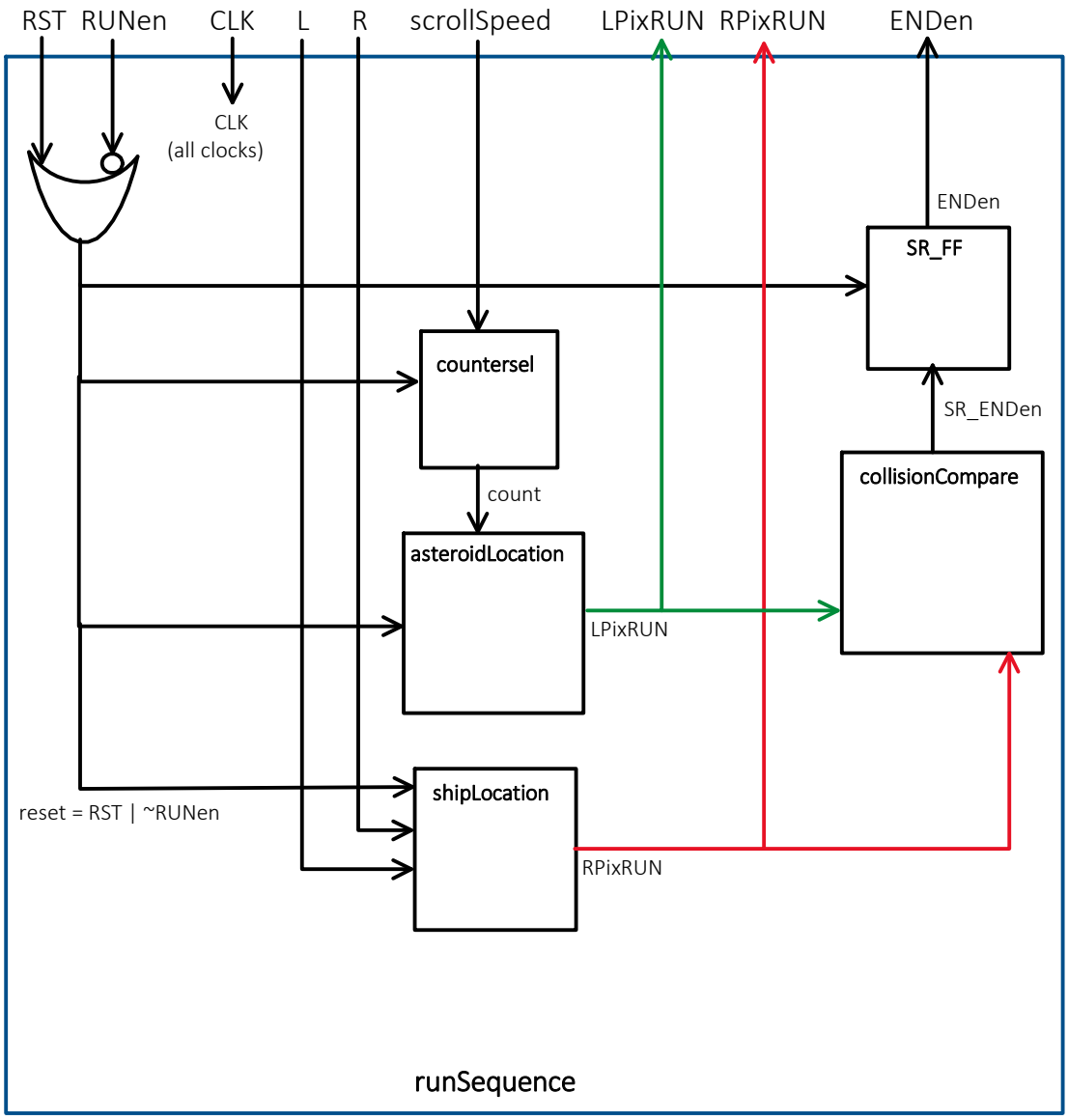
Asteroid Avoider - Top level block diagram



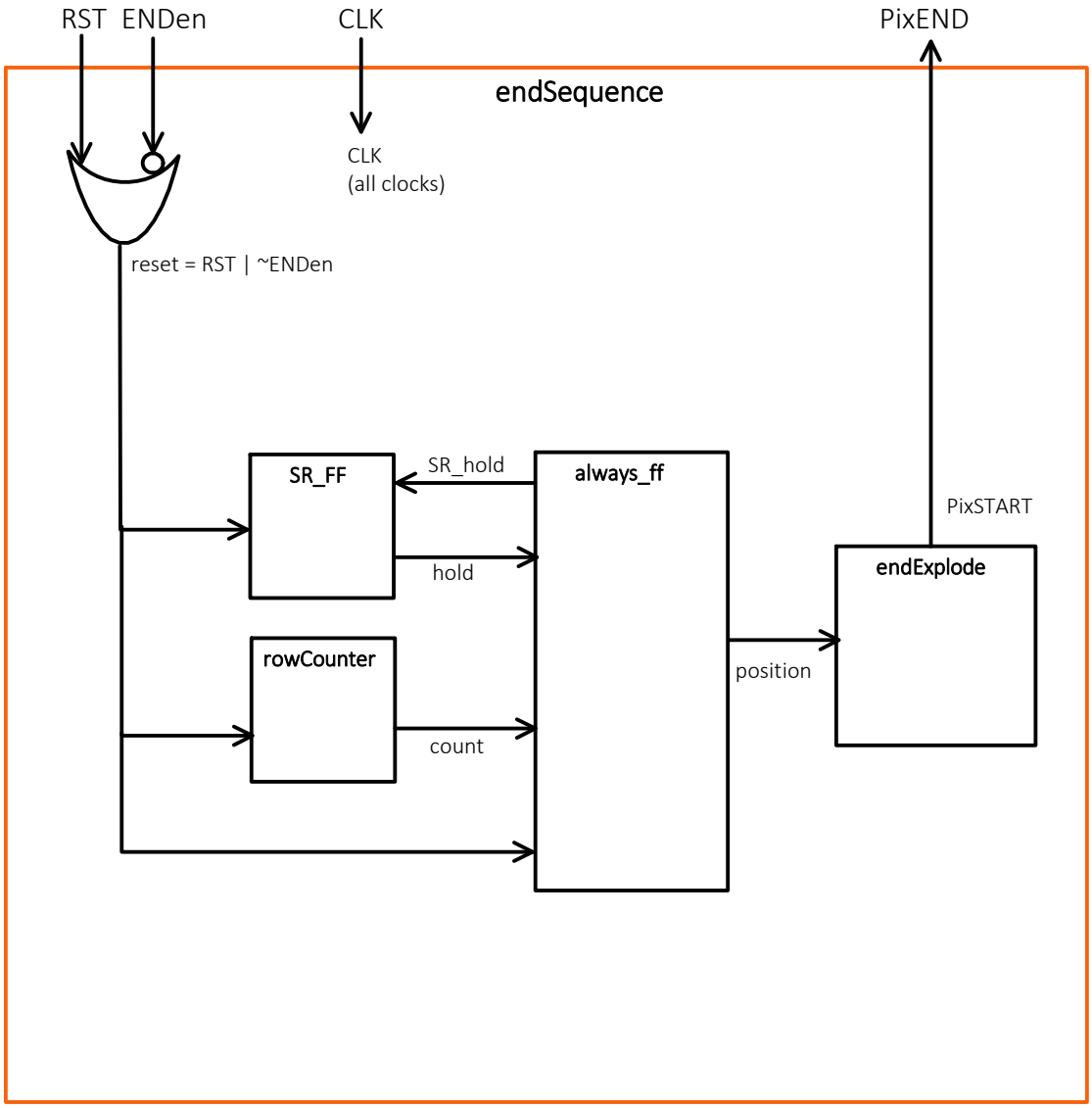
Asteroid Avoider - startSequence block diagram



Asteroid Avoider - runSequence block diagram



Asteroid Avoider - endSequence block diagram



```
1   // James Lee
2   // University of Washington
3   // EE 271 Lab 8
4   // Fall 2021
5
6   // NOTE: This project has modified versions of the EE 271 Winter 2021 peripheral tutorial
    source files
7
8   // References (Last accessed 12/7/2021)
9   //    1. Course Page: https://class.ece.uw.edu/271/hauck2/de1/index.html
10  //    2. Starter Files: https://class.ece.uw.edu/271/hauck2/de1/LEDboard/led_driver.zip
11
12  // Top-level module that defines the I/Os for the DE-1 SoC board
13  module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, SW, LEDR, GPIO_1, CLOCK_50);
14     output logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
15     output logic [9:0]  LEDR;
16     input  logic [3:0]  KEY;
17     input  logic [9:0]  SW;
18     output logic [35:0] GPIO_1;
19     input logic CLOCK_50;
20
21     // Turn off HEX displays
22     assign HEX0 = '1;
23     assign HEX1 = '1;
24     assign HEX2 = '1;
25     assign HEX3 = '1;
26     assign HEX4 = '1;
27     assign HEX5 = '1;
28
29     // Assign reset input
30     logic RST, NOT_RST, qRST0, qNRST0, dRST, dNRST; // reset - toggle this on startup
31     assign dRST = SW[9];
32     assign dNRST = ~SW[9];
33
34     // feed SW9 to RST, NOT_RST through 2 DFF to prevent metastable inputs
35     RST_FF DRST0 (.q(qRST0), .d(dRST), .CLK(CLOCK_50)); // input SW9
36     RST_FF DRST1 (.q(RST), .d(qRST0), .CLK(CLOCK_50)); // feed to next DFF
37     RST_FF DNRST0 (.q(qNRST0), .d(dNRST), .CLK(CLOCK_50)); // input ~SW9
38     RST_FF DNRST1 (.q(NOT_RST), .d(qNRST0), .CLK(CLOCK_50)); // feed to next DFF
39
40     /* Set up LED base clock
41     ================================================================
42     *  CLOCK_50 = System clock for key and switch inputs (50 MHz)
43     *  LED_CLOCK = top-level LED clock
44        *  LED clock chosen for max brightness and min flicker
45     *  Example divider calculation
46        *  clk[14] --> (50 MHz / 2**(14+1)) = 1526 Hz
47      ================================================================*/
48
49  // logic [31:0] clk;
50  // logic LED_CLOCK;
51  //
52  // clock_divider divider (.clock(CLOCK_50), .divided_clocks(clk));
53  //
54  // assign LED_CLOCK = clk[14]; // 1526 Hz clock signal
55
56     /* If you notice flickering, set LED_CLOCK faster.
57     However, this may reduce the brightness of the LED board. */
58
59
60     /* Reset sequence
61     ================================================================
62     Sets display for reset condition
63     Outputs START flag when sequence finished
64     ================================================================ */
65     logic STARTen;
66     logic [15:0][15:0] PixRST;
67
68     resetSequence RSTseq0 (.NOT_RST, .CLK(CLOCK_50), .PixRST, .STARTen);
69
70
71     /* Start sequence
72     ================================================================
```

```
    Starts on STARTen flag from reset sequence
    Sets display for start condition
    Outputs RUN enable flag when sequence finished
    ================================================================ */
    logic RUNen; // Run sequence enable
    logic [15:0][15:0] PixSTART;

    startSequence STARTSeq0 (.RST, .CLK(CLOCK_50), .PixSTART, .STARTen, .RUNen);

    /* Run sequence
    ================================================================
    Starts on RUNen flag from start sequence
    Sets display for reset condition
    Outputs END enable flag when sequence finished
    ================================================================ */
    logic ENDen; // End sequence enables
    logic [15:0][15:0] RPixRUN, GPixRUN;

    // User inputs
    logic L, R;
    UserInput RKey(.out(R), .in(~KEY[0]), .CLK(CLOCK_50), .RST); // move right
    UserInput LKey(.out(L), .in(~KEY[3]), .CLK(CLOCK_50), .RST); // move left

    // Set scroll speed with switch inputs, use D_FF for metastability
    logic q0SW0, q1SW0, q0SW1, q1SW1, q0SW2, q1SW2;
    D_FF Dq0SW0 (.q(q0SW0), .d(SW[0]), .RST, .CLK(CLOCK_50));
    D_FF Dq1SW0 (.q(q1SW0), .d(q0SW0), .RST, .CLK(CLOCK_50));
    D_FF Dq0SW1 (.q(q0SW1), .d(SW[1]), .RST, .CLK(CLOCK_50));
    D_FF Dq1SW1 (.q(q1SW1), .d(q0SW1), .RST, .CLK(CLOCK_50));
    D_FF Dq0SW2 (.q(q0SW2), .d(SW[2]), .RST, .CLK(CLOCK_50));
    D_FF Dq1SW2 (.q(q1SW2), .d(q0SW2), .RST, .CLK(CLOCK_50));

    logic [2:0] scrollSpeed;
    assign scrollSpeed = {q1SW2, q1SW1, q1SW0};

    runSequence RUNseq0 (.CLK(CLOCK_50), .RST, .L, .R, .scrollSpeed, .RPixRUN, .GPixRUN, .
RUNen, .ENDen);

    /* End sequence
    ================================================================
    Starts on ENDen flag from start sequence
    ================================================================ */
    logic [15:0][15:0] PixEND;

    endSequence ENDseq0 (.RST, .CLK(CLOCK_50), .ENDen, .PixEND);


    /* Pixel Selector
    ================================================================
    Choses which pixel configuration to input to LEDdriver
    Based on enable flags
    ================================================================ */
    pixelSel pixelSel0 (.RST, .CLK(CLOCK_50), .PixRST, .PixSTART, .RPixRUN, .GPixRUN, .PixEND
, .STARTen, .RUNen, .ENDen, .RedPixels, .GrnPixels);

    /* LED board driver
    ================================================================ */
    logic [15:0][15:0]RedPixels; // 16 x 16 array representing red LEDs
    logic [15:0][15:0]GrnPixels; // 16 x 16 array representing green LEDs

    // Set counter to adust refresh rate

    logic [3:0] RowSelect, RSTRowSelect;
    rowcounter0 #(.FREQDIV(14)) LEDrow0 (.CLK(CLOCK_50), .RST, .MSB0(RowSelect), .
