```
1   //Creators: Hannah Brooks, Wasif Butt
2   //if you steal this ill be very upset
3
4   module project
5      (
6         CLOCK_50,                  // On Board 50 MHz
7         SW,
8         KEY,                       // On Board Keys
9         LEDR,
10        VGA_CLK,                   // VGA Clock
11        VGA_HS,                    // VGA H_SYNC
12        VGA_VS,                    // VGA V_SYNC
13        VGA_BLANK_N,               // VGA BLANK
14        VGA_SYNC_N,                // VGA SYNC
15        VGA_R,                     // VGA Red[9:0]
16        VGA_G,                     // VGA Green[9:0]
17        VGA_B,                     // VGA Blue[9:0]
18        PS2_CLK,
19        PS2_DAT,
20        AUD_ADCDAT,
21
22        // Bidirectionals
23        AUD_BCLK,
24        AUD_ADCLRCK,
25        AUD_DACLRCK,
26
27        FPGA_I2C_SDAT,
28
29        // Outputs
30        AUD_XCK,
31        AUD_DACDAT,
32
33        FPGA_I2C_SCLK,
34        HEX0,
35        HEX1,
36        HEX2,
37        HEX3
38     );
39
40     /*****************************************************************/
41     // keyboard wires
42     wire  [7:0] the_command;
43     wire        send_command;
44
45     inout       PS2_CLK;
46     inout       PS2_DAT;
47
48     wire        command_was_sent;
49     wire        error_communication_timed_out;
50
51     wire  [7:0] received_data;
52     wire        received_data_en;
53     /*****************************************************************/
54     //VGA stuff
55     input       CLOCK_50;          // 50 MHz
56     input [3:0] KEY;
57     input [9:0] SW;
58     output [9:0] LEDR;
59
60     output         VGA_CLK;            // VGA Clock
61     output         VGA_HS;             // VGA H_SYNC
62     output         VGA_VS;             // VGA V_SYNC
63     output         VGA_BLANK_N;        // VGA BLANK
64     output         VGA_SYNC_N;         // VGA SYNC
65     output   [7:0] VGA_R;              // VGA Red[7:0] Changed from 10 to 8-bit DAC
66     output   [7:0] VGA_G;              // VGA Green[7:0]
67     output   [7:0] VGA_B;              // VGA Blue[7:0]
68     output   [6:0] HEX0;
69     output   [6:0] HEX1;
70     output   [6:0] HEX2;
71     output   [6:0] HEX3;
72
73
```

```
74      vga_adapter VGA(
75            .resetn(resetn),
76            .clock(CLOCK_50),
77            .colour(colour),
78            .x(x),
79            .y(y),
80            .plot(writeEn),
81            /* Signals for the DAC to drive the monitor. */
82            .VGA_R(VGA_R),
83            .VGA_G(VGA_G),
84            .VGA_B(VGA_B),
85            .VGA_HS(VGA_HS),
86            .VGA_VS(VGA_VS),
87            .VGA_BLANK(VGA_BLANK_N),
88            .VGA_SYNC(VGA_SYNC_N),
89            .VGA_CLK(VGA_CLK));
90         defparam VGA.RESOLUTION = "160x120";
91         defparam VGA.MONOCHROME = "FALSE";
92         defparam VGA.BITS_PER_COLOUR_CHANNEL = 4;
93         defparam VGA.BACKGROUND_IMAGE = "black.mif";
94
95
96
    //*****************************************************************************************
    *********
97
98      wire outofBounds, ground;
99
100     //keys using keyboard
101     reg keyR, keyL, keyUp, keyDown, keySpace;
102     wire right, left, up, down, spaecbar;
103
104     assign right = keyR;
105     assign left = keyL;
106     assign up = keyUp;
107     assign down = keyDown;
108     assign spacebar = keySpace;
109
110     PS2_Controller ps2(
111        // Inputs
112        CLOCK_50,
113
114        the_command,
115        send_command,
116
117        // Bidirectionals
118        PS2_CLK,              // PS2 Clock
119        PS2_DAT,              // PS2 Data
120
121        // Outputs
122        command_was_sent,
123        error_communication_timed_out,
124
125        received_data,
126        received_data_en        // If 1 - new data has been received
127     );
128     reg moving;
129
130     always @(posedge received_data_en) begin
131        if ((received_data == 8'b1111_0000) && moving == 1) begin
132           keySpace <= 0;
133           keyL <= 0;
134           keyR <= 0;
135           keyUp <= 0;
136           keyDown <= 0;
137           moving <= 0;
138        end
139        if (moving == 0)
140           moving <= 1;
141        else if ((received_data == 7'b0101001) && start)
142           keySpace <= 1;
143        else if ((received_data == 7'b1110010) && moving == 1 && (lvl1 || lvl2))
144           keyDown <= 1;
```

```
