

```

1  module vga_demo (
2      input      CLOCK_50,           // 50 MHz
3      input [17:0] SW,
4      input [3:0]  KEY,
5      output [17:0] LEDR,
6      output [7:0]  LEDG,
7      output [6:0]  HEX0,
8      output [6:0]  HEX1,
9      output [6:0]  HEX2,
10     output [6:0]  HEX3,
11     output [6:0]  HEX4,
12     output [6:0]  HEX5,
13     output [6:0]  HEX6,
14     output [6:0]  HEX7,
15     output      VGA_CLK,           // VGA Cl ock
16     output      VGA_HS,           // VGA H_SYNC
17     output      VGA_VS,           // VGA V_SYNC
18     output      VGA_BLANK,        // VGA BLANK
19     output      VGA_SYNC,        // VGA SYNC
20     output [9:0]  VGA_R,           // VGA Red[9:0]
21     output [9:0]  VGA_G,           // VGA Green[9:0]
22     output [9:0]  VGA_B           // VGA Bl ue[9:0]
23 );
24
25 assign LEDR = SW;
26 assign LEDG = 0;
27
28 //Turns off HEX1 and HEX0, which aren't used
29 assign HEX1 = 7'h7f, HEX0 = 7'h7f;
30
31 //hexdi git modules simply assigned to display what values they represent
32 hexdi git mode (
33     .in  ({2'b00, SW[16:15]}),
34     .out  (HEX7)
35 );
36
37
38 hexdi git speed (
39     .in  ({2'b0, SW[14:13]}),
40     .out  (HEX3)
41 );
42
43 hexdi git red (
44     .in  ({2'b0, SW[6:5]}),
45     .out  (HEX6)
46 );
47
48 hexdi git green (
49     .in  ({2'b00, SW[4:3]}),
50     .out  (HEX5)
51 );
52
53 hexdi git bl ue (
54     .in  ({2'b00, SW[2:1]}),
55     .out  (HEX4)
56 );
57
58
59 hexdi git si ze (
60     .in  ({2'b00, SW[12:11]}),
61     .out  (HEX2)
62 );
63
64
65 wire w3;
66 wire w4;
67 wire w5;
68 wire w6;
69
70 //Four counters, whose enables are KEYS[0:3]
71 //Depending on the value of s, f will output a
72 //1 once the counter reaches a certain number. This
73 //is done to allow the user to control the cursor at
74 //a reasonable speed. Otherwise, the cursor would move
75 //across the screen 50 million times a second.
76

```

```

77
78 //KEY0 - cursor right
79 //KEY1 - cursor left
80 //KEY2 - cursor down
81 //KEY3 - cursor up
82
83 counter c0 (
84     .clk (CLOCK_50),
85     .en (~KEY[0]),
86     .s (SW[14:13]),
87     .f (w3)
88 );
89
90 counter c1 (
91     .clk (CLOCK_50),
92     .en (~KEY[1]),
93     .s (SW[14:13]),
94     .f (w4)
95 );
96
97 counter c2 (
98     .clk (CLOCK_50),
99     .en (~KEY[2]),
100    .s (SW[14:13]),
101    .f (w5)
102 );
103
104 counter c3 (
105     .clk (CLOCK_50),
106     .en (~KEY[3]),
107     .s (SW[14:13]),
108     .f (w6)
109 );
110
111
112 wire [7:0] xLoc;
113 wire [7:0] yLoc;
114
115 //Screen Resolution is 160 x 120, so max for
116 //x is 160, max for y is 120
117
118 buttonLogic bX (
119     .clk (CLOCK_50),
120     .max (160),
121     .addsignal (w3),
122     .subsignal (w4),
123     .k (xLoc[7:0])
124 );
125
126 buttonLogic bY (
127     .clk (CLOCK_50),
128     .max (120),
129     .addsignal (w5),
130     .subsignal (w6),
131     .k (yLoc[7:0])
132 );
133
134 wire w7;
135 wire [5:0] cw;
136
137 counter c4 (
138     .clk (CLOCK_50),
139     .en (1),
140     .s (SW[14:13]),
141     .f (w7)
142 );
143
144
145 //Each of these cycle color modules have a six bit
146 //output, which corresponds to the 6 bits of color data.
147 //This 6 bit output changes with the clock, so the color
148 //constantly changes. The exact specifics of the inputs
149 //to these cycle color modules were simply tested to find
150 //colors that looked appealing.
151
152 cycleColor cC0 (