EnableCount0(1));
    rowcounter0 #(.FREQDIV(14)) LEDrow1 (.CLK(CLOCK_50), .RST(NOT_RST), .MSB0(RSTRowSelect),
.EnableCount0(1));

    // Modified LEDDriver to put counter in separate module
    LEDDriver Driver (.RST, .RedPixels, .GrnPixels, .GPIO_1, .RowSelect, .RSTRowSelect);
endmodule
```

```
1   module resetSequence (NOT_RST, CLK, PixRST, STARTen);
2      input logic NOT_RST, CLK;
3      output logic STARTen;
4      output logic [15:0][15:0] PixRST;
5
6         assign PixRST[00] = 16'b0001110001111000;
7         assign PixRST[01] = 16'b0001001001000000;
8         assign PixRST[02] = 16'b0001110001111000;
9         assign PixRST[03] = 16'b0001100001000000;
10        assign PixRST[04] = 16'b0001010001111000;
11        assign PixRST[05] = 16'b0000000000000000;
12        assign PixRST[06] = 16'b0000000000000000;
13        assign PixRST[07] = 16'b0000000000000000;
14        assign PixRST[08] = 16'b0000000000000000;
15        assign PixRST[09] = 16'b0000000000000000;
16        assign PixRST[10] = 16'b0000000000000000;
17        assign PixRST[11] = 16'b1111011110111110;
18        assign PixRST[12] = 16'b1000010000001000;
19        assign PixRST[13] = 16'b1111011110001000;
20        assign PixRST[14] = 16'b0001010000001000;
21        assign PixRST[15] = 16'b1111011110001000;
22
23     logic q0;
24     RST_FF DRST0 (.q(q0), .d(NOT_RST), .CLK);
25     RST_FF DRST1 (.q(STARTen), .d(q0), .CLK);
26  endmodule
27
28  //module resetSequence_testbench();
29  // logic NOT_RST, CLK;
30  // logic STARTen;
31  // logic [15:0][15:0] PixRST;
32  //
33  // resetSequence dut (.NOT_RST, .CLK, .STARTen, .PixRST);
34  //
35  // // Set up a simulated  clock
36  // parameter CLOCK_PERIOD = 100;
37  // initial begin
38  //    CLK <= 0;
39  //    forever # (CLOCK_PERIOD/2) CLK <= ~CLK;  // Forever toggle clock
40  // end
41  //
42  // initial begin
43  //    NOT_RST <= 0; repeat(2) @(posedge CLK);
44  //    NOT_RST <= 1; repeat(5) @(posedge CLK);
45  //    $stop;
46  // end
47  //endmodule
```

```
1   module startSequence (RST, CLK, PixSTART, STARTen, RUNen);
2      input logic RST, CLK, STARTen;
3      output logic [15:0][15:0] PixSTART;
4      output logic RUNen;
5
6      logic reset;
7      assign reset = RST | ~STARTen;
8
9      // count 0.75Hz
10     logic [3:0] count;
11     // Uncomment one
12     rowcounter0 #(.FREQDIV(25)) startCounter0 (.CLK, .RST(reset), .MSB0(count), .EnableCount0
       (STARTen)); // board
13  // rowcounter0 #(.FREQDIV(2)) startCounter0 (.CLK, .RST(reset), .MSB0(count),
       .EnableCount0(STARTen)); // simulation
14
15     logic [3:0] ps,ns;
16     logic [1:0] SR_hold, SR_RUNen;
17     logic hold;
18
19     // SR latches for holding final display and RUNen signal
20     SR_FF holdSR (.SR(SR_hold), .Q(hold), .CLK, .RST(reset));
21     SR_FF RUNSR (.SR(SR_RUNen), .Q(RUNen), .CLK, .RST(reset));
22
23
24     always_ff @(posedge CLK) begin
25        if (reset) begin // reset to 0
26           ps <= '0;
27           SR_hold <= 2'b01;
28           SR_RUNen <= 2'b01;
29        end else if (count == 4) begin
30           ps <= 4;
31           SR_hold <= 2'b10;
32        end else if (count == 5) begin
33           ps <= 5;
34           SR_RUNen <= 2'b10;
35        end else if (hold) begin
36           ps <= 6;
37        end else
38           ps <= count;
39     end
40
41     // Instantiate start sequence positions
42
43     logic [3:0] STARTpos;
44     assign STARTpos = ps;
45
46     startPositions startPos0 (.STARTpos, .PixSTART);
47  endmodule
48
49  //module startSequence_testbench();
50  // logic RST, CLK, STARTen;
51  // logic [15:0][15:0] PixSTART;
52  // logic RUNen;
53  //
54  // // Set up a simulated clock
55  // parameter CLOCK_PERIOD=100;
56  // initial begin
57  //    CLK <= 0;
58  //    forever #(CLOCK_PERIOD/2) CLK <= ~CLK;// Forever toggle the clock
59  // end
60  //
61  // startSequence dut (.RST, .CLK, .PixSTART, .STARTen, .RUNen);
62  //
63  // initial begin
64  //    RST <= 1; STARTen <= 0; repeat(1) @(posedge CLK);
65  //    RST <= 0; repeat(1) @(posedge CLK);
66  //    STARTen <= 1; repeat(60) @(posedge CLK);
67  //    $stop;
68  // end
69  //endmodule
```

```
1   module runSequence (CLK, RST, L, R, scrollSpeed, RPixRUN, GPixRUN, RUNen, ENDen);
2      input logic CLK, RST, L, R, RUNen;
3      input logic [2:0] scrollSpeed;
4      output logic [15:0][15:0] RPixRUN, GPixRUN;
5      output logic ENDen;
6
7      logic reset;
8      assign reset = RST | ~RUNen; // start when enabled and not reset
9
10     logic [15:0][15:0] RedPixels, GrnPixels;
11
12     /* Collision comparator
13     ==========================*/
14
15
16     // collision when green and red on at same time in LED
17     logic check;
18     collisionCompare collisionComp0 (.RedPixels, .GrnPixels, .check);
19
20     logic [1:0] SR;
21     assign SR = {check,1'b0};
22     SR_FF collision0 (.SR, .Q(ENDen), .CLK, .RST(reset));
23
24     /* astroid pixels (green)
25     ==========================*/
26
27     // scroll speed selector
28     logic [3:0] ROW;
29     countersel speedsel0 (.CLK, .RST(reset), .MSB(ROW), .whichCounter(scrollSpeed));
30     // asteroid pixel location
31     asteroidLocation astLoc0 (.CLK, .RST(reset), .ROW, .GrnPixels);
32
33     /* Ship pixels (red)
34     ==========================*/
35
36     // Ship location
37     shipLocation ship0 (.RedPixels, .L, .R, .RST(reset), .CLK);
38
39     assign RPixRUN = RedPixels;
40     assign GPixRUN = GrnPixels;
41  endmodule
42
43  //module runSequence_testbench();
44  // input logic CLK, LED_CLOCK, RST, L, R;
45  // input logic [2:0] scrollSpeed;
46  // output logic [15:0][15:0] RPixRUN, GPixRUN;
47  // output logic ENDen;
48  //endmodule
```

```
1   module endSequence (RST, CLK, ENDen, PixEND);
2      input logic RST, CLK, ENDen;
3      output logic [15:0][15:0] PixEND;
4
5
6      logic [3:0] ps;
7      logic [1:0] S_hold;
8      logic hold;
9
10     // SR latch for holding final end display
11     SR_FF holdSR (.SR(S_hold), .Q(hold), .CLK, .RST);
12
13     logic reset;
14     assign reset = RST | ~ENDen;
15
16     logic [3:0] count;
17     // Uncomment 1 counter
18     rowcounter0 #(.FREQDIV(25)) endcounter (.CLK, .RST, .MSB0(count), .EnableCount0(ENDen));
   // board
19 // rowcounter0 #(.FREQDIV(2)) endcounter (.CLK, .RST, .MSB0(count), .EnableCount0(ENDen));
   // simulation
20
21     always_ff @(posedge CLK) begin
22        if (reset) begin // reset to 0
23           ps <= '0;
24           S_hold <= 2'b01;
25        end else if (count == 4'b0110) begin
26           ps <= count;
27           S_hold <= 2'b10;
28        end else if (hold) begin
29           ps <= 4'b0110;
30        end else
31           ps <= count;
32     end
33
34     // Instantiate end sequence positions
35     logic [3:0] position;
36     assign position = ps;
37
38     endExplode end0 (.position, .PixEND);
39  endmodule
40
41  //module endSequence_testbench();
42  // logic RST, CLK, ENDen;
43  // logic [15:0][15:0] PixEND;
44  //
45  // // Set up a simulated clock
46  // parameter CLOCK_PERIOD=100;
47  // initial begin
48  //    CLK <= 0;
49  //    forever #(CLOCK_PERIOD/2) CLK <= ~CLK;// Forever toggle the clock
50  // end
51  //
52  // endSequence dut (.RST, .CLK, .ENDen, .PixEND);
53  //
54  // integer i;
55  // initial begin
56  //    RST <= 1; ENDen <= 0; repeat(2) @(posedge CLK);
57  //    RST <= 0; repeat(2) @(posedge CLK);
58  //    ENDen <= 1; repeat(80) @(posedge CLK);
59  //    $stop;
60  // end
61  //endmodule
```

```
1   // User Input signal translates key press into single pulse
2   module UserInput (out, in, CLK, RST);
3      input logic in, CLK, RST;
4      output logic out;
5
6      // Set up 3 DFF in series, which will all be offset by a clock cycle
7      logic q0, q1, q2;
8      D_FF DFF0 (.q(q0), .d(in), .RST, .CLK);
9      D_FF DFF1 (.q(q1), .d(q0), .RST, .CLK);
10     D_FF DFF2 (.q(q2), .d(q1), .RST, .CLK);
11
12     assign out = q1 & ~q2; // output only one cycle when q1 and q2 are offset
13
14  endmodule
15
16  //module UserInput_testbench();
17  //
18  // logic CLOCK_50, KEY[3:0], SW[9:0], out;
19  //
20  // UserInput dut (.out, .in(KEY[3]), .CLK(CLOCK_50), .RST(SW[9]));
21  //
22  // // Set up a simulated  clock
23  // parameter CLOCK_PERIOD = 100;
24  // initial begin
25  //    CLOCK_50 <= 0;
26  //    forever # (CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // Forever toggle clock
27  // end
28  //
29  // initial begin
30  //                    repeat(1) @(posedge CLOCK_50);
31  //    SW[9]    <= 1; KEY[3] <= 0; repeat(4) @(posedge CLOCK_50); // reset on, key press off
32  //    SW[9]    <= 0; repeat(1) @(posedge CLOCK_50);   // reset off
33  //    KEY[3]   <= 1; repeat(4) @(posedge CLOCK_50);   // key press on (long)
34  //    KEY[3]   <= 0; repeat(4) @(posedge CLOCK_50);   // key press off
35  //    KEY[3]   <= 1; repeat(1) @(posedge CLOCK_50);   // key press on (short)
36  //    KEY[3]   <= 0; repeat(4) @(posedge CLOCK_50);   // key press off
37  //    KEY[3]   <= 1; repeat(1) @(posedge CLOCK_50);   // key press on
38  //    SW[9]    <= 1; repeat(5) @(posedge CLOCK_50);   // reset on
39  //    $stop;   // end simulation
40  // end
41  //endmodule
```

```
1   // James Lee
2   // University of Washington
3   // EE 271 Lab 8
4   // Fall 2021
5
6   // NOTE: This project has modified versions of the EE 271 Winter 2021 peripheral tutorial
    source files
7
8   // References (Last accessed 12/7/2021)
9   //    1. Course Page: https://class.ece.uw.edu/271/hauck2/de1/index.html
10  //    2. Starter Files: https://class.ece.uw.edu/271/hauck2/de1/LEDboard/led_driver.zip
11
12  module pixelSel (RST, CLK, PixRST, PixSTART, RPixRUN, GPixRUN, PixEND, STARTen, RUNen, ENDen
    , RedPixels, GrnPixels);
13     input logic RST, CLK, STARTen, RUNen, ENDen;
14     input logic [15:0][15:0] PixRST, PixSTART, RPixRUN, GPixRUN, PixEND;
15     output logic [15:0][15:0] RedPixels, GrnPixels;
16
17     always_ff @(posedge CLK) begin
18        if (ENDen) begin // End Condition
19           RedPixels <= PixEND;
20           GrnPixels <= '0;
21        end
22
23        else if (RUNen) begin // Run condition
24           RedPixels <= RPixRUN;
25           GrnPixels <= GPixRUN;
26        end
27
28        else if (STARTen) begin // Start condition
29           RedPixels <= PixSTART;
30           GrnPixels <= '0;
31        end
32
33        else begin // Reset condition
34           RedPixels <= '0;
35           GrnPixels <= PixRST;
36        end
37     end
38
39  endmodule
40
41  //module pixelSel_testbench();
42  // logic RST, CLK, STARTen, RUNen, ENDen;
43  // logic [15:0][15:0] PixRST, PixSTART, PixRUN, PixEND;
44  // logic [15:0][15:0] RedPixels;
45  //
46  // shipPositions pos0 (.position(4'b0000), .RedPixels(PixRST));
47  // shipPositions pos1 (.position(4'b0001), .RedPixels(PixSTART));
48  // shipPositions pos2 (.position(4'b0010), .RedPixels(PixRUN));
49  // shipPositions pos3 (.position(4'b0011), .RedPixels(PixEND));
50  //
51  // // Set up a simulated  clock
52  // parameter CLOCK_PERIOD = 100;
53  // initial begin
54  //    CLK <= 0;
55  //    forever # (CLOCK_PERIOD/2) CLK <= ~CLK;  // Forever toggle clock
56  // end
57  //
58  // pixelSel dut (.RST, .CLK, .PixRST, .PixSTART, .RPixRUN, .GPixRUN, .PixEND, .STARTen,
    .RUNen, .ENDen, .RedPixels);
59  //
60  // initial begin
61  //    {ENDen, RUNen, STARTen} <= 3'b000;
62  //    RST <= 1;            repeat(10) @(posedge CLK);
63  //    {RST, STARTen} <= 2'b01;   repeat(10) @(posedge CLK);
64  //    RUNen <= 1;          repeat(10) @(posedge CLK);
65  //    ENDen <= 1;          repeat(10) @(posedge CLK);
66  //    $stop;
67  // end
68  //endmodule
```

```
1   // James Lee
2   // University of Washington
3   // EE 271 Lab 8
4   // Fall 2021
5
6   // NOTE: This project has modified versions of the EE 271 Winter 2021 peripheral tutorial
    source files
7
8   // References (Last accessed 12/7/2021)
9   //    1. Course Page: https://class.ece.uw.edu/271/hauck2/de1/index.html
10  //    2. Starter Files: https://class.ece.uw.edu/271/hauck2/de1/LEDboard/led_driver.zip
11
12  // A driver for the 16x16x2 LED display expansion board.
