Connor Aksama EE 371 January 27, 2023 Lab 3 Report

Procedure

This lab is comprised of two tasks. The first task involved implementing Bersenham's algorithm in Verilog and demonstrating its functionality via a VGA display; the second task involved utilizing this line drawing algorithm to animate a line on the video display.

Task 1

I approached this task by first planning out the implementation of the C-style algorithm in a Verilog module. I determined that the initial part of the algorithm before the main loop should be contained with the combinational block of the module since these calculations were entirely based on the input points. Thus, the portion of the algorithm contained in the main loop would belong in the sequential block of the module since the output values were dependent on the previous sequence of values.

A testbench was then created for this module, simulated in ModelSIm, and was compared against the output from a procedural implementation of the algorithm for correctness.

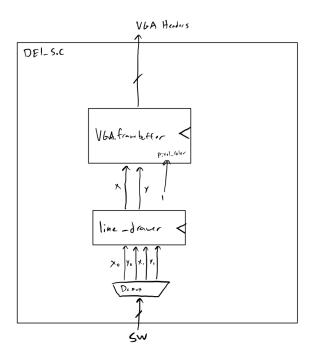


Figure 1. Top-Level Block Diagram for Lab 3 Task 1.

Task 2

I approached this task by first planning out the animation that was going to be displayed. I decided that I would implement this animation as a sequence of timed "frames" (pairs of points) interlaced by screen clears. To achieve this, I created a timed module that would cycle through a series of pre-programmed points once an internal timer reached a certain value. The module would output the next pair of points to be drawn on the screen, which was fed into the line drawing module from Task 1.

Additionally, a module that would clear the screen when enabled was created. The function of this module is simply to cycle through and output each coordinate on the screen in sequence. This point is directly fed into the VGA buffer module along with a black pixel color. Given enough cycles, this would clear the entire video display.

These modules were then tested and simulated in ModelSim before uploaded and tested on hardware.

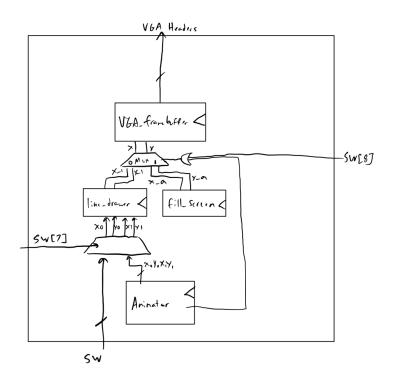


Figure 2. Top-Level Block Diagram for Lab 3 Task 2.

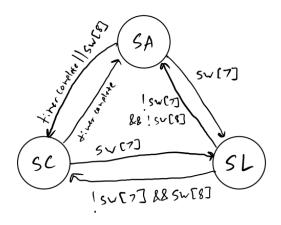


Figure 3. FSM for the Lab 3 Task 2 Design.

This FSM transitions between types of output to the VGA display. In state SA, a line is drawn to the screen whose endpoints are determined by a sequence stored in the animator module. After an internal timer in the animator module expires, it outputs a clear signal (state SC), which triggers the fill screen module to write to each pixel on the screen. This state is also enterable when SW8 is raised and SW7 is low. If SW7 is raised (state SL), the endpoints of the line are determined by the status of the input switches.

Results

Task 1

The following are screenshots from ModelSim simulations for each module used in the simulated RAM module.

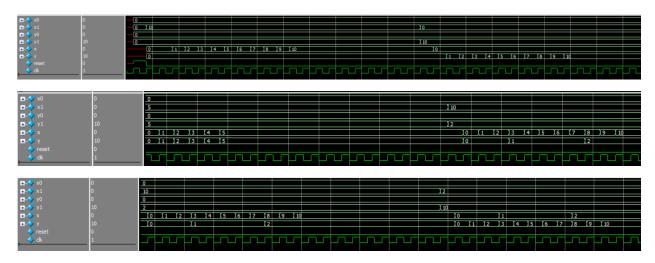


Figure 4. ModelSim Waveform for the Line Drawer testbench.

This testbench runs through different types of line slopes: horizontal, vertical, diagonal, shallow and steep. The output x and y were compared against the output of a C implementation of Bersenham's algorithm, which verified that the module was outputting the correct values for x and y each clock cycle.

The testbench for the top level module can be found in the following section.

Task 2

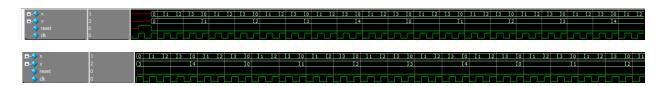


Figure 5. ModelSim Waveform for the Fill Screen testbench.

This testbench instantiates the fill screen module for a test screen with width 4 and height 5. The testbench then monitors the output of the module, which is expected to cycle through each pixel on the screen. The waveform shows that every pixel on this screen gets output and cycles back to (0,0).

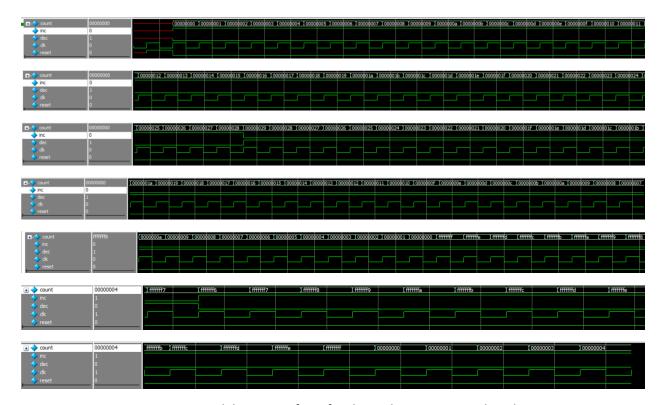


Figure 6. ModelSim Waveform for the 32-bit Counter testbench.

This testbench shows the counter being incremented, decremented, underflowed, and overflowed correctly.