```

```

153     .clk (CLOCK_50),
154     .max (6'b111111),
155     .changeSig(w7),
156     .col (cw)
157 );
158
159
160 wire [3:0] cw1;
161 cycleColor4 cC1 (
162     .clk (CLOCK_50),
163     .max (4'b1111),
164     .changeSig(w7),
165     .col (cw1)
166 );
167
168 wire [3:0] cw2;
169 cycleColor4 cC2 (
170     .clk (CLOCK_50),
171     .max (4'b1111),
172     .changeSig(w7),
173     .col (cw2)
174 );
175
176 wire [7:0] cr;
177
178 //Decides which color should be displayed on the screen.
179 //Either the color determined by the switches, or the color
180 //determined by one of the cycle color modules
181
182 muxX4 m0 (
183     .a ({2'b00, SW[6:1]}),
184     .b ({2'b00, cw}),
185     .c ({4'b0011, cw1}),
186     .d ({2'b00, cw2[3:2], 2'b11, cw2[1:0]}),
187     .s (SW[16:15]),
188     .f (cr)
189 );
190
191 wire scw;
192
193
194 //This counter is essentially used as a clock, which
195 //slightly offset from the internal CLOCK50.
196 Scounter sc0 (
197     .clk (CLOCK_50),
198     .f (scw)
199 );
200
201
202 wire [7:0] ox1w;
203 wire [7:0] ox2w;
204 wire [7:0] ox3w;
205
206 wire [7:0] oy1w;
207 wire [7:0] oy2w;
208 wire [7:0] oy3w;
209
210
211 //Oscillators are used to change the size of what is
212 //being drawn. For example, if the user's cursor is
213 //2x2, the program must plot four pixels total. So the
214 //xLocation oscillates between its original position and the
215 //position one to the left, and the yLocation osciallates between
216 //its original position and the position one below it. Since the
217 //clock is 50MHZ, even though all these pixels are not plotted at
218 //the same time, to the user, their cursor appears to be the desired
219 //size.
220
221 OscillatorX1 ox1(
222     .clk (CLOCK_50),
223     .x (xLoc),
224     .f (ox1w)
225 );
226
227 OscillatorX2 ox2(
228     .clk (CLOCK_50),

```

```

229     . x (xLoc),
230     . f (ox2w)
231 );
232
233 OscillatorX ox3(
234     . clk (CLOCK_50),
235     . x (xLoc),
236     . f (ox3w)
237 );
238
239 OscillatorX1 oy1(
240     . clk (scw),
241     . x ({1'b0, yLoc[6:0]}),
242     . f (oy1w)
243 );
244
245 OscillatorX2 oy2(
246     . clk (scw),
247     . x ({1'b0, yLoc[6:0]}),
248     . f (oy2w)
249 );
250
251
252 OscillatorX oy3(
253     . clk (scw),
254     . x ({1'b0, yLoc[6:0]}),
255     . f (oy3w)
256 );
257
258
259 wire [7:0] fxLoc;
260 wire [7:0] fyLoc;
261
262
263 //Muxes to select the size of the cursor based
264 //on the values of SW[12:11]
265
266 muxX4 mX4(
267     . a (xLoc),
268     . b (ox1w),
269     . c (ox2w),
270     . d (ox3w),
271     . s (SW[12:11]),
272     . f (fxLoc),
273 );
274
275 muxX4 mY4(
276     . a ({1'b0, yLoc[6:0]}),
277     . b (oy1w),
278     . c (oy2w),
279     . d (oy3w),
280     . s (SW[12:11]),
281     . f (fyLoc),
282 );
283
284 wire [7:0] finalX;
285 wire [7:0] finalY;
286 wire [7:0] finalColor;
287
288 //These final four muxes are simply used for the erase feature.
289 //The erase feature is toggled with SW[17]. When the feature
290 //is toggled on, plot is set to 1, the color is set to black, and
291 //the location sweeps across the screen. In this way, the screen is
292 //completely plotted black, which erases the screen.
293
294
295 //Also, for these last four muxes, inputs c and d can never be output,
296 //since the s input can only be 00 or 01.
297
298 muxX4 fcm (
299     . a (cr),
300     . b (8'b00000000),
301     . c (8'b00000000),
302     . d (8'b00000000),
303     . s ({1'b0, SW[17]}),
304     . f (finalColor)