13  // Read below for an overview of the ports.
14  // IMPORTANT: You do not need to necessarily modify this file. But if you do, be sure you
    know what you are doing.
15
16  // FREQDIV: (Parameter) Sets the scanning speed (how often the display cycles through rows)
17  //          The CLK input divided by 2^(FREQDIV) is the interval at which the driver
    switches rows.
18  // GPIO_1: (Output) The 36-pin GPIO1 header, as on the DE1-SoC board.
19  // RedPixels: (Input) A 16x16 array of logic items corresponding to the red pixels you'd
    like to have lit on the display.
20  // GrnPixels: (Input) A 16x16 array of logic items corresponding to the green pixels you'd
    like to have lit on the display.
21  // EnableCount: (Input) Whether to continue moving through the rows.
22  // CLK: (Input) The system clock.
23  // RST: (Input) Resets the display driver. Required during startup before use.
24  module LEDDriver (RST, GPIO_1, RedPixels, GrnPixels, RowSelect, RSTRowSelect);
25      output logic [35:0] GPIO_1;
26      input logic [15:0][15:0] RedPixels ;
27      input logic [15:0][15:0] GrnPixels ;
28      input logic [3:0] RowSelect, RSTRowSelect;
29      input logic RST;
30
31      logic [3:0] whichRowSelect;
32
33      always_comb begin
34        case (RST)
35           0: whichRowSelect = RowSelect;
36           default: whichRowSelect = RSTRowSelect;
37        endcase
38      end
39
40      assign GPIO_1[35:32] = whichRowSelect;
41      assign GPIO_1[31:16] = { GrnPixels[whichRowSelect][0], GrnPixels[whichRowSelect][1],
    GrnPixels[whichRowSelect][2], GrnPixels[whichRowSelect][3], GrnPixels[whichRowSelect][4],
    GrnPixels[whichRowSelect][5], GrnPixels[whichRowSelect][6], GrnPixels[whichRowSelect][7],
    GrnPixels[whichRowSelect][8], GrnPixels[whichRowSelect][9], GrnPixels[whichRowSelect][10],
    GrnPixels[whichRowSelect][11], GrnPixels[whichRowSelect][12], GrnPixels[whichRowSelect][13],
     GrnPixels[whichRowSelect][14], GrnPixels[whichRowSelect][15] };
42      assign GPIO_1[15:0] = { RedPixels[whichRowSelect][0], RedPixels[whichRowSelect][1],
    RedPixels[whichRowSelect][2], RedPixels[whichRowSelect][3], RedPixels[whichRowSelect][4],
    RedPixels[whichRowSelect][5], RedPixels[whichRowSelect][6], RedPixels[whichRowSelect][7],
    RedPixels[whichRowSelect][8], RedPixels[whichRowSelect][9], RedPixels[whichRowSelect][10],
    RedPixels[whichRowSelect][11], RedPixels[whichRowSelect][12], RedPixels[whichRowSelect][13],
     RedPixels[whichRowSelect][14], RedPixels[whichRowSelect][15] };
43  endmodule
44
45  //module LEDDriver_Test();
46  //    logic CLK, RST, EnableCount;
47  //    logic [15:0][15:0]RedPixels;
48  //    logic [15:0][15:0]GrnPixels;
49  //    logic [35:0] GPIO_1;
50  //
51  //    LEDDriver #(.FREQDIV(2)) Driver(.GPIO_1, .RedPixels, .GrnPixels, .EnableCount, .CLK,
    .RST);
52  //
53  //    initial
54  //    begin
55  //        CLK <= 1'b0;
56  //        forever #50 CLK <= ~CLK;
57  //    end
```

```
58  //
59  //    initial
60  //    begin
61  //        EnableCount <= 1'b0;
62  //        RedPixels <= '{default:0};
63  //        GrnPixels <= '{default:0};
64  //        @(posedge CLK);
65  //
66  //        RST <= 1; @(posedge CLK);
67  //        RST <= 0; @(posedge CLK);
68  //        @(posedge CLK); @(posedge CLK); @(posedge CLK);
69  //
70  //        GrnPixels[1][1] <= 1'b1; @(posedge CLK);
71  //        EnableCount <= 1'b1; @(posedge CLK); #1000;
72  //        RedPixels[2][2] <= 1'b1;
73  //        RedPixels[2][3] <= 1'b1;
74  //        GrnPixels[2][3] <= 1'b1; @(posedge CLK); #1000;
75  //        EnableCount <= 1'b0; @(posedge CLK); #1000;
76  //        GrnPixels[1][1] <= 1'b0; @(posedge CLK);
77  //        $stop;
78  //
79  //    end
80  //endmodule
81  //
82  //module LEDDriver_TestPhysical(CLOCK_50, RST, Speed, GPIO_1);
83  //    input logic CLOCK_50, RST;
84  //    input logic [9:0] Speed;
85  //    output logic [35:0] GPIO_1;
86  //    logic [15:0][15:0]RedPixels;
87  //    logic [15:0][15:0]GrnPixels;
88  //    logic [31:0] Counter;
89  //    logic EnableCount;
90  //
91  //    LEDDriver #(.FREQDIV(15)) Driver (.CLK(CLOCK_50), .RST, .EnableCount, .RedPixels,
    .GrnPixels, .GPIO_1);
92  //
93  //    //                          F E D C B A 9 8 7 6 5 4 3 2 1 0
94  //    assign RedPixels[00] = '{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
95  //    assign RedPixels[01] = '{1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1};
96  //    assign RedPixels[02] = '{1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1};
97  //    assign RedPixels[03] = '{1,0,1,1,0,0,0,0,0,0,0,0,1,1,0,1};
98  //    assign RedPixels[04] = '{1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1};
99  //    assign RedPixels[05] = '{1,0,1,0,1,1,0,0,0,0,1,1,0,1,0,1};
100 //    assign RedPixels[06] = '{1,0,1,0,1,0,1,1,1,1,0,1,0,1,0,1};
101 //    assign RedPixels[07] = '{1,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1};
102 //    assign RedPixels[08] = '{1,0,1,0,1,0,1,1,0,1,0,1,0,1,0,1};
103 //    assign RedPixels[09] = '{1,0,1,0,1,0,1,1,1,1,0,1,0,1,0,1};
104 //    assign RedPixels[10] = '{1,0,1,0,1,1,0,0,0,0,1,1,0,1,0,1};
105 //    assign RedPixels[11] = '{1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1};
106 //    assign RedPixels[12] = '{1,0,1,1,0,0,0,0,0,0,0,0,1,1,0,1};
107 //    assign RedPixels[13] = '{1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1};
108 //    assign RedPixels[14] = '{1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1};
109 //    assign RedPixels[15] = '{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
110 //
111 //    assign GrnPixels[00] = '{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
112 //    assign GrnPixels[01] = '{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0};
113 //    assign GrnPixels[02] = '{0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0};
114 //    assign GrnPixels[03] = '{0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0};
115 //    assign GrnPixels[04] = '{0,1,0,1,1,0,0,0,0,0,0,1,1,0,1,0};
116 //    assign GrnPixels[05] = '{0,1,0,1,0,1,1,1,1,1,1,1,0,1,0,1,0};
117 //    assign GrnPixels[06] = '{0,1,0,1,0,1,1,0,0,1,1,0,1,0,1,0};
118 //    assign GrnPixels[07] = '{0,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0};
119 //    assign GrnPixels[08] = '{0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,0};
120 //    assign GrnPixels[09] = '{0,1,0,1,0,1,1,0,0,1,1,0,1,0,1,0};
121 //    assign GrnPixels[10] = '{0,1,0,1,0,1,1,1,1,1,1,0,1,0,1,0};
122 //    assign GrnPixels[11] = '{0,1,0,1,1,0,0,0,0,0,0,1,1,0,1,0};
123 //    assign GrnPixels[12] = '{0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0};
124 //    assign GrnPixels[13] = '{0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0};
125 //    assign GrnPixels[14] = '{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0};
126 //    assign GrnPixels[15] = '{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
127 //
128 //    always_ff @(posedge CLOCK_50)
129 //    begin
```

```
130   //         if(RST) Counter <= 'b0;
131   //         else
132   //         begin
133   //             Counter <= Counter + 1'b1;
134   //             if(Counter >= Speed)
135   //             begin
136   //                 EnableCount <= 1'b1;
137   //                 Counter <= 'b0;
138   //             end
139   //             else EnableCount <= 1'b0;
140   //         end
141   //     end
142   //endmodule
```

```
1   // Start Positions maps given STARTpos to set of pixels for display
2   module startPositions (STARTpos, PixSTART);
3      input logic [3:0] STARTpos;
4      output logic [15:0][15:0] PixSTART;
5
6      always_comb begin
7         case (STARTpos)
8            0: PixSTART = '0;
9
10           1: begin
11              PixSTART[00] = 16'b0000000000000000;
12              PixSTART[01] = 16'b0000000000000000;
13              PixSTART[02] = 16'b0000000000000000;
14              PixSTART[03] = 16'b0000001111000000;
15              PixSTART[04] = 16'b0000010000100000;
16              PixSTART[05] = 16'b0000000000100000;
17              PixSTART[06] = 16'b0000000000100000;
18              PixSTART[07] = 16'b0000000000100000;
19              PixSTART[08] = 16'b0000000111000000;
20              PixSTART[09] = 16'b0000000000100000;
21              PixSTART[10] = 16'b0000000000100000;
22              PixSTART[11] = 16'b0000010000100000;
23              PixSTART[12] = 16'b0000001111000000;
24              PixSTART[13] = 16'b0000000000000000;
25              PixSTART[14] = 16'b0000000000000000;
26              PixSTART[15] = 16'b0000000000000000;
27           end
28
29           2: begin
30              PixSTART[00] = 16'b0000000000000000;
31              PixSTART[01] = 16'b0000000000000000;
32              PixSTART[02] = 16'b0000000000000000;
33              PixSTART[03] = 16'b0000001111000000;
34              PixSTART[04] = 16'b0000010000100000;
35              PixSTART[05] = 16'b0000000000100000;
36              PixSTART[06] = 16'b0000000000100000;
37              PixSTART[07] = 16'b0000000001000000;
38              PixSTART[08] = 16'b0000000010000000;
39              PixSTART[09] = 16'b0000000100000000;
40              PixSTART[10] = 16'b0000001000000000;
41              PixSTART[11] = 16'b0000010000000000;
42              PixSTART[12] = 16'b0000011111100000;
43              PixSTART[13] = 16'b0000000000000000;
44              PixSTART[14] = 16'b0000000000000000;
45              PixSTART[15] = 16'b0000000000000000;
46           end
47
48           3: begin
49              PixSTART[00] = 16'b0000000000000000;
50              PixSTART[01] = 16'b0000000000000000;
51              PixSTART[02] = 16'b0000000000000000;
52              PixSTART[03] = 16'b0000001110000000;
53              PixSTART[04] = 16'b0000010010000000;
54              PixSTART[05] = 16'b0000000010000000;
55              PixSTART[06] = 16'b0000000010000000;
56              PixSTART[07] = 16'b0000000010000000;
57              PixSTART[08] = 16'b0000000010000000;
58              PixSTART[09] = 16'b0000000010000000;
59              PixSTART[10] = 16'b0000000010000000;
60              PixSTART[11] = 16'b0000000010000000;
61              PixSTART[12] = 16'b0000001111000000;
62              PixSTART[13] = 16'b0000000000000000;
63              PixSTART[14] = 16'b0000000000000000;
64              PixSTART[15] = 16'b0000000000000000;
65           end
66
67           default: begin
68              PixSTART[00] = 16'b0000000000000000;
69              PixSTART[01] = 16'b0000000000000000;
70              PixSTART[02] = 16'b0000000000000000;
71              PixSTART[03] = 16'b0111100011110011;
72              PixSTART[04] = 16'b1000010100001011;
73              PixSTART[05] = 16'b1000000100001011;
```

```
74                PixSTART[06] = 16'b1000000100001011;
75                PixSTART[07] = 16'b1000000100001011;
76                PixSTART[08] = 16'b1000000100001011;
77                PixSTART[09] = 16'b1001100100001011;
78                PixSTART[10] = 16'b1000100100001000;
79                PixSTART[11] = 16'b1000100100001011;
80                PixSTART[12] = 16'b0111000011110011;
81                PixSTART[13] = 16'b0000000000000000;
82                PixSTART[14] = 16'b0000000000000000;
83                PixSTART[15] = 16'b0000000000000000;
84             end
85          endcase
86       end
87    endmodule
```

```
1   module collisionCompare (RedPixels, GrnPixels, check);