- - → x0	397	0 420	(419				(418		(417	(416		(4)	15)	414	(412	(411	(410	(409	(407	406	(404)
■	243	0 220	221				222		(223	224		2:	25	226	228	(229	230	231	233	234	(236)
	303	0 240	243	246	249	(252	255	258	(261	264	26	7 (2)	70):	273	276	279	282	285	288	290	(293
- y 1	177	0 240	237	(234	231	(228	225	222	(219	(216	(21	3 (2)	10 (207	204	(201	(198	195	192	190	(187
🔷 dear	0														$\overline{}$						
∳ dk	0	السدا	$\overline{\mathbf{M}}$	MM	ww	ww	MM	ww	MM	$\overline{\mathbf{u}}$	M	ww	MM	$\overline{\mathbf{M}}$	$\overline{\mathbf{M}}$	MM	$\overline{\text{MM}}$	$\overline{\mathbf{m}}$	$\overline{\mathbf{M}}$	$\overline{\mathbf{m}}$	$\overline{\mathbf{u}}$
reset	0																				
V																					
	_																				
₽ - → x0	397	X402									383	(381	378	376	(373						
□ - ♦ x1	243								^		257	259	262	264	267						
⊕ → y0	303		=						^		317	319	320	322	(324		, <u>, , , , , , , , , , , , , , , , , , </u>				(332
⊕ - ♦ y1	177	(184	182	179	177	174	72 (1	170 1	168	165	163	161	160	158	(156	(15	4 (1	53 (15	1 (15	0 (149	(148
dear "	0							. ہے۔							-4				. المال		
♦ dk	0	וחרחר	ww	חחחח		וחחחח	וחחחו	וחולחה	חחה		חחח	whv		חחח	וחחח	חחח			חחחח	וחלחם	ww
	_ "	_											$\overline{}$								
B - 4 x0	397	(353	Y 350	347	344 X	341	338 X	335 Y	332 X	329	Y 326	323	() 420	(419				(4		X 417	7 (416
□- ×0 □-	243	287									X 314	323) (120) 2:), 223	
. → y0	303	1334) 336			338		339	311	1314	1,317) (220		Y 246	124	9 Y2				
	177	146	1145			T	142		141				1 1 240		1 234				25 1 22		
dear	0	1140	(143	1744		143	172	- 	177	$\overline{}$	+			1,237	123			20 /2	3 122	4 (219	(210
dk dk dk dk	n		ተለከተ			1077				7070	겁니다	7000	5007	700	77	7074		427	. 477		\overline{M}
→ reset	0		30000	, , , , , ,			0000	0000	0000	0000	7 1 1 1 1					3000		1000		00001	
	-																				
∓ - 4 x0	397 243	417	(1 16	X	415	414 X	12 (4	11 (4	10	109	407	¥406	(404	(402	(400	(399	1 39	7 (39	392	(390	1388
■ - ♦ x1	243	223	(224		225	226) 2	28 (2	229 2	30 (231	233	(234	236	238	240	(241	. (24	3 (24)	248	(250	(252
. → y0	303	261	(264	267	270 (273) 2	76 (2	279 (2	82 (285	288	(290	293	296	298	(301	.)(30	3 (306	(308	(310	(312
■ - ◇ y1	177	219	(216	213	210	207 (2	04 (2	201 (1	98	195	192	(190	(187	184	182	(179) 17	7 (174	172	(170	(168
🔷 dear	0		\Box		$\neg \bot$	$\neg \bot$			╅	$\neg \bot$			ᅫ	╧		ட	யு				\perp
🔷 dk	0	MMM	MM	MM	ww	NUVU	MM	MM	സസ	M	MM	\mathbf{n}	MM	M	M	MM	MM	ww	MM	MM	MMM
reset	0																				
,																					
- 4		I v	v					V-					v								
⊡ - ∲ x0	397 243				78 3								356	353	350	347	(344		(338	335	
1 x1	303				62 2								284	287	290	293	296		(302	305	
 → y0	177				20 3								332	334	335	(336		(337) 143	(338		(339
□ - → y1	1//	165	163	161 (1	.60 1	58 (19	6 (15	54 (15	3 1	51	150	149	148	146	145	(144		, 143	(142		141
	0													70 46		-					MMM
	0	10000	hvv	ا ال الدام	וחחחר	ww			hanr	חחחי	nnnh		חחח	וחחחו	ם טיטינ	hanr	וחאחו	0001	1000		
	10																				

+ - ♦ x0	397	(329	326 (32	3 (420	419			(418		(417	416	415	(414	(412	411 (41	409	(407
⊕ - ♦ x1	243	(311	314 (31	7 (220	221			222		(223	224	225	226	(228	229 (23	231	233
	303	339		240	243	(246	249 (25)	2 (255	258	(261	(264) 267	270	273	(276	279 (28	2 (285	288
⊕ - ♦ y1	177	141) (240	237	(234	231) 228	8 (225	222	219	(216)(213	210	207	(204	201 (19	195	192
🔷 dear	0						\vdash	\uparrow									
🔷 dk	0	NUUUU	\mathbf{m}	MMM)	MM	www	www	www	www	MM	www	MMM	mm	ww	www	www	سس
reset	0																
-																	

Figure 7. ModelSim Waveform for the Animator testbench.

This testbench shows the animator cycling through each "frame" (set of line endpoints) being sequentially output after the specified frame duration of 2 clock cycles (for testing purposes). After two cycles of the current frame being output, the clear signal is raised for 2 cycles, then the next frame is output. This process is continued for every frame in the sequence, which then cycles back to the beginning of the sequence once the last frame is output.

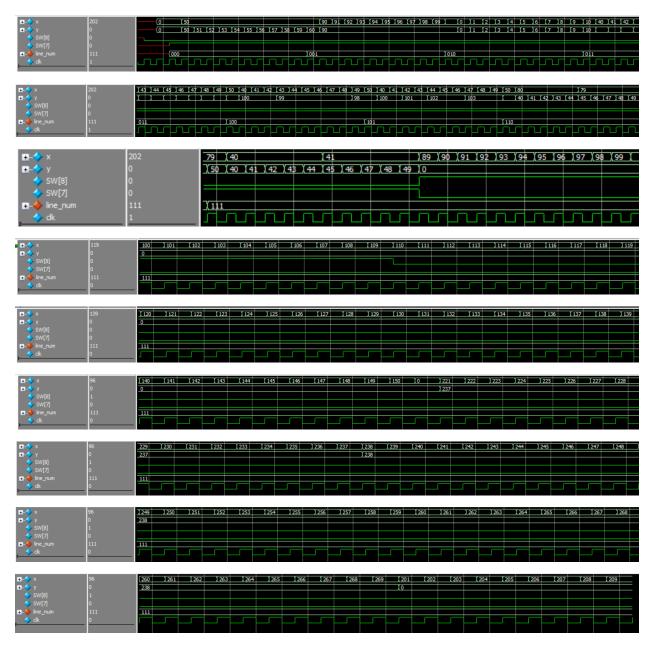


Figure 8. ModelSim Waveform for the top level DE1_SoC testbench.

This testbench first shows a portion of each of the arbitrary lines drawable in the SL state (i.e. when SW7 is raised). Next, the simulation demonstrates that when SW7 is lowered and SW8 is raised,

Final Product

The overarching goal of this lab was to gain experience working with a VGA controller, a VGA buffer, and implementing a procedural algorithm in Verilog. The main result of Task 1 was a module that implemented Bersenham's algorithm by taking in two cartesian coordinates as input, and outputting the coordinates of the line between the two points. As requested by the spec, the functionality of this module was then demonstrated by drawing different arbitrary lines on the display, which were toggled using the onboard switches.