```

```

305 );
306
307 reg zX = 8'b00000000;
308 reg zY = 7'b0000000;
309
310 wire [7:0] runni ngX;
311 wire [7:0] runni ngY;
312
313 wire modY;
314
315 //These RunX modules are used to determine the location
316 //of the cursor when erase mode is toggled on. rX sweeps
317 //the cursor from left to right across the screen, and its
318 //output next is 1 when it reaches the very right of the screen.
319 //rY uses this next output as its clock. So when the cursor reaches
320 //the very right of the screen, the yLocation is incremented by 1.
321 //In this way, every pixel of the screen is plotted black. Because
322 //of the speed of the 50MHZ clock, the user does not notice this sweeping
323 //across the screen, and simply sees the entire screen immediately erased.
324
325 RunX rX (
326     .clk (CLOCK_50),
327     .max (8'b11111111),
328     .next (modY),
329     .f (runni ngX)
330 );
331
332 wire empty;
333
334 RunX rY (
335     .clk (modY),
336     .max (8'b01111111),
337     .next (empty),
338     .f (runni ngY)
339 );
340
341 muxX4 fxm (
342     .a (fxLoc),
343     .b (runni ngX),
344     .c (8'b00000000),
345     .d (8'b00000000),
346     .s ({1'b0, SW[17]}),
347     .f (fi nal X)
348 );
349
350 muxX4 fym (
351     .a (fyLoc),
352     .b (runni ngY),
353     .c (8'b00000000),
354     .d (8'b00000000),
355     .s ({1'b0, SW[17]}),
356     .f (fi nal Y)
357 );
358
359 wire [7:0] fi nal Pl ot;
360
361
362 //This mux ensures that plot is set to 1 when erase mode
363 //is on. If plot was off, then no black would be plotted
364 //and the erase feature would not work.
365 muxX4 m1x0 (
366     .a ({7'b0000000, SW[0]}),
367     .b (8'b00000001),
368     .c (8'b00000000),
369     .d (8'b00000000),
370     .s ({1'b0, SW[17]}),
371     .f (fi nal Pl ot)
372 );
373
374 //The vga_adapter is sent all the bits its should have based on the user input.
375
376 vga_adapter VGA(
377     .resetn (1),
378     .clock (CLOCK_50),
379     .col our (fi nal Col or [5:0]),
380     .x (fi nal X),

```

```
381         .y                (fi nal Y[6: 0]),
382         .pl ot             (fi nal Pl ot [0]),
383         .VGA_R              (VGA_R),
384         .VGA_G              (VGA_G),
385         .VGA_B              (VGA_B),
386         .VGA_HS             (VGA_HS),
387         .VGA_VS             (VGA_VS),
388         .VGA_BLANK          (VGA_BLANK),
389         .VGA_SYNC           (VGA_SYNC),
390         .VGA_CLK            (VGA_CLK)
391     );
392     defparam VGA. RESOLUTION = "160x120";
393     defparam VGA. MONOCHROME = "FALSE";
394     defparam VGA. BI TS_PER_COLOUR_CHANNEL = 2;
395
396 endmodul e
397
```