2      input logic [15:0][15:0] RedPixels, GrnPixels;
3      output check;
4
5      logic [15:0][15:0] collision;
6
7      integer i,j;
8      always_comb begin
9         for (i=0; i<16; i++) begin
10           for (j=0; j<16; j++) begin
11              collision[i][j] = RedPixels[i][j] & GrnPixels[i][j];
12           end
13        end
14     end
15
16     collisionCheck collChk0 (.in(collision), .out(check));
17  endmodule
18
19  //module collisionCompare_testbench();
20  // logic [15:0][15:0] RedPixels, GrnPixels;
21  // logic check;
22  //
23  // collisionCompare dut (.RedPixels, .GrnPixels, .check);
24  //
25  // initial begin
26  //    RedPixels <= '0; GrnPixels <= '0; #10;
27  //    RedPixels[0][0] <= 1; #10;
28  //    GrnPixels[1][1] <= 1; #10;
29  //    GrnPixels[0][0] <= 1; #10;
30  // end
31  //endmodule
```

```
1   module collisionCheck (in, out);
2      input logic [15:0][15:0] in;
3      output logic out;
4
5      assign out = in[00][00]|in[00][01]|in[00][02]|in[00][03]|in[00][04]|in[00][05]|in[00][06
   ]|in[00][07]|in[00][08]|in[00][09]|in[00][10]|in[00][11]|in[00][12]|in[00][13]|in[00][14]|in
   [00][15]
6                     |in[01][00]|in[01][01]|in[01][02]|in[01][03]|in[01][04]|in[01][05]|in[01][06
   ]|in[01][07]|in[01][08]|in[01][09]|in[01][10]|in[01][11]|in[01][12]|in[01][13]|in[01][14]|in
   [01][15]
7                     |in[02][00]|in[02][01]|in[02][02]|in[02][03]|in[02][04]|in[02][05]|in[02][06
   ]|in[02][07]|in[02][08]|in[02][09]|in[02][10]|in[02][11]|in[02][12]|in[02][13]|in[02][14]|in
   [02][15]
8                     |in[03][00]|in[03][01]|in[03][02]|in[03][03]|in[03][04]|in[03][05]|in[03][06
   ]|in[03][07]|in[03][08]|in[03][09]|in[03][10]|in[03][11]|in[03][12]|in[03][13]|in[03][14]|in
   [03][15]
9                     |in[04][00]|in[04][01]|in[04][02]|in[04][03]|in[04][04]|in[04][05]|in[04][06
   ]|in[04][07]|in[04][08]|in[04][09]|in[04][10]|in[04][11]|in[04][12]|in[04][13]|in[04][14]|in
   [04][15]
10                    |in[05][00]|in[05][01]|in[05][02]|in[05][03]|in[05][04]|in[05][05]|in[05][06
   ]|in[05][07]|in[05][08]|in[05][09]|in[05][10]|in[05][11]|in[05][12]|in[05][13]|in[05][14]|in
   [05][15]
11                    |in[06][00]|in[06][01]|in[06][02]|in[06][03]|in[06][04]|in[06][05]|in[06][06
   ]|in[06][07]|in[06][08]|in[06][09]|in[06][10]|in[06][11]|in[06][12]|in[06][13]|in[06][14]|in
   [06][15]
12                    |in[07][00]|in[07][01]|in[07][02]|in[07][03]|in[07][04]|in[07][05]|in[07][06
   ]|in[07][07]|in[07][08]|in[07][09]|in[07][10]|in[07][11]|in[07][12]|in[07][13]|in[07][14]|in
   [07][15]
13                    |in[08][00]|in[08][01]|in[08][02]|in[08][03]|in[08][04]|in[08][05]|in[08][06
   ]|in[08][07]|in[08][08]|in[08][09]|in[08][10]|in[08][11]|in[08][12]|in[08][13]|in[08][14]|in
   [08][15]
14                    |in[09][00]|in[09][01]|in[09][02]|in[09][03]|in[09][04]|in[09][05]|in[09][06
   ]|in[09][07]|in[09][08]|in[09][09]|in[09][10]|in[09][11]|in[09][12]|in[09][13]|in[09][14]|in
   [09][15]
15                    |in[10][00]|in[10][01]|in[10][02]|in[10][03]|in[10][04]|in[10][05]|in[10][06
   ]|in[10][07]|in[10][08]|in[10][09]|in[10][10]|in[10][11]|in[10][12]|in[10][13]|in[10][14]|in
   [10][15]
16                    |in[11][00]|in[11][01]|in[11][02]|in[11][03]|in[11][04]|in[11][05]|in[11][06
   ]|in[11][07]|in[11][08]|in[11][09]|in[11][10]|in[11][11]|in[11][12]|in[11][13]|in[11][14]|in
   [11][15]
17                    |in[12][00]|in[12][01]|in[12][02]|in[12][03]|in[12][04]|in[12][05]|in[12][06
   ]|in[12][07]|in[12][08]|in[12][09]|in[12][10]|in[12][11]|in[12][12]|in[12][13]|in[12][14]|in
   [12][15]
18                    |in[13][00]|in[13][01]|in[13][02]|in[13][03]|in[13][04]|in[13][05]|in[13][06
   ]|in[13][07]|in[13][08]|in[13][09]|in[13][10]|in[13][11]|in[13][12]|in[13][13]|in[13][14]|in
   [13][15]
19                    |in[14][00]|in[14][01]|in[14][02]|in[14][03]|in[14][04]|in[14][05]|in[14][06
   ]|in[14][07]|in[14][08]|in[14][09]|in[14][10]|in[14][11]|in[14][12]|in[14][13]|in[14][14]|in
   [14][15]
20                    |in[15][00]|in[15][01]|in[15][02]|in[15][03]|in[15][04]|in[15][05]|in[15][06
   ]|in[15][07]|in[15][08]|in[15][09]|in[15][10]|in[15][11]|in[15][12]|in[15][13]|in[15][14]|in
   [15][15];
21  endmodule
22
23  //module collisionCheck_testbench();
24  // logic out;
25  // logic [15:0][15:0] in;
26  //
27  // collisionCheck dut (.in, .out);
28  //
29  // initial begin
30  //    in <= '0; #10;
31  //    in[5][14] <= 1; #10;
32  //    in <= '0; #10;
33  //    in[0][0] <= 1; #10;
34  // end
35  //endmodule
```

```
1   // Counter select used for selecting different speed counters based on input
2   module countersel (CLK, RST, MSB, whichCounter);
3      input logic CLK, RST;
4      input logic [2:0] whichCounter;
5      output logic [3:0] MSB;
6
7      logic [31:0] MSBs;
8      logic [7:0] EnableCounts;
9
10     // Increasingly slower counter options
11     rowcounter0 counter0 (.CLK, .RST, .MSB0(MSBs[3:0]), .EnableCount0(EnableCounts[0]));
12     rowcounter1 counter1 (.CLK, .RST, .MSB1(MSBs[7:4]), .EnableCount1(EnableCounts[1]));
13     rowcounter2 counter2 (.CLK, .RST, .MSB2(MSBs[11:8]), .EnableCount2(EnableCounts[2]));
14     rowcounter3 counter3 (.CLK, .RST, .MSB3(MSBs[15:12]), .EnableCount3(EnableCounts[3]));
15     rowcounter4 counter4 (.CLK, .RST, .MSB4(MSBs[19:16]), .EnableCount4(EnableCounts[4]));
16     rowcounter5 counter5 (.CLK, .RST, .MSB5(MSBs[23:20]), .EnableCount5(EnableCounts[5]));
17     rowcounter6 counter6 (.CLK, .RST, .MSB6(MSBs[27:24]), .EnableCount6(EnableCounts[6]));
18     rowcounter7 counter7 (.CLK, .RST, .MSB7(MSBs[31:28]), .EnableCount7(EnableCounts[7]));
19
20     logic [3:0] whichMSB;
21
22     always_comb begin
23        if          (whichCounter == 3'b000) begin
24           whichMSB = MSBs[3:0];
25           EnableCounts = 8'b00000001;
26        end else if (whichCounter == 3'b001) begin
27           whichMSB = MSBs[7:4];
28           EnableCounts = 8'b00000010;
29        end else if (whichCounter == 3'b010) begin
30           whichMSB = MSBs[11:8];
31           EnableCounts = 8'b00000100;
32        end else if (whichCounter == 3'b011) begin
33           whichMSB = MSBs[15:12];
34           EnableCounts = 8'b00001000;
35        end else if (whichCounter == 3'b100) begin
36           whichMSB = MSBs[19:16];
37           EnableCounts = 8'b00010000;
38        end else if (whichCounter == 3'b101) begin
39           whichMSB = MSBs[23:20];
40           EnableCounts = 8'b00100000;
41        end else if (whichCounter == 3'b110) begin
42           whichMSB = MSBs[27:24];
43           EnableCounts = 8'b01000000;
44        end else if (whichCounter == 3'b111) begin
45           whichMSB = MSBs[31:28];
46           EnableCounts = 8'b10000000;
47        end else begin
48           whichMSB = MSBs[3:0];
49           EnableCounts = 8'b00000000;
50        end
51     end
52
53     assign MSB = whichMSB;
54  endmodule
```

```
1   // James Lee
2   // University of Washington
3   // EE 271 Lab 8
4   // Fall 2021
5
6   // NOTE: This project has modified versions of the EE 271 Winter 2021 peripheral tutorial
    source files
7
8   // References (Last accessed 12/7/2021)
9   //    1. Course Page: https://class.ece.uw.edu/271/hauck2/de1/index.html
10  //    2. Starter Files: https://class.ece.uw.edu/271/hauck2/de1/LEDboard/led_driver.zip
11
12
13  // Counters take incoming clock and output repeated counts based on width
14  // MSB = 3 most significant bits
15  module rowcounter0 #(parameter FREQDIV = 27) (CLK, RST, MSB0, EnableCount0);
16     input logic CLK, RST, EnableCount0;
17     output logic [3:0]   MSB0;
18
19     reg [(FREQDIV + 3):0] Counter;
20     assign MSB0 = Counter[(FREQDIV + 3):FREQDIV];
21
22     logic reset;
23     assign reset = RST | ~EnableCount0;
24
25     always_ff @(posedge CLK)
26     begin
27        if(reset) Counter <= 'b0;
28        else Counter <= Counter + 1'b1;
29     end
30  endmodule
31
32  module rowcounter1 #(parameter FREQDIV = 26) (CLK, RST, MSB1, EnableCount1);
33     input logic CLK, RST, EnableCount1;
34     output logic [3:0]   MSB1;
35
36     reg [(FREQDIV + 3):0] Counter;
37     assign MSB1 = Counter[(FREQDIV + 3):FREQDIV];
38
39     logic reset;
40     assign reset = RST | ~EnableCount1;
41
42     always_ff @(posedge CLK)
43     begin
44        if(reset) Counter <= 'b0;
45        else Counter <= Counter + 1'b1;
46     end
47  endmodule
48
49  module rowcounter2 #(parameter FREQDIV = 25) (CLK, RST, MSB2, EnableCount2);
50     input logic CLK, RST, EnableCount2;
51     output logic [3:0]   MSB2;
52
53     reg [(FREQDIV + 3):0] Counter;
54     assign MSB2 = Counter[(FREQDIV + 3):FREQDIV];
55
56     logic reset;
57     assign reset = RST | ~EnableCount2;
58
59     always_ff @(posedge CLK)
60     begin
61        if(reset) Counter <= 'b0;
62        else Counter <= Counter + 1'b1;
63     end
64  endmodule
65
66  module rowcounter3 #(parameter FREQDIV = 24) (CLK, RST, MSB3, EnableCount3);
67     input logic CLK, RST, EnableCount3;
68     output logic [3:0]   MSB3;
69
70     reg [(FREQDIV + 3):0] Counter;
71     assign MSB3 = Counter[(FREQDIV + 3):FREQDIV];
72
```

```
73      logic reset;
74      assign reset = RST | ~EnableCount3;
75
76      always_ff @(posedge CLK)
77      begin
78         if(reset) Counter <= 'b0;
79         else Counter <= Counter + 1'b1;
80      end
81   endmodule
82
83   module rowcounter4 #(parameter FREQDIV = 23) (CLK, RST, MSB4, EnableCount4);
84      input logic CLK, RST, EnableCount4;
85      output logic [3:0]   MSB4;
86
87      reg [(FREQDIV + 3):0] Counter;
88      assign MSB4 = Counter[(FREQDIV + 3):FREQDIV];
89
90      logic reset;
91      assign reset = RST | ~EnableCount4;
92
93      always_ff @(posedge CLK)
94      begin
95         if(reset) Counter <= 'b0;
96         else Counter <= Counter + 1'b1;
97      end
98   endmodule
99
100  module rowcounter5 #(parameter FREQDIV = 22) (CLK, RST, MSB5, EnableCount5);
101     input logic CLK, RST, EnableCount5;
102     output logic [3:0]   MSB5;
103
104     reg [(FREQDIV + 3):0] Counter;
105     assign MSB5 = Counter[(FREQDIV + 3):FREQDIV];
106
107     logic reset;
108     assign reset = RST | ~EnableCount5;
109
110     always_ff @(posedge CLK)
111     begin
112        if(reset) Counter <= 'b0;
113        else Counter <= Counter + 1'b1;
114     end
115  endmodule
116
117  module rowcounter6 #(parameter FREQDIV = 21) (CLK, RST, MSB6, EnableCount6);
118     input logic CLK, RST, EnableCount6;
119     output logic [3:0]   MSB6;
120
121     reg [(FREQDIV + 3):0] Counter;
122     assign MSB6 = Counter[(FREQDIV + 3):FREQDIV];
123
124     logic reset;
125     assign reset = RST | ~EnableCount6;
126
127     always_ff @(posedge CLK)
128     begin
129        if(reset) Counter <= 'b0;
130        else Counter <= Counter + 1'b1;
131     end
132  endmodule
133
134  module rowcounter7 #(parameter FREQDIV = 20) (CLK, RST, MSB7, EnableCount7);
135     input logic CLK, RST, EnableCount7;
136     output logic [3:0]   MSB7;
137
138     reg [(FREQDIV + 3):0] Counter;
139     assign MSB7 = Counter[(FREQDIV + 3):FREQDIV];
140
141     logic reset;
142     assign reset = RST | ~EnableCount7;
143
144     always_ff @(posedge CLK)
145     begin
```

```
146         if(reset) Counter <= 'b0;
147         else Counter <= Counter + 1'b1;
148       end
149   endmodule
```

```
module asteroidLocation (CLK, RST, ROW, GrnPixels, RUNen);
   input logic CLK, RST, RUNen;