In Task 2, I created additional modules to display an animation on the display. When the animation was enabled, a line would rotate and jump around the screen in a cyclic manner. As stated in the spec, this display also had the ability to be cleared to black, which is achieved by turning on an onboard switch.

Appendix: SystemVerilog Code

(See next page)

```
/**********
 2
 3
     animator.sv
 4
     ***********
 5
 6
 7
8
     * Connor Aksama
9
     * 01/30/2023
     * CSE 371
10
11
     * Lab 3
     * /
12
13
14
15
     * Outputs the endpoints of the current line to draw on the display and a signal
      indicating whether
16
      * the scren should be cleared.
17
18
     * Inputs:
19
     * clk [1 bit] - the clock to use for this module
     ^{\star} reset [1 bit] - the reset signal for this module; resets to the first frame when
20
     raised
21
     * Outputs:
22
23
     * x0 [10 bit] - The x coordinate of the first point of the line to draw
24
        y0 [9 bit] - The y coordinate of the first point of the line to draw
25
        x1 [10 bit] - The x coordinate of the second point of the line to draw
        y1 [9 bit] - The y coordinate of the second point of the line to draw
26
27
        clear [1 bit] - Signal to indicate when the screen should be cleared
    * /
28
29
    module animator#(
30
        parameter frame time = 8333333
31
        ) (
        output logic [9:0] x0, x1
32
33
         ,output logic [8:0] y0, y1
34
         ,output logic clear
35
         ,input logic clk, reset
36
         );
37
38
         logic next frame, next frame q;
39
         logic [31:0] frame num;
40
41
        // Counter to time the duration of each frame
42
        // timer out -> number of cycles current frame is active
43
        // inc on every cycle; no dec; use same clock, reset on frame switch/reset
44
        logic [31:0] timer out;
45
         counter 32b timer (.count(timer out), .inc(1'b1), .dec(1'b0), .clk, .reset(reset |
         next frame));
46
47
         // Counter to keep track of the current frame number
48
         // frame num -> current frame number
49
         // inc on every frame switch; no dec; use same clock, reset on end of sequence/reset
50
         logic frame reset;
51
         counter_32b frame_counter (.count(frame_num), .inc(next_frame_q), .dec(1'b0), .clk, .
         reset(reset | frame reset));
52
53
         // Determine if timer expired, switch to next frame logic
54
         always_comb begin
55
56
             // Check for next frame
57
            next_frame = (timer_out >= frame_time - 1) && !next_frame_q;
58
59
             // Clear screen every other frame
60
            clear = ~frame num[0];
61
62
            // End of sequence reached
63
            frame_reset = (frame_num >= 100);
64
            // Frames
65
```

```
66
             case (frame num[31:1])
67
                 0:begin x0=420; y0=240; x1=220; y1=240; end 1:begin x0=419; y0=243; x1=221; y1=237;
68
                 end 2:begin x0=419; y0=246; x1=221; y1=234; end 3:begin x0=419; y0=249; x1=221; y1
69
                 4:begin x0=419;y0=252;x1=221;y1=228;end 5:begin x0=418;y0=255;x1=222;y1=225;
                 end 6:begin x0=418;y0=258;x1=222;y1=222;end 7:begin x0=417;y0=261;x1=223;y1
                 =219;end
70
                 8:begin x0=416;y0=264;x1=224;y1=216;end 9:begin x0=416;y0=267;x1=224;y1=213;
                 end 10:begin x0=415;y0=270;x1=225;y1=210;end 11:begin x0=414;y0=273;x1=226;
                 12: begin x0=412; y0=276; x1=228; y1=204; end 13: begin x0=411; y0=279; x1=229; y1=
71
                 201; end 14: begin x0=410; y0=282; x1=230; y1=198; end 15: begin x0=409; y0=285; x1=
                 231; y1=195; end
                 16:begin x0=407;y0=288;x1=233;y1=192;end 17:begin x0=406;y0=290;x1=234;y1=
72
                 190; end 18: begin x0=404; y0=293; x1=236; y1=187; end 19: begin x0=402; y0=296; x1=
                 238; y1=184; end
                 20:begin x0=400;y0=298;x1=240;y1=182;end 21:begin x0=399;y0=301;x1=241;y1=
73
                 179; end 22: begin x0=397; y0=303; x1=243; y1=177; end 23: begin x0=395; y0=306; x1=
                 245; y1=174; end
74
                 24:begin x0=392;y0=308;x1=248;y1=172;end 25:begin x0=390;y0=310;x1=250;y1=
                 170; end 26:begin x0=388; y0=312; x1=252; y1=168; end 27:begin x0=386; y0=315; x1=
                 254; y1=165; end
75
                 28:begin x0=383;y0=317;x1=257;y1=163;end 29:begin x0=381;y0=319;x1=259;y1=
                 161; end 30: begin x0=378; y0=320; x1=262; y1=160; end 31: begin x0=376; y0=322; x1=
                 264; y1=158; end
                 32: begin x0=373; y0=324; x1=267; y1=156; end 33: begin x0=370; y0=326; x1=270; y1=
76
                 154; end 34: begin x0=368; y0=327; x1=272; y1=153; end 35: begin x0=365; y0=329; x1=
                 275; y1=151; end
77
                 36:begin x0=362;y0=330;x1=278;y1=150;end 37:begin x0=359;y0=331;x1=281;y1=
                 149;end 38:begin x0=356;y0=332;x1=284;y1=148;end 39:begin x0=353;y0=334;x1=
                 287; y1=146; end
78
                 40:begin x0=350;y0=335;x1=290;y1=145;end 41:begin x0=347;y0=336;x1=293;y1=
                 144; end 42: begin x0=344; y0=336; x1=296; y1=144; end 43: begin x0=341; y0=337; x1=
                 299; y1=143; end
                 44:begin x0=338;y0=338;x1=302;y1=142;end 45:begin x0=335;y0=338;x1=305;y1=
79
                 142; end 46: begin x0=332; y0=339; x1=308; y1=141; end 47: begin x0=329; y0=339; x1=
                 311; y1=141; end
80
                 48:begin x0=326;y0=339;x1=314;y1=141;end 49:begin x0=323;y0=339;x1=317;y1=
                 141;end 50:begin x0=320;y0=340;x1=320;y1=140;end 51:begin x0=317;y0=339;x1=
                 323;y1=141;end
                 52:begin x0=314;y0=339;x1=326;y1=141;end 53:begin x0=311;y0=339;x1=329;y1=
81
                 141;end 54:begin x0=308;y0=339;x1=332;y1=141;end 55:begin x0=305;y0=338;x1=
                 335; y1=142; end
82
                 56:begin x0=302;y0=338;x1=338;y1=142;end 57:begin x0=299;y0=337;x1=341;y1=
                 143; end 58: begin x0=296; y0=336; x1=344; y1=144; end 59: begin x0=293; y0=336; x1=
                 347;y1=144;end
                 60:begin x0=290; y0=335; x1=350; y1=145; end 61:begin x0=287; y0=334; x1=353; y1=
83
                 146; end 62: begin x0=284; y0=332; x1=356; y1=148; end 63: begin x0=281; y0=331; x1=
                 359; y1=149; end
84
                 64: begin x0=278; y0=330; x1=362; y1=150; end 65: begin x0=275; y0=329; x1=365; y1=
                 151; end 66: begin x0=272; y0=327; x1=368; y1=153; end 67: begin x0=270; y0=326; x1=
                 370;y1=154;end
                 68:begin x0=267;y0=324;x1=373;y1=156;end 69:begin x0=264;y0=322;x1=376;y1=
85
                 158; end 70: begin x0=262; y0=320; x1=378; y1=160; end 71: begin x0=259; y0=319; x1=
                 381; y1=161; end
86
                 72:begin x0=257; y0=317; x1=383; y1=163; end 73:begin x0=254; y0=315; x1=386; y1=
                 165; end 74: begin x0=252; y0=312; x1=388; y1=168; end 75: begin x0=250; y0=310; x1=
                 390;y1=170;end
                 76:begin x0=248;y0=308;x1=392;y1=172;end 77:begin x0=245;y0=306;x1=395;y1=
                 174; end 78: begin x0=243; y0=303; x1=397; y1=177; end 79: begin x0=241; y0=301; x1=
                 399;y1=179;end
                 80:begin x0=240; y0=298; x1=400; y1=182; end 81:begin x0=238; y0=296; x1=402; y1=
88
                 184;end 82:begin x0=236;y0=293;x1=404;y1=187;end 83:begin x0=234;y0=290;x1=
                 406; y1=190; end
                 84:begin x0=233;y0=288;x1=407;y1=192;end 85:begin x0=231;y0=285;x1=409;y1=
89
                 195;end 86:begin x0=230;y0=282;x1=410;y1=198;end 87:begin x0=229;y0=279;x1=
                 411; y1=201; end
90
                 88:begin x0=228;y0=276;x1=412;y1=204;end 89:begin x0=226;y0=273;x1=414;y1=
```

```
207; end 90: begin x0=225; y0=270; x1=415; y1=210; end 91: begin x0=224; y0=267; x1=
                   416; y1=213; end
 91
                   92:begin x0=224; y0=264; x1=416; y1=216; end 93:begin x0=223; y0=261; x1=417; y1=
                   219; end 94: begin x0=222; y0=258; x1=418; y1=222; end 95: begin x0=222; y0=255; x1=
                   418; y1=225; end
 92
                   96:begin x0=221;y0=252;x1=419;y1=228;end 97:begin x0=221;y0=249;x1=419;y1=
                   231; end 98: begin x0=221; y0=246; x1=419; y1=234; end 99: begin x0=221; y0=243; x1=
                  419;y1=237;end
 93
                  default: begin x0=0; y0=0; x1=0; y1=0; end
 94
 95
              endcase
 96
 97
          end
 98
 99
          // Register the next frame signal to frame counter
100
          always ff @(posedge clk) begin
101
              if (reset) begin
102
                  next frame q <= 0;</pre>
103
              end else begin
104
                  next frame q <= next frame;</pre>
105
              end
106
          end
107
108
      endmodule // animator
109
110
      // Module to test the functionality of the animator module
111
      module animator testbench();
112
113
          logic [9:0] x0, x1;
114
          logic [8:0] y0, y1;
115
          logic clear;
116
          logic clk, reset;
117
118
          animator \#(2) dut (.*);
119
120
          parameter CLOCK PERIOD = 100;
121
          initial begin
122
              clk <= 0;
123
              forever #(CLOCK PERIOD / 2) clk <= ~clk;</pre>
124
          end
125
126
          initial begin
127
              integer i;
128
129
              @(posedge clk) reset <= 1;</pre>
130
              @(posedge clk) reset <= 0;</pre>
131
132
              // Run through each of the frames in the animation
133
              for (i = 0; i < 500; i++) @(posedge clk);</pre>
134
135
              $stop;
136
          end
137
138
139
      endmodule // animator testbench
140
141
      /**********
142
143
      counter 32b.sv
144
      ***********
145
146
147
       * Connor Aksama
148
       * 01/30/2023
149
150
       * CSE 371
       * Lab 3
151
152
      */
```