145         else if ((received_data == 7'b1101011) && moving == 1 && (lvl1 || lvl2 || lvl3))
146            keyL <= 1;
147         else if ((received_data == 7'b1110100) && moving == 1 && (lvl1 || lvl2 || lvl3))
148            keyR <= 1;
149         else if ((received_data == 7'b1110101) && moving == 1 && (lvl1 || lvl2 || lvl3))
150            keyUp <= 1;
151      end
152
153
154      //VGA input controlling wires
155      reg [11:0] colour;
156      reg [7:0] x, y;
157      reg writeEn;
158      wire resetn;
159      assign resetn = KEY[3];
160
161      //draw controls
162      wire resetAddress, resetAddress2, resetAddress3, fall;
163      wire drM, drML, jump, jumpL, drStage1; //draw controls level 1
164      wire drM2, drML2, jump2, jumpL2, drStage2; //draw controls level 2
165      wire drM3, drML3, jump3, jumpL3, drStage3; //draw controls level 3
166      wire erM, jumping, falling, pipe, dead; //erase controls level 1
167      wire erM2, jumping2, falling2, next; //erase controls level 2
168      wire erM3, jumping3, falling3, flag; //erase controls level
169      wire [11:0] marioColourRight, marioColourLeft, jumpColourRight, jumpColourLeft,
    stage1Colour, stage2Colour, stage3Colour, startColour; //colours
170      wire [7:0] Mariox, Marioy, Jumpx, Jumpy, lvl1bkgx, lvl1bkgy; //x y controls for drawing
171      wire moveRight, moveLeft, moveUp; //draw in direction ********TO BE DELETED********
172      wire [7:0] px, py; //position tracker for mario
173      wire [4:0] jumpCounter; //jumping position tracker
174      wire startEnable, lvl1Enable, lvl2Enable, lvl3Enable; //write enables
175
176      //overall FSM controls
177      wire start, draw, lvl1, lvl2;
178
179      //counters
180      wire done;
181      reg [14:0] address;
182      wire [14:0] outAddress;
183
184      //rate divider enable
185      wire go, goJump;
186      wire timer_enable;
187
188      //accessing memories to draw images
189      mario drawMarioRight(address, CLOCK_50, marioColourRight);
190      marioLeft drawMarioLeft(address, CLOCK_50, marioColourLeft);
191      jump jumpMarioRight(address, CLOCK_50, jumpColourRight);
192      jumpLeft jumpMarioLeft(address, CLOCK_50, jumpColourLeft);
193      start_screen startscreen(address, CLOCK_50, startColour);
194      stage1_bkg stage1(address, CLOCK_50, stage1Colour);
195      stage2_bkg stage2(address, CLOCK_50, stage2Colour);
196      stage3_bkg stage3(address, CLOCK_50, stage3Colour);
197
198      always @(*) begin
199         if (start) begin
200            writeEn <= startEnable;
201            if(draw) begin
202               colour <= startColour;
203               x <= lvl1bkgx;
204               y <= lvl1bkgy;
205               address <= outAddress;
206            end
207            else begin
208               address <= 0;
209               x <= 0;
210               y <= 0;
211            end
212         end
213         else if (lvl1) begin
214            writeEn <= lvl1Enable;
215            if (resetAddress)
216               address <= 0;
```

```
217             else
218                address <= outAddress;
219
220             if (drStage1 || erM || jumping || falling)  begin
221                colour <= stage1Colour;
222                x <= lvl1bkgx;
223                y <= lvl1bkgy;
224             end
225
226             else if(drM || drML || jump || jumpL) begin
227                if (jump) begin
228                   colour <= jumpColourRight;
229                   address <= outAddress;
230                end
231                else if (jumpL) begin
232                   colour <= jumpColourLeft;
233                   address <= outAddress;
234                end
235                else if (drM) begin
236                   colour <= marioColourRight;
237                   address <= outAddress;
238                end
239                else begin
240                   colour <= marioColourLeft;
241                   address <= outAddress;
242                end
243
244                x <= Mariox;
245                y <= Marioy;
246             end
247          end
248          else if (lvl2) begin
249             writeEn <= lvl2Enable;
250             if (resetAddress2)
251                address <= 0;
252             else
253                address <= outAddress;
254
255             if (drStage2 || erM2 || jumping2 || falling2)  begin
256                colour <= stage2Colour;
257                x <= lvl1bkgx;
258                y <= lvl1bkgy;
259             end