   input logic [3:0] ROW;
   output logic [15:0][15:0] GrnPixels;

   logic [15:0][15:0] ps,ns;

   // LSFR chooses pseudorandom column
   logic [9:0] COLchoice;
   LSFR10b randColGen0 (.out(COLchoice), .RST, .CLK);

   // choice toggle latches choice at beginning of ROW count sequence
   logic [3:0] choice, choice_in;
   assign choice_in = COLchoice[3:0];
   choiceToggle toggle0 (.choice, .choice_in, .ROW, .RST, .CLK);

   // maps choice value to column pixel for given row
   logic [15:0] COL;
   columnChoice colChoice0 (.choice, .COL);

   always_comb begin
      case (ROW)
      0:  begin ns = '0; ns[00] = COL; end
      1:  begin ns = '0; ns[01] = COL; end
      2:  begin ns = '0; ns[02] = COL; end
      3:  begin ns = '0; ns[03] = COL; end
      4:  begin ns = '0; ns[04] = COL; end
      5:  begin ns = '0; ns[05] = COL; end
      6:  begin ns = '0; ns[06] = COL; end
      7:  begin ns = '0; ns[07] = COL; end
      8:  begin ns = '0; ns[08] = COL; end
      9:  begin ns = '0; ns[09] = COL; end
      10: begin ns = '0; ns[10] = COL; end
      11: begin ns = '0; ns[11] = COL; end
      12: begin ns = '0; ns[12] = COL; end
      13: begin ns = '0; ns[13] = COL; end
      14: begin ns = '0; ns[14] = COL; end
      15: begin ns = '0; ns[15] = COL; end
      default: begin ns = '0; end// all pixels off by default
      endcase
   end

   always_ff @(posedge CLK) begin
      if (RST) begin
         ps <= '0;
      end else begin
         ps <= ns;
      end
   end

   assign GrnPixels = ps;

endmodule

//module asteroidLocation_testbench();
// logic CLK, RST;
// logic [3:0] ROW;
// logic [15:0][15:0] GrnPixels;
//
// // Set up a simulated  clock
// parameter CLOCK_PERIOD = 100;
//
// initial begin
//    CLK <= 0;
//    forever # (CLOCK_PERIOD/2) CLK <= ~CLK;  // Forever toggle clock
// end
//
// asteroidLocation dut (.CLK, .RST, .ROW, .GrnPixels);
// integer i,j;
// initial begin
//    RST <=1; ROW <= '0; @(posedge CLK);
//    RST <=0; @(posedge CLK);
//    for (j=0; j<3; j++) begin
```

```
74  //        for (i=0; i<16; i++) begin
75  //           ROW = i; @(posedge CLK);
76  //        end
77  //     end
78  //     $stop;
79  // end
80  //endmodule
```

```
1   // Choice Toggle updates column choice and beginning of count sequence
2   module choiceToggle (choice, choice_in, ROW, RST, CLK);
3      input logic RST, CLK;
4      input logic [3:0] choice_in, ROW;
5      output logic [3:0] choice;
6
7      logic d;
8
9      always_comb begin
10        case (ROW)
11           0: d = 1; // set toggle only at beggining of clock cycle
12           default d = 0;
13        endcase
14     end
15
16     // use offset of flip flops to output one toggle pulse
17     // CLK cycle is much faster than ROW count sequence
18     logic toggle, q0, q1, q2;
19     D_FF Dtoggle0 (.q(q0), .d, .RST, .CLK);
20     D_FF Dtoggle1 (.q(q1), .d(q0), .RST, .CLK);
21     D_FF Dtoggle2 (.q(q2), .d(q1), .RST, .CLK);
22
23     assign toggle = q1 & ~q2; // output only one cycle when q1 and q2 are offset
24
25     always_ff @(posedge CLK) begin
26        if (RST) begin // reset choice to 0
27           choice <= '0;
28        end else if (toggle) begin // set choice on toggle
29           choice <= choice_in;
30        end else begin // keep choice the same
31           choice <= choice;
32        end
33     end
34
35  endmodule
```

```
1   // Column Choice maps a 4-bit choice to a 16-bit pixel row
2   module columnChoice (choice, COL);
3      input logic [3:0] choice;
4      output logic [15:0] COL;
5
6
7      always_comb begin
8         case(choice)
9            0:  COL = 16'b0000000000000001;
10           1:  COL = 16'b0000000000000010;
11           2:  COL = 16'b0000000000000100;
12           3:  COL = 16'b0000000000001000;
13           4:  COL = 16'b0000000000010000;
14           5:  COL = 16'b0000000000100000;
15           6:  COL = 16'b0000000001000000;
16           7:  COL = 16'b0000000010000000;
17           8:  COL = 16'b0000000100000000;
18           9:  COL = 16'b0000001000000000;
19           10: COL = 16'b0000010000000000;
20           11: COL = 16'b0000100000000000;
21           12: COL = 16'b0001000000000000;
22           13: COL = 16'b0010000000000000;
23           14: COL = 16'b0100000000000000;
24           15: COL = 16'b1000000000000000;
25           default: COL = 0;
26        endcase
27     end
28  endmodule
29
30  //module columnChoice_testbench();
31  // logic [3:0] choice;
32  // integer COL;
33  //
34  // columnChoice dut (.choice, .COL);
35  //
36  // initial begin
37  //    choice <= 0; #10;
38  //    choice <= 4; #10;
39  //    choice <= 10; #10;
40  // end
41  //endmodule
```

```
// Ship Location maps user inputs to ship pixel location
module shipLocation (RedPixels, L, R, RST, CLK);
   input logic                RST, CLK, L, R; //clock, reset, left, right
   output logic [15:0][15:0]  RedPixels; // 16x16 array of red LEDs

   logic [3:0] ps, ns; // position present state, next state

   // 13 possible ship positions
   // 0 = far left position
   // 12 = far right position
   always_comb begin
      // Conditions
      // (R && L):   simultaneous key press -> no change
      // (R):        R input -> move right 1
      // (L):        L input -> move left 1
      //             No input -> no change
      case (ps)
         4'b0000: begin // Position 0
            if (R && L) ns = 4'b0000;
            else if (L) ns = 4'b0000; // no change at edge
            else if (R) ns = 4'b0001;
            else        ns = 4'b0000;
         end
         4'b0001: begin // Position 1
            if (R && L) ns = 4'b0001;
            else if (L) ns = 4'b0000;
            else if (R) ns = 4'b0010;
            else        ns = 4'b0001;
         end
         4'b0010: begin // Position 2
            if (R && L) ns = 4'b0010;
            else if (L) ns = 4'b0001;
            else if (R) ns = 4'b0011;
            else        ns = 4'b0010;
         end
         4'b0011: begin // Position 3
            if (R && L) ns = 4'b0011;
            else if (L) ns = 4'b0010;
            else if (R) ns = 4'b0100;
            else        ns = 4'b0011;
         end
         4'b0100: begin // Position 4
            if (R && L) ns = 4'b0100;
            else if (L) ns = 4'b0011;
            else if (R) ns = 4'b0101;
            else        ns = 4'b0100;
         end
         4'b0101: begin // Position 5
            if (R && L) ns = 4'b0101;
            else if (L) ns = 4'b0100;
            else if (R) ns = 4'b0110;
            else        ns = 4'b0101;
         end
         4'b0111: begin // Position 7
            if (R && L) ns = 4'b0111;
            else if (L) ns = 4'b0110;
            else if (R) ns = 4'b1000;
            else        ns = 4'b0111;
         end
         4'b1000: begin // Position 8
            if (R && L) ns = 4'b1000;
            else if (L) ns = 4'b0111;
            else if (R) ns = 4'b1001;
            else        ns = 4'b1000;
         end
         4'b1001: begin // Position 9
            if (R && L) ns = 4'b1001;
            else if (L) ns = 4'b1000;
            else if (R) ns = 4'b1010;
            else        ns = 4'b1001;
         end
         4'b1010: begin // Position 10
            if (R && L) ns = 4'b1010;
```

```
74                else if (L) ns = 4'b1001;
75                else if (R) ns = 4'b1011;
76                else        ns = 4'b1010;
77             end
78             4'b1011: begin // Position 11
79                if (R && L) ns = 4'b1011;
80                else if (L) ns = 4'b1010;
81                else if (R) ns = 4'b1100;
82                else        ns = 4'b1011;
83             end
84             4'b1100: begin // Position 12
85                if (R && L) ns = 4'b1100;
86                else if (L) ns = 4'b1011;
87                else if (R) ns = 4'b1100; // no change at edge
88                else        ns = 4'b1100;
89             end
90             default: begin // Position 6 (center)
91                if (R && L) ns = 4'b0110;
92                else if (L) ns = 4'b0101;
93                else if (R) ns = 4'b0111;
94                else        ns = 4'b0110;
95             end
96          endcase
97       end
98
99
100      // DFFs
101      always_ff @(posedge CLK) begin
102         if       (RST) ps <= 4'b0110;
103         else           ps <= ns;
104      end
105
106
107      // instantiate position to pixel map
108      logic [3:0] position;
109      assign position = ps;
110      shipPositions shipPos0 (.position, .RedPixels);
111   endmodule
```

```
1   // James Lee
2   // University of Washington
3   // EE 271 Lab 8
4   // Fall 2021
5
6   // NOTE: This project has modified versions of the EE 271 Winter 2021 peripheral tutorial
    source files
7
8   // References (Last accessed 12/7/2021)
9   //    1. Course Page: https://class.ece.uw.edu/271/hauck2/de1/index.html
10  //    2. Starter Files: https://class.ece.uw.edu/271/hauck2/de1/LEDboard/led_driver.zip
11
12  module shipPositions (position, RedPixels);
13     input logic [3:0] position;
14     output logic [15:0][15:0] RedPixels;
15
16     logic [15:0][15:0] ShipPixels;
17
18     // 13 possible ship positions
19     // 0 = far left position
20     // 12 = far right position
21     always_comb begin
22        case (position)
23           4'b0000: begin // Position 0
24              ShipPixels[00] = 16'b0000000000000000;
25              ShipPixels[01] = 16'b0000000000000000;
26              ShipPixels[02] = 16'b0000000000000000;
27              ShipPixels[03] = 16'b0000000000000000;
28              ShipPixels[04] = 16'b0000000000000000;
29              ShipPixels[05] = 16'b0000000000000000;
30              ShipPixels[06] = 16'b0000000000000000;
31              ShipPixels[07] = 16'b0000000000000000;
32              ShipPixels[08] = 16'b0000000000000000;
33              ShipPixels[09] = 16'b0000000000000000;
34              ShipPixels[10] = 16'b0000000000000000;
35              ShipPixels[11] = 16'b0000000000000000;
36              ShipPixels[12] = 16'b0110000000000000;
37              ShipPixels[13] = 16'b0110000000000000;
38              ShipPixels[14] = 16'b1111000000000000;
39              ShipPixels[15] = 16'b1001000000000000;
40           end
41
42           4'b0001: begin // Position 1
43              ShipPixels[00] = 16'b0000000000000000;
44              ShipPixels[01] = 16'b0000000000000000;
45              ShipPixels[02] = 16'b0000000000000000;
46              ShipPixels[03] = 16'b0000000000000000;
47              ShipPixels[04] = 16'b0000000000000000;
48              ShipPixels[05] = 16'b0000000000000000;
49              ShipPixels[06] = 16'b0000000000000000;
50              ShipPixels[07] = 16'b0000000000000000;
51              ShipPixels[08] = 16'b0000000000000000;
52              ShipPixels[09] = 16'b0000000000000000;
53              ShipPixels[10] = 16'b0000000000000000;
54              ShipPixels[11] = 16'b0000000000000000;
55              ShipPixels[12] = 16'b0011000000000000;
56              ShipPixels[13] = 16'b0011000000000000;
57              ShipPixels[14] = 16'b0111100000000000;
58              ShipPixels[15] = 16'b0100100000000000;
59           end
60
61           4'b0010: begin // Position 2
62              ShipPixels[00] = 16'b0000000000000000;
63              ShipPixels[01] = 16'b0000000000000000;
64              ShipPixels[02] = 16'b0000000000000000;
65              ShipPixels[03] = 16'b0000000000000000;
66              ShipPixels[04] = 16'b0000000000000000;