153

```
154
155
      * A thirty two-bit increment/decrement counter. Overflows to 8'h0; underflows to
       8'hFFFF FFFF.
156
       * reset takes precedence; if inc and dec are simultaneously raised, count is unchanged.
157
       * Inputs:
158
          inc [1 bit] - Increments the count by 1 when raised. Does nothing if signal is low.
159
       * dec [1 bit] - Decrements the count by 1 when raised. Does nothing if signal is low.
160
       * clk [1 bit] - The clock to use for this module.
161
       * reset [1 bit] - Resets the counter to 0 when raised. Does nothing if signal is low.
162
163
       * Outputs:
164
               count [32 bit] - The count of this counter.
165
       * /
166
      module counter 32b(
167
          output logic [31:0] count
           ,input logic inc, dec, clk, reset
168
169
          );
170
171
          // Handles increment/decrement/reset operations
172
          always ff @(posedge clk) begin
173
174
               if (reset)
175
                   count <= 32'b0;
176
               else if (inc & ~dec)
177
                   count <= count + 32'b1;</pre>
178
               else if (dec & ~inc)
179
                   count <= count - 32'b1;
180
181
          end
182
183
      endmodule // counter 32b
184
185
      // Module to test the functionality of the counter 32b module
186
      module counter 32b testbench();
187
188
           logic [31:0] count;
189
          logic inc, dec, clk, reset;
190
191
          counter 32b dut (.*);
192
193
          parameter CLOCK PERIOD = 100;
194
          initial begin
195
               clk \leq 0:
196
               forever #(CLOCK PERIOD / 2) clk <= ~clk;</pre>
197
          end
198
199
          initial begin
200
               integer i;
201
202
               @(posedge clk) reset <= 1;</pre>
203
               @ (posedge clk) reset \le 0; inc \le 1; dec \le 0;
204
               for (i = 0; i < 40; i++) begin
205
                   @(posedge clk);
206
207
208
               @ (posedge clk) reset \leftarrow 0; inc \leftarrow 0; dec \leftarrow 1;
209
               for (i = 0; i < 50; i++) begin
210
                   @(posedge clk);
211
               end
212
213
               @ (posedge clk) reset \leftarrow 0; inc \leftarrow 1; dec \leftarrow 0;
214
               for (i = 0; i < 15; i++) begin
215
                   @(posedge clk);
216
               end
217
218
               $stop;
219
          end
220
```