260
261             else if(drM2 || drML2 || jump2 || jumpL2) begin
262                if (marioColourRight == 12'b000010101110 && drM2) begin
263                   address <= outAddress;
264                   colour <= 12'b000000000000;
265                end
266                else if (marioColourLeft == 12'b000010101110 && drML2) begin
267                   address <= outAddress;
268                   colour <= 12'b000000000000;
269                end
270                else if (jumpColourRight == 12'b000010101110 && jump2) begin
271                   address <= outAddress;
272                   colour <= 12'b000000000000;
273                end
274                else if (jumpColourLeft == 12'b000010101110 && jumpL2) begin
275                   address <= outAddress;
276                   colour <= 12'b000000000000;
277                end
278                else if (jump2) begin
279                   colour <= jumpColourRight;
280                   address <= outAddress;
281                end
282                else if (jumpL2) begin
283                   colour <= jumpColourLeft;
284                   address <= outAddress;
285                end
286                else if (drM2) begin
287                   colour <= marioColourRight;
288                   address <= outAddress;
289                end
```

```
290                 else begin
291                     colour <= marioColourLeft;
292                     address <= outAddress;
293                 end
294 
295                 x <= Mariox;
296                 y <= Marioy;
297             end
298         end
299         else if (lvl3) begin
300             writeEn <= lvl3Enable;
301             if (resetAddress3)
302                 address <= 0;
303             else
304                 address <= outAddress;
305 
306             if (drStage3 || erM3 || jumping3 || falling3)  begin
307                 colour <= stage3Colour;
308                 x <= lvl1bkgx;
309                 y <= lvl1bkgy;
310             end
311 
312             else if(drM3 || drML3 || jump3 || jumpL3) begin
313                 if (flag && ((Mariox == 8'd105 && Marioy == 8'd70) || (Mariox == 8'd106 &&
    Marioy == 8'd70) || (Mariox == 8'd105 && Marioy == 8'd71) || (Mariox == 8'd106 && Marioy ==
    8'd71))) begin
314                     address <= outAddress;
315                     colour <= 12'b101111100001;
316                 end
317                 else if (jump3) begin
318                     colour <= jumpColourRight;
319                     address <= outAddress;
320                 end
321                 else if (jumpL3) begin
322                     colour <= jumpColourLeft;
323                     address <= outAddress;
324                 end
325                 else if (drM3) begin
326                     colour <= marioColourRight;
327                     address <= outAddress;
328                 end
329                 else begin
330                     colour <= marioColourLeft;
331                     address <= outAddress;
332                 end
333 
334                 x <= Mariox;
335                 y <= Marioy;
336             end
337         end
338     end
339 
340     //FSMs
341     overallFSM overall(CLOCK_50, resetn, spacebar, pipe, next, flag, done, start, draw, lvl1,
     lvl2, lvl3, startEnable, dead, timer_enable);
342 
343     lvl1FSM level1(CLOCK_50, resetn, go, goJump, right, left, up, down,
344                     resetAddress, drM, drML, erM, jump, jumpL, jumping, falling, drStage1,
    moveRight,
345                     done, jumpCounter, fall,
346                     lvl1Enable, rightColour, leftColour, ground, outofBounds, pipe, lvl1,
    start, dead);
347 
348     lvl2FSM level2(CLOCK_50, resetn, go, goJump, right, left, up, down,
349                     resetAddress2, drM2, drML2, erM2, jump2, jumpL2, jumping2, falling2,
    drStage2, moveRight,
350                     done, jumpCounter, fall,
351                     lvl2Enable, rightColour, leftColour, ground, outofBounds, next, lvl2,
    start);
352 
353     lvl3FSM level3(CLOCK_50, resetn, go, goJump, right, left, up, down,
354                     resetAddress3, drM3, drML3, erM3, jump3, jumpL3, jumping3, falling3,
    drStage3, moveRight,
```

```
355                     done, jumpCounter, fall,
356                     lvl3Enable, rightColour, leftColour, ground, outofBounds, flag, lvl3,
    start);
357
358     //datapath