67              ShipPixels[05] = 16'b0000000000000000;
68              ShipPixels[06] = 16'b0000000000000000;
69              ShipPixels[07] = 16'b0000000000000000;
70              ShipPixels[08] = 16'b0000000000000000;
71              ShipPixels[09] = 16'b0000000000000000;
72              ShipPixels[10] = 16'b0000000000000000;
```

```
73                ShipPixels[11] = 16'b0000000000000000;
74                ShipPixels[12] = 16'b0001100000000000;
75                ShipPixels[13] = 16'b0001100000000000;
76                ShipPixels[14] = 16'b0011110000000000;
77                ShipPixels[15] = 16'b0010010000000000;
78            end
79
80            4'b0011: begin // Position 3
81                ShipPixels[00] = 16'b0000000000000000;
82                ShipPixels[01] = 16'b0000000000000000;
83                ShipPixels[02] = 16'b0000000000000000;
84                ShipPixels[03] = 16'b0000000000000000;
85                ShipPixels[04] = 16'b0000000000000000;
86                ShipPixels[05] = 16'b0000000000000000;
87                ShipPixels[06] = 16'b0000000000000000;
88                ShipPixels[07] = 16'b0000000000000000;
89                ShipPixels[08] = 16'b0000000000000000;
90                ShipPixels[09] = 16'b0000000000000000;
91                ShipPixels[10] = 16'b0000000000000000;
92                ShipPixels[11] = 16'b0000000000000000;
93                ShipPixels[12] = 16'b0000110000000000;
94                ShipPixels[13] = 16'b0000110000000000;
95                ShipPixels[14] = 16'b0001111000000000;
96                ShipPixels[15] = 16'b0001001000000000;
97            end
98
99            4'b0100: begin // Position 4
100               ShipPixels[00] = 16'b0000000000000000;
101               ShipPixels[01] = 16'b0000000000000000;
102               ShipPixels[02] = 16'b0000000000000000;
103               ShipPixels[03] = 16'b0000000000000000;
104               ShipPixels[04] = 16'b0000000000000000;
105               ShipPixels[05] = 16'b0000000000000000;
106               ShipPixels[06] = 16'b0000000000000000;
107               ShipPixels[07] = 16'b0000000000000000;
108               ShipPixels[08] = 16'b0000000000000000;
109               ShipPixels[09] = 16'b0000000000000000;
110               ShipPixels[10] = 16'b0000000000000000;
111               ShipPixels[11] = 16'b0000000000000000;
112               ShipPixels[12] = 16'b0000011000000000;
113               ShipPixels[13] = 16'b0000011000000000;
114               ShipPixels[14] = 16'b0000111100000000;
115               ShipPixels[15] = 16'b0000100100000000;
116           end
117
118           4'b0101: begin // Position 5
119               ShipPixels[00] = 16'b0000000000000000;
120               ShipPixels[01] = 16'b0000000000000000;
121               ShipPixels[02] = 16'b0000000000000000;
122               ShipPixels[03] = 16'b0000000000000000;
123               ShipPixels[04] = 16'b0000000000000000;
124               ShipPixels[05] = 16'b0000000000000000;
125               ShipPixels[06] = 16'b0000000000000000;
126               ShipPixels[07] = 16'b0000000000000000;
127               ShipPixels[08] = 16'b0000000000000000;
128               ShipPixels[09] = 16'b0000000000000000;
129               ShipPixels[10] = 16'b0000000000000000;
130               ShipPixels[11] = 16'b0000000000000000;
131               ShipPixels[12] = 16'b0000001100000000;
132               ShipPixels[13] = 16'b0000001100000000;
133               ShipPixels[14] = 16'b0000011110000000;
134               ShipPixels[15] = 16'b0000010010000000;
135           end
136
137           4'b0111: begin // Position 7
138               ShipPixels[00] = 16'b0000000000000000;
139               ShipPixels[01] = 16'b0000000000000000;
140               ShipPixels[02] = 16'b0000000000000000;
141               ShipPixels[03] = 16'b0000000000000000;
142               ShipPixels[04] = 16'b0000000000000000;
143               ShipPixels[05] = 16'b0000000000000000;
144               ShipPixels[06] = 16'b0000000000000000;
145               ShipPixels[07] = 16'b0000000000000000;
```

```
                ShipPixels[08] = 16'b0000000000000000;
                ShipPixels[09] = 16'b0000000000000000;
                ShipPixels[10] = 16'b0000000000000000;
                ShipPixels[11] = 16'b0000000000000000;
                ShipPixels[12] = 16'b0000000011000000;
                ShipPixels[13] = 16'b0000000011000000;
                ShipPixels[14] = 16'b0000000111100000;
                ShipPixels[15] = 16'b0000000100100000;
            end

            4'b1000: begin // Position 8
                ShipPixels[00] = 16'b0000000000000000;
                ShipPixels[01] = 16'b0000000000000000;
                ShipPixels[02] = 16'b0000000000000000;
                ShipPixels[03] = 16'b0000000000000000;
                ShipPixels[04] = 16'b0000000000000000;
                ShipPixels[05] = 16'b0000000000000000;
                ShipPixels[06] = 16'b0000000000000000;
                ShipPixels[07] = 16'b0000000000000000;
                ShipPixels[08] = 16'b0000000000000000;
                ShipPixels[09] = 16'b0000000000000000;
                ShipPixels[10] = 16'b0000000000000000;
                ShipPixels[11] = 16'b0000000000000000;
                ShipPixels[12] = 16'b0000000001100000;
                ShipPixels[13] = 16'b0000000001100000;
                ShipPixels[14] = 16'b0000000011110000;
                ShipPixels[15] = 16'b0000000010010000;
            end

            4'b1001: begin // Position 9
                ShipPixels[00] = 16'b0000000000000000;
                ShipPixels[01] = 16'b0000000000000000;
                ShipPixels[02] = 16'b0000000000000000;
                ShipPixels[03] = 16'b0000000000000000;
                ShipPixels[04] = 16'b0000000000000000;
                ShipPixels[05] = 16'b0000000000000000;
                ShipPixels[06] = 16'b0000000000000000;
                ShipPixels[07] = 16'b0000000000000000;
                ShipPixels[08] = 16'b0000000000000000;
                ShipPixels[09] = 16'b0000000000000000;
                ShipPixels[10] = 16'b0000000000000000;
                ShipPixels[11] = 16'b0000000000000000;
                ShipPixels[12] = 16'b0000000000110000;
                ShipPixels[13] = 16'b0000000000110000;
                ShipPixels[14] = 16'b0000000001111000;
                ShipPixels[15] = 16'b0000000001001000;
            end

            4'b1010: begin // Position 10
                ShipPixels[00] = 16'b0000000000000000;
                ShipPixels[01] = 16'b0000000000000000;
                ShipPixels[02] = 16'b0000000000000000;
                ShipPixels[03] = 16'b0000000000000000;
                ShipPixels[04] = 16'b0000000000000000;
                ShipPixels[05] = 16'b0000000000000000;
                ShipPixels[06] = 16'b0000000000000000;
                ShipPixels[07] = 16'b0000000000000000;
                ShipPixels[08] = 16'b0000000000000000;
                ShipPixels[09] = 16'b0000000000000000;
                ShipPixels[10] = 16'b0000000000000000;
                ShipPixels[11] = 16'b0000000000000000;
                ShipPixels[12] = 16'b0000000000011000;
                ShipPixels[13] = 16'b0000000000011000;
                ShipPixels[14] = 16'b0000000000111100;
                ShipPixels[15] = 16'b0000000000100100;
            end

            4'b1011: begin // Position 11
                ShipPixels[00] = 16'b0000000000000000;
                ShipPixels[01] = 16'b0000000000000000;
                ShipPixels[02] = 16'b0000000000000000;
                ShipPixels[03] = 16'b0000000000000000;
                ShipPixels[04] = 16'b0000000000000000;
```

```
219                ShipPixels[05] = 16'b0000000000000000;
220                ShipPixels[06] = 16'b0000000000000000;
221                ShipPixels[07] = 16'b0000000000000000;
222                ShipPixels[08] = 16'b0000000000000000;
223                ShipPixels[09] = 16'b0000000000000000;
224                ShipPixels[10] = 16'b0000000000000000;
225                ShipPixels[11] = 16'b0000000000000000;
226                ShipPixels[12] = 16'b0000000000001100;
227                ShipPixels[13] = 16'b0000000000001100;
228                ShipPixels[14] = 16'b0000000000011110;
229                ShipPixels[15] = 16'b0000000000010010;
230             end
231
232             4'b1100: begin // Position 12
233                ShipPixels[00] = 16'b0000000000000000;
234                ShipPixels[01] = 16'b0000000000000000;
235                ShipPixels[02] = 16'b0000000000000000;
236                ShipPixels[03] = 16'b0000000000000000;
237                ShipPixels[04] = 16'b0000000000000000;
238                ShipPixels[05] = 16'b0000000000000000;
239                ShipPixels[06] = 16'b0000000000000000;
240                ShipPixels[07] = 16'b0000000000000000;
241                ShipPixels[08] = 16'b0000000000000000;
242                ShipPixels[09] = 16'b0000000000000000;
243                ShipPixels[10] = 16'b0000000000000000;
244                ShipPixels[11] = 16'b0000000000000000;
245                ShipPixels[12] = 16'b0000000000000110;
246                ShipPixels[13] = 16'b0000000000000110;
247                ShipPixels[14] = 16'b0000000000001111;
248                ShipPixels[15] = 16'b0000000000001001;
249             end
250
251             default: begin // position 6 (middle) is default
252                ShipPixels[00] = 16'b0000000000000000;
253                ShipPixels[01] = 16'b0000000000000000;
254                ShipPixels[02] = 16'b0000000000000000;
255                ShipPixels[03] = 16'b0000000000000000;
256                ShipPixels[04] = 16'b0000000000000000;
257                ShipPixels[05] = 16'b0000000000000000;
258                ShipPixels[06] = 16'b0000000000000000;
259                ShipPixels[07] = 16'b0000000000000000;
260                ShipPixels[08] = 16'b0000000000000000;
261                ShipPixels[09] = 16'b0000000000000000;
262                ShipPixels[10] = 16'b0000000000000000;
263                ShipPixels[11] = 16'b0000000000000000;
264                ShipPixels[12] = 16'b0000000110000000;
265                ShipPixels[13] = 16'b0000000110000000;
266                ShipPixels[14] = 16'b0000001111000000;
267                ShipPixels[15] = 16'b0000001001000000;
268             end
269          endcase
270       end
271
272       assign RedPixels = ShipPixels;
273    endmodule
274
275    //module shipPositions_testbench();
276    // logic [3:0] position;
277    // logic [15:0][15:0] RedPixels;
278    //
279    // shipPositions dut (.position, .RedPixels);
280    //
281    // integer i;
282    // initial begin
283    //    for (i=0; i<16; i++) begin
284    //       position = i; #10;
285    //    end
286    // end
287    //endmodule
```

```
module endExplode (position, PixEND);
   input logic [3:0] position;
   output logic [15:0][15:0] PixEND;


   always_comb begin
      case (position)
         0: PixEND = '0;

         1: begin // Explode Position 0
            PixEND[00] = 16'b0000000000000000;
            PixEND[01] = 16'b0000000000000000;
            PixEND[02] = 16'b0000000000000000;
            PixEND[03] = 16'b0000000000000000;
            PixEND[04] = 16'b0000000000000000;
            PixEND[05] = 16'b0000000000000000;
            PixEND[06] = 16'b0000000000000000;
            PixEND[07] = 16'b0000001100000000;