221

endmodule // counter 32b testbench

```
222
223
      /**********
224
225
      DE1 SoC.sv
226
227
      ************
228
229
      * Connor Aksama
230
      * 01/30/2023
231
232
      * CSE 371
      * Lab 3
233
234
      * /
235
236
237
      * Top-level module for Lab 3. Instantiates drawing logic modules and connections to VGA
      controller
238
      * Inputs:
239
      * SW [10 bit] - The 10 onboard switches, respectively.
240
      * KEY [4 bit] - The 4 onboard keys, respectively.
241
242
      * CLOCK 50 [1 bit] - The system clock to use for this module.
243
244
       * Outputs:
245
       * HEXO [7 bit] - Data to show on the HEXO display, formatted in standard 7 segment
       display format.
       * HEX1 [7 bit] - Data to show on the HEX1 display, formatted in standard 7 segment
246
      display format.
247
       * HEX2 [7 bit] - Data to show on the HEX1 display, formatted in standard 7 segment
       display format.
248
      * HEX3 [7 bit] - Data to show on the HEX1 display, formatted in standard 7 segment
       display format.
      * HEX4 [7 bit] - Data to show on the HEX4 display, formatted in standard 7 segment
249
       display format.
250
      * HEX5 [7 bit] - Data to show on the HEX5 display, formatted in standard 7 segment
       display format.
251
      * LEDR [10 bit] - Signal to output to 10 onboard LEDs, respectively.
252
         VGA R [8 bit] - The red channel value of the pixel to write to the display
253
         VGA G [8 bit] - The green channel value of the pixel to write to the display
254
         VGA B [8 bit] - The blue channel value of the pixel to write to the display
255
      * VGA BLANK N [1 bit] - Blank area signal
      * VGA CLK [1 bit] - Divided clock for VGA signal
256
      * VGA HS [1 bit] - VGA timing signal
257
258
      * VGA SYNC N [1 bit] - Unused
259
         VGA VS [1 bit] - VGA timing signal
     * /
260
261
     module DE1 SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, CLOCK 50,
262
         VGA R, VGA G, VGA B, VGA BLANK N, VGA CLK, VGA HS, VGA SYNC N, VGA VS);
263
264
          output logic [6:0] HEXO, HEX1, HEX2, HEX3, HEX4, HEX5;
265
          output logic [9:0] LEDR;
266
          input logic [3:0] KEY;
267
          input logic [9:0] SW;
268
269
          input CLOCK 50;
270
          output [7:0] VGA R;
271
          output [7:0] VGA G;
272
          output [7:0] VGA B;
273
          output VGA BLANK N;
274
         output VGA CLK;
275
          output VGA_HS;
276
          output VGA SYNC N;
277
         output VGA VS;
278
279
         assign HEX1 = '1;
280
          assign HEX0 = '1;
281
         assign HEX2 = '1;
          assign HEX3 = '1;
282
          assign HEX4 = '1;
283
```

```
284
          assign HEX5 = '1;
285
          assign LEDR = SW;
286
287
          // Outputs from drawing modules
288
          logic [9:0] x0_1, x0_a, x1, x1_1, x1_a;
289
          logic [8:0] y0_1, y0_a, y1, y1_1, y1_a;
290
          // Inputs into frame buffer
291
          logic [9:0] x 1, x a, x c, x;
          logic [8:0] y 1, y a, y c, y;
292
293
          logic frame start;
294
          logic pixel color;
295
          logic reset, clear screen;
296
297
          assign reset = SW[9];
298
          assign clear screen = SW[8];
299
300
301
          ////// DOUBLE FRAME BUFFER //////
302
          logic dfb en;
303
          assign dfb en = 1'b0;
          304
305
          VGA framebuffer fb(.clk(CLOCK 50), .rst(1'b0), .x, .y,
306
307
                       .pixel_color, .pixel_write(1'b1), .dfb_en, .frame start,
308
                       .VGA R, .VGA G, .VGA B, .VGA CLK, .VGA HS, .VGA VS,
                       .VGA_BLANK_N, .VGA SYNC N);
309
310
311
          // Line Drawing Module
312
          // Inputs -> system clock, reset signal, (x0_1, y0_1), (x1_1, y1_1) -> from local
          temp logic; defines endpoints of line
          // Outputs -> (x_1, y_1) -> pixel to draw to VGA - changes by at most one pixel each
313
          clock cycle, starting one
                                       cycle after input points change
314
          //
315
          line drawer lines (.clk(CLOCK 50), .reset,
                       .x0(x0_1), .y0(y0_1), .x1(x1_1), .y1(y1_1), .x(x_1), .y(y_1);
316
317
318
          // Animator Module
319
          // Inputs -> system clock, reset signal
320
          // Outputs \rightarrow (x0_a, y0_a), (x1_a, y1_a) \rightarrow Endpoints of line to draw
321
          //
                        clear -> signal is high when screen should be cleared
322
          logic clear;
323
          animator \#(50000000) anim (.x0(x0_a), .x1(x1_a), .y0(y0_a), .y1(y1_a), .clear, .clk(
          CLOCK 50), .reset);
324
325
          // Screen Clearning Module; cycles through each pixel on screen
          // Inputs \rightarrow system clock, reset signal
326
327
          // Output -> (x c, y c) -> Coordinate of pixel to clear on display
328
          fill space \#(640,480) clearer (.x(x_c), .y(y_c), .clk(CLOCK_50), .reset);
329
330
          // Multiplex endpoints into line drawing module
331
          // Multiplex coordinates into frame buffer
332
          always comb begin
333
              if (SW[7]) begin
334
                  // Display arbitrary lines based on SW input
335
                  case (SW[2:0])
336
                      0: begin
337
                          // vertical
338
                          x0 1 = 50;
339
                          y0 1 = 50;
340
                          x1_1 = 50;
                           y1_1 = 400;
341
342
                          pixel color = 1'b1;
343
                      end
344
                      1: begin
345
                          // horizontal
346
                          x0 1 = 90;
                          y0^{-1} = 90;
347
                          x1 1 = 450;
348
349
                          y1 1 = 90;
```

```
350
                            pixel color = 1'b1;
351
                        end
352
                        2: begin
353
                            // negative diagonal
354
                            x0 1 = 240;
355
                            y0_1 = 240;
356
                            x1_1 = 0;
                            y1_1 = 0;
357
358
                            pixel color = 1'b1;
359
                        end
360
                        3: begin
                            // positive diagonal
361
                            x0 1 = 400;
362
                            y0^{-1} = 40;
363
                            x1^{-1} = 40;
364
365
                            y1 1 = 400;
366
                            pixel_color = 1'b1;
367
                        end
368
                        4: begin
369
                            // positive shallow
370
                            x0 1 = 400;
371