359     drawStuff drawObjects(CLOCK_50, drM, drML, erM, jump, jumpL, jumping, falling, drStage1,
360                         drM2, drML2, erM2, jump2, jumpL2, jumping2, falling2, drStage2,
361                         drM3, drML3, erM3, jump3, jumpL3, jumping3, falling3, drStage3,
362                         done, jumpCounter, outAddress, fall,
363                         Mariox, Marioy, Jumpx, Jumpy, lvl1bkgx, lvl1bkgy, px, py, draw);
364
365     //movement registers
366     marioReg marioMovement(CLOCK_50, go, ground, px, py,
367                         erM, jump, jumpL, jumpCounter, jumping, falling, drStage1, drStage2,
    drStage3,
368                         right, left, up, down, outofBounds, pipe, next, flag, start, lvl1, lvl2
    , lvl3, dead);
369
370
371     //refresh rate divider
372     rateDivider moverate(go, CLOCK_50);
373     MarioJumpRateDivider jumprate(goJump, CLOCK_50);
374
375     //clock
376     timer timer(HEX0, HEX1, HEX2, HEX3, CLOCK_50, timer_enable);
377
378     input           AUD_ADCDAT;
379
380     inout           AUD_BCLK;
381     inout           AUD_ADCLRCK;
382     inout           AUD_DACLRCK;
383
384     inout           FPGA_I2C_SDAT;
385
386     output             AUD_XCK;
387     output             AUD_DACDAT;
388     output             FPGA_I2C_SCLK;
389
390     wire            audio_in_available;
391     wire     [31:0]    left_channel_audio_in;
392     wire     [31:0]    right_channel_audio_in;
393     wire            read_audio_in;
394
395     wire            audio_out_allowed;
396     wire     [31:0]    left_channel_audio_out;
397     wire     [31:0]    right_channel_audio_out;
398     wire            write_audio_out;
399     wire     [7:0] data_received;
400
401     reg [18:0] delay_cnt;
402     wire [18:0] delay;
403
404     reg snd;
405
406     reg [22:0] beatCountMario;
407     reg [9:0] addressMario;
408
409     sound r1(.address(addressMario), .clock(CLOCK_50), .q(delay));
410
411     always @(posedge CLOCK_50)
412        if(delay_cnt == delay) begin
413           delay_cnt <= 0;
414           snd <= !snd;
415        end else delay_cnt <= delay_cnt + 1;
416
417     always @(posedge CLOCK_50) begin
418        if(beatCountMario == 23'b10011000100101101000000)begin
419           beatCountMario <= 23'b0;
420           if(addressMario < 10'd999)
421              addressMario <= addressMario + 1;
422           else begin
423              addressMario <= 0;
424              beatCountMario <= 0;
```

```
425             end
426          end
427          else
428             beatCountMario <= beatCountMario + 1;
429       end
430
431       wire [31:0] sound = snd ? 32'd100000000 : -32'd100000000;
432
433       assign read_audio_in       = audio_in_available & audio_out_allowed;
434       assign left_channel_audio_out = left_channel_audio_in+sound;
435       assign right_channel_audio_out   = left_channel_audio_in+sound;
436       assign write_audio_out        = audio_in_available & audio_out_allowed;
437
438       Audio_Controller Audio_Controller (
439          // Inputs
440          .CLOCK_50                  (CLOCK_50),
441          .reset                     (~KEY[0]),
442
443          .clear_audio_in_memory     (),
444          .read_audio_in             (read_audio_in),
445
446          .clear_audio_out_memory    (),
447          .left_channel_audio_out    (left_channel_audio_out),
448          .right_channel_audio_out   (right_channel_audio_out),
449          .write_audio_out           (write_audio_out),
450
451          .AUD_ADCDAT                (AUD_ADCDAT),
452
453          // Bidirectionals
454          .AUD_BCLK                  (AUD_BCLK),
455          .AUD_ADCLRCK               (AUD_ADCLRCK),
456          .AUD_DACLRCK               (AUD_DACLRCK),
457
458
459          // Outputs
460          .audio_in_available        (audio_in_available),
461          .left_channel_audio_in     (left_channel_audio_in),
462          .right_channel_audio_in    (right_channel_audio_in),
463
464          .audio_out_allowed         (audio_out_allowed),
465
466          .AUD_XCK                (AUD_XCK),
467          .AUD_DACDAT                (AUD_DACDAT)
468
469       );
470
471       avconf #(.USE_MIC_INPUT(1)) avc (
472          .FPGA_I2C_SCLK             (FPGA_I2C_SCLK),
473          .FPGA_I2C_SDAT             (FPGA_I2C_SDAT),
474          .CLOCK_50                  (CLOCK_50),
475          .reset                     (~KEY[0])
476       );
477
478
479    endmodule
480
481
482
483
```