            PixEND[08] = 16'b0000001100000000;
            PixEND[09] = 16'b0000000000000000;
            PixEND[10] = 16'b0000000000000000;
            PixEND[11] = 16'b0000000000000000;
            PixEND[12] = 16'b0000000000000000;
            PixEND[13] = 16'b0000000000000000;
            PixEND[14] = 16'b0000000000000000;
            PixEND[15] = 16'b0000000000000000;
         end

         2: begin // Explode Position 1
            PixEND[00] = 16'b0000000000000000;
            PixEND[01] = 16'b0000000000000000;
            PixEND[02] = 16'b0000000000000000;
            PixEND[03] = 16'b0000000000000000;
            PixEND[04] = 16'b0000000000000000;
            PixEND[05] = 16'b0000100001000000;
            PixEND[06] = 16'b0000010010000000;
            PixEND[07] = 16'b0000001100000000;
            PixEND[08] = 16'b0000001100000000;
            PixEND[09] = 16'b0000010010000000;
            PixEND[10] = 16'b0000100001000000;
            PixEND[11] = 16'b0000000000000000;
            PixEND[12] = 16'b0000000000000000;
            PixEND[13] = 16'b0000000000000000;
            PixEND[14] = 16'b0000000000000000;
            PixEND[15] = 16'b0000000000000000;
         end

         3: begin // Explode Position 2
            PixEND[00] = 16'b0000000000000000;
            PixEND[01] = 16'b0000000000000000;
            PixEND[02] = 16'b0000000000000000;
            PixEND[03] = 16'b0010001100010000;
            PixEND[04] = 16'b0001010010100000;
            PixEND[05] = 16'b0000100001000000;
            PixEND[06] = 16'b0000010010000000;
            PixEND[07] = 16'b0000001100000000;
            PixEND[08] = 16'b0000001100000000;
            PixEND[09] = 16'b0000010010000000;
            PixEND[10] = 16'b0000100001000000;
            PixEND[11] = 16'b0001010010100000;
            PixEND[12] = 16'b0010001100010000;
            PixEND[13] = 16'b0000000000000000;
            PixEND[14] = 16'b0000000000000000;
            PixEND[15] = 16'b0000000000000000;
         end

         4: begin // Explode Position 3
            PixEND[00] = 16'b0001000010000000;
            PixEND[01] = 16'b0010000100000000;
            PixEND[02] = 16'b0100001000000000;
            PixEND[03] = 16'b0010001100010000;
            PixEND[04] = 16'b0001010010100000;
            PixEND[05] = 16'b0000100001010000;
```

```
                PixEND[06] = 16'b0000010010001000;
                PixEND[07] = 16'b0000001100000100;
                PixEND[08] = 16'b0000001100000000;
                PixEND[09] = 16'b0000010010000000;
                PixEND[10] = 16'b0000100001000000;
                PixEND[11] = 16'b0001010010100000;
                PixEND[12] = 16'b0010001100010000;
                PixEND[13] = 16'b0010000010000000;
                PixEND[14] = 16'b0001000001000000;
                PixEND[15] = 16'b0010000000000000;
            end

            5: begin // ALL RED
                PixEND = '1;
            end

            6: begin // END
                PixEND[00] = 16'b0000000000000000;
                PixEND[01] = 16'b0000000000000000;
                PixEND[02] = 16'b0000000000000000;
                PixEND[03] = 16'b0000000000000000;
                PixEND[04] = 16'b0000000000000000;
                PixEND[05] = 16'b0111010001011100;
                PixEND[06] = 16'b0100011001010010;
                PixEND[07] = 16'b0111010101010010;
                PixEND[08] = 16'b0100010011010010;
                PixEND[09] = 16'b0111010001011100;
                PixEND[10] = 16'b0000000000000000;
                PixEND[11] = 16'b0000000000000000;
                PixEND[12] = 16'b0000000000000000;
                PixEND[13] = 16'b0000000000000000;
                PixEND[14] = 16'b0000000000000000;
                PixEND[15] = 16'b0000000000000000;
            end

            7: begin // END
                PixEND[00] = 16'b0000000000000000;
                PixEND[01] = 16'b0000000000000000;
                PixEND[02] = 16'b0000000000000000;
                PixEND[03] = 16'b0000000000000000;
                PixEND[04] = 16'b0000000000000000;
                PixEND[05] = 16'b0111010001011100;
                PixEND[06] = 16'b0100011001010010;
                PixEND[07] = 16'b0111010101010010;
                PixEND[08] = 16'b0100010011010010;
                PixEND[09] = 16'b0111010001011100;
                PixEND[10] = 16'b0000000000000000;
                PixEND[11] = 16'b0000000000000000;
                PixEND[12] = 16'b0000000000000000;
                PixEND[13] = 16'b0000000000000000;
                PixEND[14] = 16'b0000000000000000;
                PixEND[15] = 16'b0000000000000000;
            end

            8: begin // END
                PixEND[00] = 16'b0000000000000000;
                PixEND[01] = 16'b0000000000000000;
                PixEND[02] = 16'b0000000000000000;
                PixEND[03] = 16'b0000000000000000;
                PixEND[04] = 16'b0000000000000000;
                PixEND[05] = 16'b0111010001011100;
                PixEND[06] = 16'b0100011001010010;
                PixEND[07] = 16'b0111010101010010;
                PixEND[08] = 16'b0100010011010010;
                PixEND[09] = 16'b0111010001011100;
                PixEND[10] = 16'b0000000000000000;
                PixEND[11] = 16'b0000000000000000;
                PixEND[12] = 16'b0000000000000000;
                PixEND[13] = 16'b0000000000000000;
                PixEND[14] = 16'b0000000000000000;
                PixEND[15] = 16'b0000000000000000;
            end

```

```
147             9: begin // END
148                PixEND[00] = 16'b0000000000000000;
149                PixEND[01] = 16'b0000000000000000;
150                PixEND[02] = 16'b0000000000000000;
151                PixEND[03] = 16'b0000000000000000;
152                PixEND[04] = 16'b0000000000000000;
153                PixEND[05] = 16'b0111010001011100;
154                PixEND[06] = 16'b0100011001010010;
155                PixEND[07] = 16'b0111010101010010;
156                PixEND[08] = 16'b0100010011010010;
157                PixEND[09] = 16'b0111010001011100;
158                PixEND[10] = 16'b0000000000000000;
159                PixEND[11] = 16'b0000000000000000;
160                PixEND[12] = 16'b0000000000000000;
161                PixEND[13] = 16'b0000000000000000;
162                PixEND[14] = 16'b0000000000000000;
163                PixEND[15] = 16'b0000000000000000;
164             end
165
166             10: begin // END
167                PixEND[00] = 16'b0000000000000000;
168                PixEND[01] = 16'b0000000000000000;
169                PixEND[02] = 16'b0000000000000000;
170                PixEND[03] = 16'b0000000000000000;
171                PixEND[04] = 16'b0000000000000000;
172                PixEND[05] = 16'b0111010001011100;
173                PixEND[06] = 16'b0100011001010010;
174                PixEND[07] = 16'b0111010101010010;
175                PixEND[08] = 16'b0100010011010010;
176                PixEND[09] = 16'b0111010001011100;
177                PixEND[10] = 16'b0000000000000000;
178                PixEND[11] = 16'b0000000000000000;
179                PixEND[12] = 16'b0000000000000000;
180                PixEND[13] = 16'b0000000000000000;
181                PixEND[14] = 16'b0000000000000000;
182                PixEND[15] = 16'b0000000000000000;
183             end
184
185             11: begin // END
186                PixEND[00] = 16'b0000000000000000;
187                PixEND[01] = 16'b0000000000000000;
188                PixEND[02] = 16'b0000000000000000;
189                PixEND[03] = 16'b0000000000000000;
190                PixEND[04] = 16'b0000000000000000;
191                PixEND[05] = 16'b0111010001011100;
192                PixEND[06] = 16'b0100011001010010;
193                PixEND[07] = 16'b0111010101010010;
194                PixEND[08] = 16'b0100010011010010;
195                PixEND[09] = 16'b0111010001011100;
196                PixEND[10] = 16'b0000000000000000;
197                PixEND[11] = 16'b0000000000000000;
198                PixEND[12] = 16'b0000000000000000;
199                PixEND[13] = 16'b0000000000000000;
200                PixEND[14] = 16'b0000000000000000;
201                PixEND[15] = 16'b0000000000000000;
202             end
203
204             12: begin // END
205                PixEND[00] = 16'b0000000000000000;
206                PixEND[01] = 16'b0000000000000000;
207                PixEND[02] = 16'b0000000000000000;
208                PixEND[03] = 16'b0000000000000000;
209                PixEND[04] = 16'b0000000000000000;
210                PixEND[05] = 16'b0111010001011100;
211                PixEND[06] = 16'b0100011001010010;
212                PixEND[07] = 16'b0111010101010010;
213                PixEND[08] = 16'b0100010011010010;
214                PixEND[09] = 16'b0111010001011100;
215                PixEND[10] = 16'b0000000000000000;
216                PixEND[11] = 16'b0000000000000000;
217                PixEND[12] = 16'b0000000000000000;
218                PixEND[13] = 16'b0000000000000000;
219                PixEND[14] = 16'b0000000000000000;
```

```
220                PixEND[15] = 16'b0000000000000000;
221             end
222
223             13: begin // END
224                PixEND[00] = 16'b0000000000000000;
225                PixEND[01] = 16'b0000000000000000;
226                PixEND[02] = 16'b0000000000000000;
227                PixEND[03] = 16'b0000000000000000;
228                PixEND[04] = 16'b0000000000000000;
229                PixEND[05] = 16'b0111010001011100;
230                PixEND[06] = 16'b0100011001010010;
231                PixEND[07] = 16'b0111010101010010;
232                PixEND[08] = 16'b0100010011010010;
233                PixEND[09] = 16'b0111010001011100;
234                PixEND[10] = 16'b0000000000000000;
235                PixEND[11] = 16'b0000000000000000;
236                PixEND[12] = 16'b0000000000000000;
237                PixEND[13] = 16'b0000000000000000;
238                PixEND[14] = 16'b0000000000000000;
239                PixEND[15] = 16'b0000000000000000;
240             end
241
242             14: begin // END
243                PixEND[00] = 16'b0000000000000000;
244                PixEND[01] = 16'b0000000000000000;
245                PixEND[02] = 16'b0000000000000000;
246                PixEND[03] = 16'b0000000000000000;
247                PixEND[04] = 16'b0000000000000000;
248                PixEND[05] = 16'b0111010001011100;
249                PixEND[06] = 16'b0100011001010010;
250                PixEND[07] = 16'b0111010101010010;
251                PixEND[08] = 16'b0100010011010010;
252                PixEND[09] = 16'b0111010001011100;
253                PixEND[10] = 16'b0000000000000000;
254                PixEND[11] = 16'b0000000000000000;
255                PixEND[12] = 16'b0000000000000000;
256                PixEND[13] = 16'b0000000000000000;
257                PixEND[14] = 16'b0000000000000000;
258                PixEND[15] = 16'b0000000000000000;
259             end
260
261             15: begin // END
262                PixEND[00] = 16'b0000000000000000;
263                PixEND[01] = 16'b0000000000000000;
264                PixEND[02] = 16'b0000000000000000;
265                PixEND[03] = 16'b0000000000000000;
266                PixEND[04] = 16'b0000000000000000;
267                PixEND[05] = 16'b0111010001011100;
268                PixEND[06] = 16'b0100011001010010;