                            y0 1 = 40;
372
                            x1 1 = 40;
373
                            y1^{-1} = 100;
374
                            pixel color = 1'b1;
375
                        end
376
                        5: begin
377
                            // negative shallow
378
                            x0 1 = 300;
                            y0^{-1} = 200;
379
380
                            x1 1 = 40;
381
                            y1 1 = 100;
382
                            pixel color = 1'b1;
383
                        end
384
                        6: begin
385
                             // positive steep
                            x0 1 = 40;
386
387
                            y0 1 = 400;
                            x1_1 = 80;
388
389
                            y1 1 = 40;
390
                            pixel color = 1'b1;
391
                        end
392
                        7: begin
393
                            // negative steep
394
                            x0 1 = 40;
395
                            y0 1 = 40;
                            x1^{-1} = 80;
396
397
                            y1 1 = 400;
398
                            pixel color = 1'b1;
399
                        end
400
                        default: begin
401
                            x0 1 = 0;
402
                            y0 1 = 0;
403
                            x1 1 = 0;
404
                            y1 1 = 0;
405
                            pixel_color = 1'b1;
406
                        end
407
408
                    endcase
409
                    x = x_1;
410
                    y = y_1;
411
               end
412
               else if (clear | clear screen) begin
413
                    // Clear Screen
414
                    x0 1 = 0;
415
                    y0 1 = 0;
                    x1^{-1} = 0;
416
417
                    y1^{-1} = 0;
418
                    pixel_color = 1'b0;
```

```
419
                   x = x c;
420
                   y = y_c;
421
               end
422
               else begin
423
                   // Display animation
424
                   x0_1 = x0_a;
                   y0_1 = y0_a;
425
                   x1^{-}1 = x1^{-}a;
426
427
                   y1 1 = y1 a;
428
                   pixel color = 1'b1;
429
                   x = x 1;
                   y = y_1;
430
431
               end
432
          end
433
434
435
      endmodule // DE1 SoC
436
437
      // Module to test the functionality of the DE1 SoC module
438
      module DE1 SoC testbench();
439
          logic [6:0] HEXO, HEX1, HEX2, HEX3, HEX4, HEX5;
440
          logic [9:0] LEDR;
441
          logic [3:0] KEY;
442
          logic [9:0] SW;
443
444
          logic clk;
445
          logic [7:0] VGA R;
446
          logic [7:0] VGA_G;
447
          logic [7:0] VGA B;
448
          logic VGA BLANK N;
449
          logic VGA CLK;
450
          logic VGA HS;
451
          logic VGA SYNC N;
452
          logic VGA VS;
453
          DE1 SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5
454
455
                            ,.LEDR, .KEY, .SW, .CLOCK 50 (clk)
456
                            ..VGA R, .VGA G, .VGA B, .VGA BLANK N
457
                            ,.VGA_CLK, .VGA_HS, .VGA_SYNC_N, .VGA_VS);
458
459
          parameter CLOCK PERIOD = 100;
460
          initial begin
461
               clk <= 0;
462
               forever #(CLOCK PERIOD / 2) clk <= ~clk;</pre>
463
          end
464
465
          initial begin
466
               integer i;
467
468
               @(posedge clk) SW[9] <= 1;</pre>
469
               @(posedge clk) SW[9] <= 0;</pre>
470
               @(posedge clk) SW[7] \le 0;
471
472
               for (i = 0; i < 50; i++) begin
473
                   @(posedge clk);
474
               end
475
476
               // Run through each of the arbitrary lines
477
               @ (posedge clk) SW[7] \le 1; SW[2:0] = 3'b000;
478
               for (i = 0; i < 10; i++) begin
479
                   @(posedge clk);
480
481
               @(posedge clk) SW[7] \le 1; SW[2:0] = 3'b001;
482
               for (i = 0; i < 10; i++) begin
483
                   @(posedge clk);
484
               end
               @(posedge clk) SW[7] \le 1; SW[2:0] = 3'b010;
485
486
               for (i = 0; i < 10; i++) begin
487
                   @(posedge clk);
```

```
488
              end
489
              @ (posedge clk) SW[7] \le 1; SW[2:0] = 3'b011;
490
              for (i = 0; i < 10; i++) begin
491
                  @(posedge clk);
492
493
              @(posedge clk) SW[7] \le 1; SW[2:0] = 3'b100;
494
              for (i = 0; i < 10; i++) begin
495
                  @(posedge clk);
496
              end
497
              @(posedge clk) SW[7] \leq 1; SW[2:0] = 3'b101;
498
              for (i = 0; i < 10; i++) begin
499
                  @(posedge clk);
500
              end
501
              @(posedge clk) SW[7] \le 1; SW[2:0] = 3'b110;
502
              for (i = 0; i < 10; i++) begin
503
                  @(posedge clk);
504
              end
505
              @(posedge clk) SW[7] \le 1; SW[2:0] = 3'b111;
506
              for (i = 0; i < 10; i++) begin
507
                  @(posedge clk);
508
              end
509
              // Clear the screen
510
              @(posedge clk) SW[7] <= 0; SW[8] <= 1;
511
              for (i = 0; i < 20; i++) begin
512
                  @(posedge clk);
513
              end
514
              // Run through the animation
515
              @(posedge clk) SW[8] <= 0;</pre>
516
              for (i = 0; i < 100; i++) begin
517
                  @(posedge clk);
518
              end
519
520
              $stop;
521
          end
522
523
      endmodule // DE1 SoC testbench
524
      /*********
525
526
527
      fill space.sv
528
      ************
529
530
531
     /*
532
      * Connor Aksama
533
      * 01/30/2023
534
      * CSE 371
      * Lab 3
535
536
      * /
537
538
539
      * Cycles through each pixel on a screen with the size of the parameterized dimensions.
540
      * Ex. (0,0) -> (1,0) -> (2,0) -> ... -> (638,479) -> (639, 479) -> (0,0)
      * Outputs a new pixel on each clock cycle.
541
542
543
      * Inputs:
544
         clk [1 bit] - the clock to use for this module
      ^{\star} reset [1 bit] - the reset signal for this module; resets to point (0,0)
545
546
547
      * Outputs:
         x [10 bit] - The x coordinate of the pixel to draw to
548
549
         y [9 bit] - The y coordinate of the pixel to draw to
550
      */
551
      module fill space #(
552
          parameter width = 640, height = 480
553
          )
554
          (
555
          output logic [9:0] x
556
          ,output logic [8:0] y
```

```
557
           ,input logic clk, reset
558
          );
559
560
          logic [9:0] x d, x q;
561
          logic [8:0] y_d, y_q;
562
563
          assign x = x q;
564
          assign y = y q;
565
566
          // Determine next coordinate
567
          always comb begin
568
               if (x q \ge width - 1 \&\& y q \ge height - 1) begin
569
                   // Reached x max and y max
570
                   x d = 0;
571
                   y_d = 0;
572
               end else if (x q >= width - 1) begin
573
                   // Reached x max
574
                   x d = 0;
575
                   y d = y q + 1;
576
               end else begin
577
                   // Reached y max
578
                   x d = x_q + 1;
579
                   y_d = y_q;
580
               end
581
          end
582
583
          // Register outputs
584
          always_ff @(posedge clk) begin
585
               if (reset) begin
586
                   x q \leq 0;
587
                   588
               end else begin
                   x q <= x d;
589
                   y_q <= y d;
590
591
               end
592
          end
593
594
      endmodule // fill space
595
596
      // Module to test the functionality of the fill space module
597
      module fill space testbench();
598
599
          logic [9:0] x;
600
          logic [8:0] y;
601
          logic clk, reset;
602
603
          fill space \#(4,5) dut (.*);
604
605
          parameter CLOCK PERIOD = 100;
606
          initial begin
607
               clk <= 0;
608
               forever #(CLOCK PERIOD / 2) clk <= ~clk;</pre>
609
          end
610
611
          initial begin
612
               integer i;
613
614
               @(posedge clk) reset <= 1;</pre>
615
               @(posedge clk) reset <= 0;</pre>
616
617
               // Run through each pixel in the screen
618
               for (i = 0; i < 50; i++) begin
619
                   @(posedge clk);
620
               end
621
622
               $stop;
623
          end
624
625
      endmodule // fill space testbench
```

```
626
627
      /**********
628
629
      line drawer.sv
630
     ***********
631
632
633
      * Connor Aksama
634
635
      * 01/30/2023
636
      * CSE 371
      * Lab 3
637
638
      * /
639
640
641
      * Given two endpoints (x0, y0), (x1, y1), outputs a sequence of points (x,y) that
      connect the two points.
642
      * Outputs at most one new point (x,y) every clock cycle starting on the cycle after the
       input endpoints change.
643
644
      * Inputs:
645
      * clk [1 bit] - the clock to use for this module
646
      * reset [1 bit] - the reset signal for this module; resets algorithm to first
      iteration given inputs
647
      * x0 [10 bit] - The x coordinate of the first point of the line to draw
         y0 [9 bit] - The y coordinate of the first point of the line to draw
648
         x1 [10 bit] - The x coordinate of the second point of the line to draw
649
650
      * y1 [9 bit] - The y coordinate of the second point of the line to draw
651
      * Outputs:
652
653
      * x [10 bit] - The x coordinate of the current pixel to draw
654
         y [9 bit] - The y coordinate of the current pixel to draw
     * /
655
     module line drawer(
656
657
          input logic clk, reset,
658
659
          // x and y coordinates for the start and end points of the line
660
          input logic [9:0] x0, x1,
661
          input logic [8:0] y0, y1,
662
663
          //outputs cooresponding to the coordinate pair (x, y)
664
          output logic [9:0] x,
665
          output logic [8:0] y
666
         );
667
668
          // Bresenham error
669
          logic signed [11:0] error, p error, n error;
670
671
          // Temp swap logic
672
          logic signed [9:0] x_min, x_max, y_start, y_end;
673
          logic signed [9:0] x0 r, x1 r, y0 r, y1 r;
674
          // Bresenham temp calculations
675
          logic is steep, is steep r;
676
          logic signed [10:0] delta x;
677
          logic signed [10:0] delta y;
678
          logic signed [9:0] step;
679
680
          // Registered output
681
          logic signed [9:0] px, nx;
682
          logic signed [8:0] py, ny;
683
          logic [9:0] prev_x0, prev_x1;
684
          logic [8:0] prev y0, prev y1;
685
686
          // Compute initial values before entering main loop
687
          always_comb begin
688
689
              delta x = x1 - x0;
690
             delta y = y1 - y0;
691
```

```
692
               is steep = (delta y < 0 ? -delta y : delta y) > (delta x < 0 ? -delta x : delta x
               );
693
694
               if (is_steep) begin
695
                    // Swap x, y
696
                    x0 r = y0;
697
                    y0 r = x0;
698
                    x1 r = y1;
699
                    y1 r = x1;
700
               end else begin
701
                    x0 r = x0;
702
                    x1 r = x1;
703
                    y0 r = y0;
704
                    y1 r = y1;
705
               end
706
707
               if (x0_r > x1_r) begin
708
                    // swap p0, p1
709
                    x \min = x1 r;
710
                    x max = x0 r;
711
                    y  start = y1  r;
712
                    y end = y0 r;
               end else begin
713
714
                    x_min = x0_r;
715
                    x max = x1 r;
716
                    y_start = y0_r;
                    y_end = y1 r;
717
718
               end
719
720
               delta x = x max - x min;
721
               delta y = (y \text{ end } - y \text{ start } < 0 ? - (y \text{ end } - y \text{ start}) : y \text{ end } - y \text{ start});
722
               error = -(delta \times / 2);
723
               if (y start < y end) step = 1;</pre>
724
725
               else
                                       step = -1;
726
727
               if (p_error + delta_y >= 0 && px < x_max) begin</pre>
728
                    // Step \mathbf{x} and \mathbf{y}, compute next error
729
                    ny = py + step;
730
                    nx = px + 1;
731
                    n error = p error + delta y - delta x;
732
               end else if (px < x max) begin</pre>
733
                    // Step x, compute next error
734
                    ny = py;
735
                    nx = px + 1;
736
                    n_error = p_error + delta_y;
737
               end else begin
738
                    // Reached end of line, keep outputting same point
739
                    nx = px;
740
                    ny = py;
741
                    n error = p error;
742
               end
743
744
               if (is steep r) begin
745
                    // Swap x,y
746
                    x = py;
747
                    y = px;
748
                end else begin
749
                    x = px;
750
                    y = py;
751