269                PixEND[07] = 16'b0111010101010010;
270                PixEND[08] = 16'b0100010011010010;
271                PixEND[09] = 16'b0111010001011100;
272                PixEND[10] = 16'b0000000000000000;
273                PixEND[11] = 16'b0000000000000000;
274                PixEND[12] = 16'b0000000000000000;
275                PixEND[13] = 16'b0000000000000000;
276                PixEND[14] = 16'b0000000000000000;
277                PixEND[15] = 16'b0000000000000000;
278             end
279
280             default: PixEND = '0;
281          endcase
282       end
283    endmodule
284
285    //module endExplode_testbench();
286    // logic [3:0] position;
287    // logic [15:0][15:0] PixEND;
288    //
289    // endExplode dut (.position, .PixEND);
290    //
291    // integer i;
292    // initial begin
```

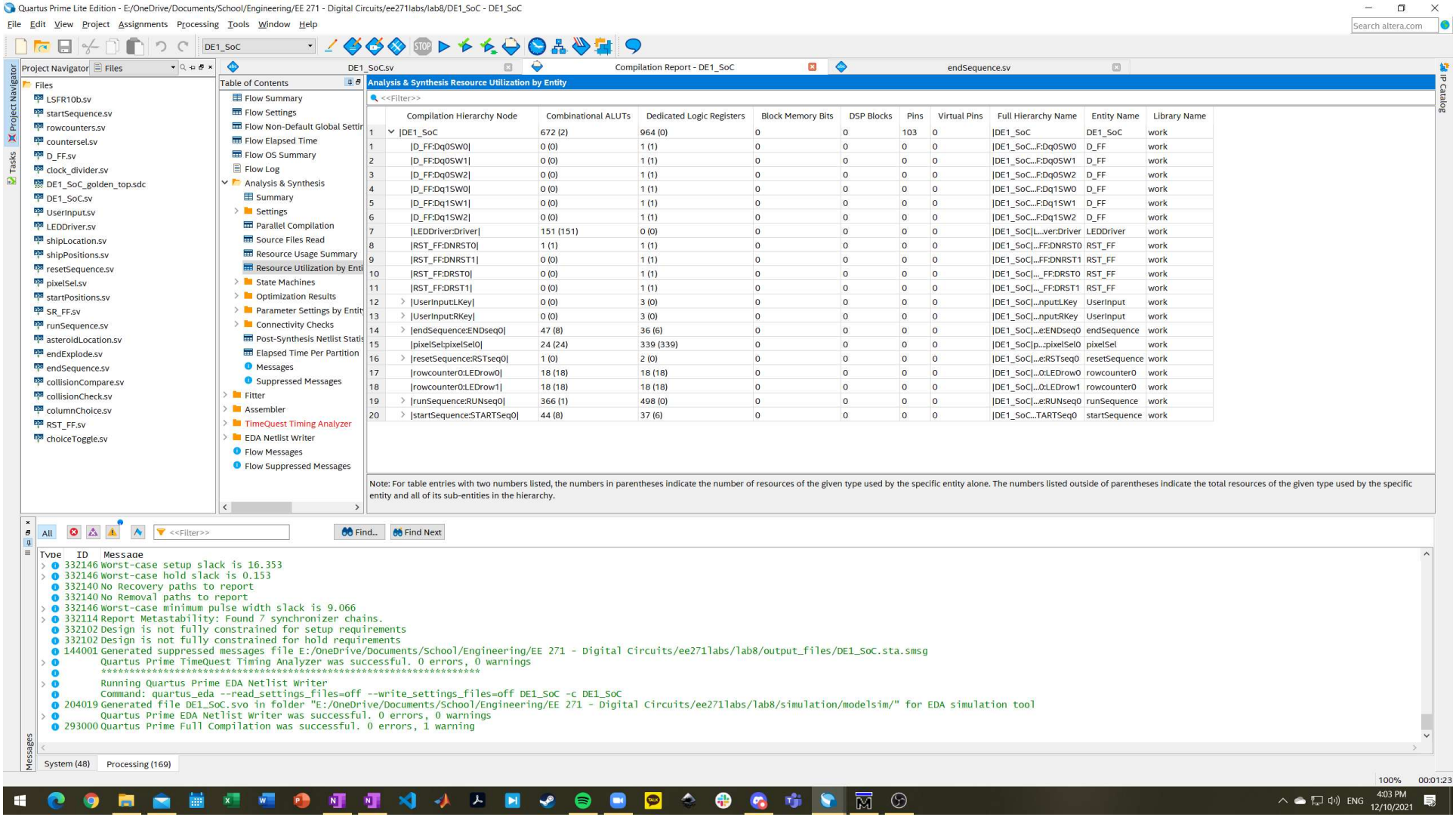
```
293   //    for (i=0; i<8; i++) begin
294   //       position = i; #10;
295   //    end
296   // end
297   //endmodule
```

```
 1   module LSFR10b (out, RST, CLK);
 2      input    logic       RST, CLK;
 3      output   logic [9:0] out;
 4
 5      logic q1,q2,q3,q4,q5,q6,q7,q8,q9,q10;
 6      logic fb; // feedback
 7      assign fb = ~( q10 ^ q7 ); // XNOR q10 and q7
 8
 9      D_FF D1  (.q(q1), .d(fb), .RST, .CLK);
10      D_FF D2  (.q(q2), .d(q1), .RST, .CLK);
11      D_FF D3  (.q(q3), .d(q2), .RST, .CLK);
12      D_FF D4  (.q(q4), .d(q3), .RST, .CLK);
13      D_FF D5  (.q(q5), .d(q4), .RST, .CLK);
14      D_FF D6  (.q(q6), .d(q5), .RST, .CLK);
15      D_FF D7  (.q(q7), .d(q6), .RST, .CLK);
16      D_FF D8  (.q(q8), .d(q7), .RST, .CLK);
17      D_FF D9  (.q(q9), .d(q8), .RST, .CLK);
18      D_FF D10 (.q(q10), .d(q9), .RST, .CLK);
19
20      assign out = {q10,q9,q8,q7,q6,q5,q4,q3,q2,q1};
21   endmodule
22
23   //module LSFR10b_testbench();
24   // logic       RST, CLOCK_50;
25   // logic [9:0] out;
26   //
27   // // Set up a simulated  clock
28   // parameter CLOCK_PERIOD = 100;
29   //
30   // initial begin
31   //    CLOCK_50 <= 0;
32   //    forever # (CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // Forever toggle clock
33   // end
34   //
35   // LSFR10b dut (.out, .RST, .CLK(CLOCK_50));
36   //
37   // initial begin
38   //                              @(posedge CLOCK_50);
39   //    RST <= 1;                 @(posedge CLOCK_50); // RST
40   //    // Run for 2^n + 2 cycles
41   //    // Check beginning and end
42   //    // Sequence should restart after 2^n
43   //    RST <= 0;   repeat(1026)   @(posedge CLOCK_50); // run for 2^n + 2 cycles
44   //    $stop;   // End the simulatioin
45   // end
46   //endmodule
```

```
1   // D flip-flop w/synchronous reset
2   module D_FF (q, d, RST, CLK);
3      output logic q;
4      input logic d, RST, CLK;
5
6      always_ff @(posedge CLK) begin // Hold val until clock edge
7         if (RST)
8            q <= 0; // On reset, set to 0
9         else
10           q <= d; // Otherwise out = d
11     end
12  endmodule
13
14  //module D_FF_testbench ();
15  // logic CLOCK_50, KEY[3:0], SW[9:0], q1, q0, RST;
16  //
17  // D_FF DFF0 (.q(q0), .d(KEY[3]), .RST(SW[9]), .CLK(CLOCK_50));
18  //
19  // D_FF dut (.q(q1), .d(q0), .RST(SW[9]), .CLK(CLOCK_50));
20  //
21  //
22  // // Set up a simulated  clock
23  // parameter CLOCK_PERIOD = 100;
24  // initial begin
25  //    CLOCK_50 <= 0;
26  //    forever # (CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // Forever toggle clock
27  // end
28  //
29  // initial begin
30  //                    repeat(1) @(posedge CLOCK_50);
31  //    SW[9]    <= 1; KEY[3] <= 0; repeat(4) @(posedge CLOCK_50); // reset on, key press off
32  //    SW[9]    <= 0; repeat(1) @(posedge CLOCK_50);   // reset off
33  //    KEY[3]   <= 1; repeat(4) @(posedge CLOCK_50);   // key press on
34  //    KEY[3]   <= 0; repeat(4) @(posedge CLOCK_50);   // key press off
35  //    KEY[3]   <= 1; repeat(4) @(posedge CLOCK_50);   // key press on
36  //    SW[9]    <= 1; repeat(5) @(posedge CLOCK_50);   // reset on
37  //    $stop;   // end simulation
38  // end
39  //
40  //endmodule
```

```
 1  module SR_FF (SR, Q, CLK, RST);
 2     input logic CLK, RST;
 3     input logic [1:0] SR;
 4     output logic Q;
 5
 6     logic ps;
 7
 8     always_ff @(posedge CLK) begin
 9        if (RST) begin
10           ps <= 0;
11        end else if (SR == 2'b00) begin
12           ps <= ps;
13        end else if (SR == 2'b01) begin
14           ps <= 0;
15        end else if (SR == 2'b10) begin
16           ps <= 1;
17        end else begin
18           ps <= 1'bx;
19        end
20     end
21
22     assign Q = ps;
23  endmodule
24
25  //module SR_FF_testbench();
26  // logic CLK, RST;
27  // logic [1:0] SR;
28  // logic Q;
29  //
30  // // Set up a simulated  clock
31  // parameter CLOCK_PERIOD = 100;
32  // initial begin
33  //    CLK <= 0;
34  //    forever # (CLOCK_PERIOD/2) CLK <= ~CLK;  // Forever toggle clock
35  // end
36  //
37  // SR_FF dut (.SR, .Q, .CLK, .RST);
38  //
39  // initial begin
40  //    RST <= 1;
41  //    SR = 2'b00; repeat(1) @(posedge CLK);
42  //    RST <= 0;
43  //    SR = 2'b10; repeat(2) @(posedge CLK);
44  //    SR = 2'b00; repeat(4) @(posedge CLK);
45  //    SR = 2'b01; repeat(2) @(posedge CLK);
46  //    SR = 2'b00; repeat(4) @(posedge CLK);
47  //    SR = 2'b11; repeat(4) @(posedge CLK);
48  //    $stop;
49  // end
50  //endmodule
```

```
1   // D flip-flop w/o reset used when input is RST signal
2   module RST_FF (q, d, CLK);
3      output logic q;
4      input logic d, CLK;
5
6      always_ff @(posedge CLK) begin // Hold val until clock edge
7         q <= d; // Otherwise out = d
8      end
9   endmodule
10
11  //module RST_FF_testbench ();
12  // logic CLOCK_50, KEY[3:0], SW[9:0], q1, q0, RST;
13  //
14  // RST_FF DFF0 (.q(q0), .d(KEY[3]), .RST(SW[9]), .CLK(CLOCK_50));
15  //
16  // RST_FF dut (.q(q1), .d(q0), .RST(SW[9]), .CLK(CLOCK_50));
17  //
18  //
19  // // Set up a simulated  clock
20  // parameter CLOCK_PERIOD = 100;
21  // initial begin
22  //    CLOCK_50 <= 0;
23  //    forever # (CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // Forever toggle clock
24  // end
25  //
26  // initial begin
27  //                    repeat(1) @(posedge CLOCK_50);
28  //    SW[9]    <= 1; KEY[3] <= 0; repeat(4) @(posedge CLOCK_50); // reset on, key press off
29  //    SW[9]    <= 0; repeat(1) @(posedge CLOCK_50);   // reset off
30  //    KEY[3]   <= 1; repeat(4) @(posedge CLOCK_50);   // key press on
31  //    KEY[3]   <= 0; repeat(4) @(posedge CLOCK_50);   // key press off
32  //    KEY[3]   <= 1; repeat(4) @(posedge CLOCK_50);   // key press on
33  //    SW[9]    <= 1; repeat(5) @(posedge CLOCK_50);   // reset on
34  //    $stop;   // end simulation
35  // end
36  //
37  //endmodule
```


Quartus Prime Lite Edition - E:/OneDrive/Documents/School/Engineering/EE 271 - Digital Circuits/ee271labs/lab8/DE1_SoC - DE1_SoC
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Files
LSFR10b.sv
startSequence.sv
rowcounters.sv
countersel.sv
D_FF.sv
clock_divider.sv
DE1_SoC_golden_top.sdc
DE1_SoC.sv
UserInput.sv
LEDDriver.sv
shipLocation.sv
shipPositions.sv
resetSequence.sv
pixelSel.sv
startPositions.sv
SR_FF.sv
runSequence.sv
asteroidLocation.sv
endExplode.sv
endSequence.sv
collisionCompare.sv
collisionCheck.sv
columnChoice.sv
RST_FF.sv
choiceToggle.sv
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global Settir
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Summary
Settings
Parallel Compilation
Source Files Read
Resource Usage Summary
Resource Utilization by Enti
State Machines
Optimization Results
Parameter Settings by Entit
Connectivity Checks
Post-Synthesis Netlist Stati
Elapsed Time Per Partition
Messages
Suppressed Messages
Fitter
Assembler
TimeQuest Timing Analyzer
EDA Netlist Writer
Flow Messages
Flow Suppressed Messages
DE1_SoC.sv
Compilation Report - DE1_SoC
endSequence.sv
Analysis & Synthesis Resource Utilization by Entity
<<Filter>>
Compilation Hierarchy Node | Combinational ALUTs | Dedicated Logic Registers | Block Memory Bits | DSP Blocks | Pins | Virtual Pins | Full Hierarchy Name | Entity Name | Library Name
1 |DE1_SoC | 672 (2) | 964 (0) | 0 | 0 | 103 | 0 | |DE1_SoC | DE1_SoC | work
1 |D_FF:Dq0SW0| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC...F:Dq0SW0 | D_FF | work
2 |D_FF:Dq0SW1| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC...F:Dq0SW1 | D_FF | work
3 |D_FF:Dq0SW2| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC...F:Dq0SW2 | D_FF | work
4 |D_FF:Dq1SW0| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC...F:Dq1SW0 | D_FF | work
5 |D_FF:Dq1SW1| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC...F:Dq1SW1 | D_FF | work
6 |D_FF:Dq1SW2| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC...F:Dq1SW2 | D_FF | work
7 |LEDDriver:Driver| | 151 (151) | 0 (0) | 0 | 0 | 0 | 0 | |DE1_SoC|L...ver:Driver | LEDDriver | work
8 |RST_FF:DNRST0| | 1 (1) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC|...FF:DNRST0 | RST_FF | work
9 |RST_FF:DNRST1| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC|...FF:DNRST1 | RST_FF | work
10 |RST_FF:DRST0| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC|..._FF:DRST0 | RST_FF | work
11 |RST_FF:DRST1| | 0 (0) | 1 (1) | 0 | 0 | 0 | 0 | |DE1_SoC|..._FF:DRST1 | RST_FF | work
12 |UserInput:LKey| | 0 (0) | 3 (0) | 0 | 0 | 0 | 0 | |DE1_SoC|...nput:LKey | UserInput | work
13 |UserInput:RKey| | 0 (0) | 3 (0) | 0 | 0 | 0 | 0 | |DE1_SoC|...nput:RKey | UserInput | work
14 |endSequence:ENDseq0| | 47 (8) | 36 (6) | 0 | 0 | 0 | 0 | |DE1_SoC|...e:ENDseq0 | endSequence | work
15 |pixelSel:pixelSel0| | 24 (24) | 339 (339) | 0 | 0 | 0 | 0 | |DE1_SoC|p...:pixelSel0 | pixelSel | work
16 |resetSequence:RSTseq0| | 1 (0) | 2 (0) | 0 | 0 | 0 | 0 | |DE1_SoC|...e:RSTseq0 | resetSequence | work
17 |rowcounter0:LEDrow0| | 18 (18) | 18 (18) | 0 | 0 | 0 | 0 | |DE1_SoC|...0:LEDrow0 | rowcounter0 | work
18 |rowcounter0:LEDrow1| | 18 (18) | 18 (18) | 0 | 0 | 0 | 0 | |DE1_SoC|...0:LEDrow1 | rowcounter0 | work
19 |runSequence:RUNseq0| | 366 (1) | 498 (0) | 0 | 0 | 0 | 0 | |DE1_SoC|...e:RUNseq0 | runSequence | work
20 |startSequence:STARTSeq0| | 44 (8) | 37 (6) | 0 | 0 | 0 | 0 | |DE1_SoC...TARTSeq0 | startSequence | work
Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.
All
<<Filter>>
Find...
Find Next
Type ID Message
332146 Worst-case setup slack is 16.353
332146 Worst-case hold slack is 0.153
332140 No Recovery paths to report
332140 No Removal paths to report
332146 Worst-case minimum pulse width slack is 9.066
332114 Report Metastability: Found 7 synchronizer chains.
332102 Design is not fully constrained for setup requirements
332102 Design is not fully constrained for hold requirements
144001 Generated suppressed messages file E:/OneDrive/Documents/School/Engineering/EE 271 - Digital Circuits/ee271labs/lab8/output_files/DE1_SoC.sta.smsg
Quartus Prime TimeQuest Timing Analyzer was successful. 0 errors, 0 warnings
Running Quartus Prime EDA Netlist Writer
Command: quartus_eda --read_settings_files=off --write_settings_files=off DE1_SoC -c DE1_SoC
204019 Generated file DE1_SoC.svo in folder "E:/OneDrive/Documents/School/Engineering/EE 271 - Digital Circuits/ee271labs/lab8/simulation/modelsim/" for EDA simulation tool
Quartus Prime EDA Netlist Writer was successful. 0 errors, 0 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 1 warning
Messages
System (48)
Processing (169)
100%
00:01:23
ENG
4:03 PM
12/10/2021