               end
752
753
           end
754
755
           // Update new values of current x, y in main loop
756
           always_ff @(posedge clk) begin
757
               prev_x0 <= x0;</pre>
758
               prev x1 <= x1;
759
               prev y0 <= y0;
```

```
760
               prev y1 <= y1;
761
               is steep r <= is steep;
762
763
               if (reset || prev x0 != x0 || prev y0 != y0 || prev x1 != x1 || prev y1 != y1)
               begin
764
                   // Reset initial values to input if input changes or reset
765
                   px <= x min;
766
                   py <= y start;</pre>
767
                   p error <= error;
768
               end else begin
769
                   // Output computed next values
770
                   px \le nx;
771
                   py <= ny;
772
                   p error <= n error;
773
               end
774
775
          end
776
777
      endmodule // line drawer
778
779
      // Module to test the functionality of the line drawer module
780
      module line drawer testbench();
781
782
          logic clk, reset;
783
784
          logic [9:0] x0, x1;
785
          logic [8:0] y0, y1;
786
787
          logic [9:0] x;
788
          logic [8:0] y;
789
790
          line drawer dut (.*);
791
792
          parameter CLOCK PERIOD = 100;
793
          initial begin
794
               clk <= 0;
795
               forever #(CLOCK PERIOD / 2) clk <= ~clk;</pre>
796
          end
797
798
          initial begin
799
               integer i;
800
801
               @ (posedge clk) reset \leq 1; x0 \leq 0; y0 \leq 0; x1 \leq 0; y1 \leq 0;
802
803
               // Horizontal
804
               @(posedge clk) reset \leq 0; x0 \leq 0; y0 \leq 0; x1 \leq 10; y1 \leq 0;
805
               for (i = 0; i < 20; i++) begin
806
                   @(posedge clk);
807
               end
808
809
               // Vertical
810
               @(posedge clk) x0 \le 0; y0 \le 0; x1 \le 0; y1 \le 10;
811
               for (i = 0; i < 20; i++) begin
812
                   @(posedge clk);
813
               end
814
815
               // Diagonal
816
               @(posedge clk) x0 \le 0; y0 \le 0; x1 \le 5; y1 \le 5;
817
               for (i = 0; i < 20; i++) begin
818
                   @(posedge clk);
819
               end
820
821
               // Shallow
822
               @(posedge clk) x0 \le 0; y0 \le 0; x1 \le 10; y1 \le 2;
823
               for (i = 0; i < 20; i++) begin
824
                   @(posedge clk);
825
               end
826
827
               // Steep
```

```
828
             @(posedge clk) x0 <= 0; y0 <= 0; x1 <= 2; y1 <= 10;
829
             for (i = 0; i < 20; i++) begin
830
                 @(posedge clk);
831
             end
832
833
             $stop;
         end
834
835
836
      endmodule // line drawer testbench
837
      /*********
838
839
840
      VGA framebuffer.sv
841
      ***********
842
843
844
      // VGA driver: provides I/O timing and double-buffering for the VGA port.
845
846
     module VGA framebuffer(
847
         input logic clk, rst,
848
          input logic [9:0] x, // The x coordinate to write to the buffer.
          input logic [8:0] y, // The y coordinate to write to the buffer.
849
         input logic pixel_color, pixel_write, // The data to write (color) and write-enable.
850
851
852
          input logic dfb en, // Double-Frame Buffer Enable
853
854
         output logic frame start, // Pulse is fired at the start of a frame.
855
856
         // Outputs to the VGA port.
857
         output logic [7:0] VGA R, VGA G, VGA B,
858
         output logic VGA CLK, VGA HS, VGA VS, VGA BLANK N, VGA SYNC N
859
     );
860
         /*
861
862
                            1279 1599 0
863
         * HCOUNT 1599 0
864
                                    | Video
865
                          Video
866
867
868
         * | SYNC | BP | <-- HACTIVE --> | FP | SYNC | BP | <-- HACTIVE
869
               VGA HS
870
871
872
         * /
873
874
         // Constants for VGA timing.
875
         localparam HPX = 11'd640*2, HFP = 11'd16*2, HSP = 11'd96*2, HBP = 11'd48*2;
876
         localparam VLN = 11'd480, VFP = 10'd11, VSP = 10'd2,
                                                                  VBP = 10'd31;
877
         localparam HTOTAL = HPX + HFP + HSP + HBP; // 800*2=1600
878
         localparam VTOTAL = VLN + VFP + VSP + VBP; // 524
879
880
         // Horizontal counter.
881
         logic [10:0] h count;
882
         logic end of line;
883
884
         assign end of line = h count == HTOTAL - 1;
885
886
          always ff @(posedge clk)
887
             if (rst) h_count <= 0;</pre>
888
             else if (end_of_line) h_count <= 0;</pre>
889
             else h count <= h count + 11'd1;</pre>
890
891
         // Vertical counter & buffer swapping.
892
         logic [9:0] v count;
893
         logic end of field;
         logic front_odd; // whether odd address is the front buffer.
894
895
896
         assign end of field = v count == VTOTAL - 1;
```

```
897
          assign frame start = !h count && !v count;
898
899
          always ff @(posedge clk)
900
              if (rst) begin
901
                   v count <= 0;
902
                   front_odd <= 0;</pre>
903
               end else if (end of line)
904
                   if (end of field) begin
905
                       v count <= 0;</pre>
906
                       front odd <= !front odd;</pre>
907
                   end else
908
                       v count <= v count + 10'd1;</pre>
909
910
          // Sync signals.
911
          assign VGA CLK = h count[0]; // 25 MHz clock: pixel latched on rising edge.
912
          assign VGA_HS = !(h_count - (HPX + HFP) < HSP);</pre>
913
          assign VGA_VS = !(v_count - (VLN + VFP) < VSP);</pre>
914
          assign VGA SYNC N = 1; // Unused by VGA
915
916
          // Blank area signal.
917
          logic blank;
918
          assign blank = h count >= HPX || v count >= VLN;
919
920
          // Double-buffering.
921
          logic buffer[640*480*2-1:0];
922
          logic [19:0] wr addr, rd addr;
923
          logic rd data;
924
925
          assign wr addr = \{y * 19'd640 + x, (!front odd & dfb en)\};
926
          assign rd addr = {v \text{ count } * 19'd640 + (h \text{ count } / 19'd2), (front odd & dfb en)};
927
928
          always ff @(posedge clk) begin
929
              if (pixel write) buffer[wr addr] <= pixel color;</pre>
930
               if (VGA CLK) begin
931
                   rd data <= buffer[rd addr];</pre>
932
                   VGA BLANK N <= ~blank;
933
               end
934
          end
935
936
          // Color output.
937
          assign {VGA R, VGA G, VGA B} = rd data ? 24'hFFFFFF : 24'h000000;
938
      endmodule
939
```