

Final Project Report (Appendix):
Wireless Musical Electrocardiogram

Amy Tang and Sinit Vitavasiri
(Group 12)

TA: Hyunjoo Jenny Lee
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 – Introductory Digital Systems Laboratory

Appendix A: Switch Assignments

switch[4]	switch[3]	switch[2]	switch[1]	switch[0]	Function
X	X	X	X	Up	Enable EKG input (Testing Mode)
X	X	X	X	Down	Disable EKG input (Testing Mode)
X	X	X	Up	X	Enable Overall System
X	X	X	Down	X	Disable Overall System
X	Down	Down	Up	X	Disable Heart Rhythm Sound and Abnormality Detection Alarm*
X	Down	Up	Up	X	Enable Abnormality Detection Alarm*
X	Up	Down	Up	X	Enable Heart Rhythm Sound*
X	Up	Up	Up	X	Enable Heart Rhythm Sound and Abnormality Detection Alarm*
Up	X	X	Up	X	Enable Classical Music* (Heart-Rate-Controlled Music Mode)

* The volume can be adjusted over the range of 0-15 using the up and down pushbuttons.

switch[7]	Function
Up	Heart-Rate Display
Down	Volume Display

switch[5]	Function
Up	Enable Wireless Transmission
Down	Disable Wireless Transmission

Appendix B: Tempo-Changing Algorithm (Matlab Code)

```
function output = timescale(sig, compression, maxfreq)
% takes in a signal in the time domain and scales its length,
% thus increasing its tempo. It scales the signal by compression,
% where compression is less than 1.
% It takes in maxfreq in order to compute how often to remove samples.

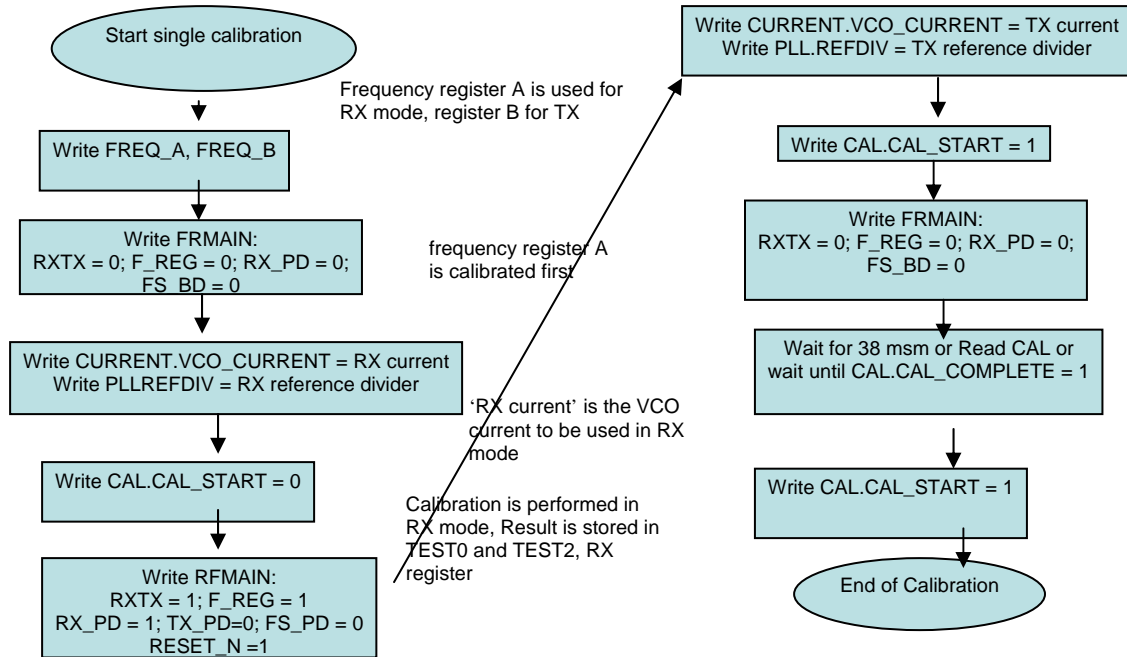
if nargin < 3, maxfreq = 4096; end
n = length(sig);

% Computes how often to remove samples
Timediv = floor(.08*maxfreq*2)

% Computes how many samples to remove
remove = floor((1-compression)*timediv)
output = 0;

% Remove samples, and recombine signals
for i = remove+1:(timediv+remove):(n-timediv)
    output = [output; sig((i-remove):(i+timediv-remove))];
end
```

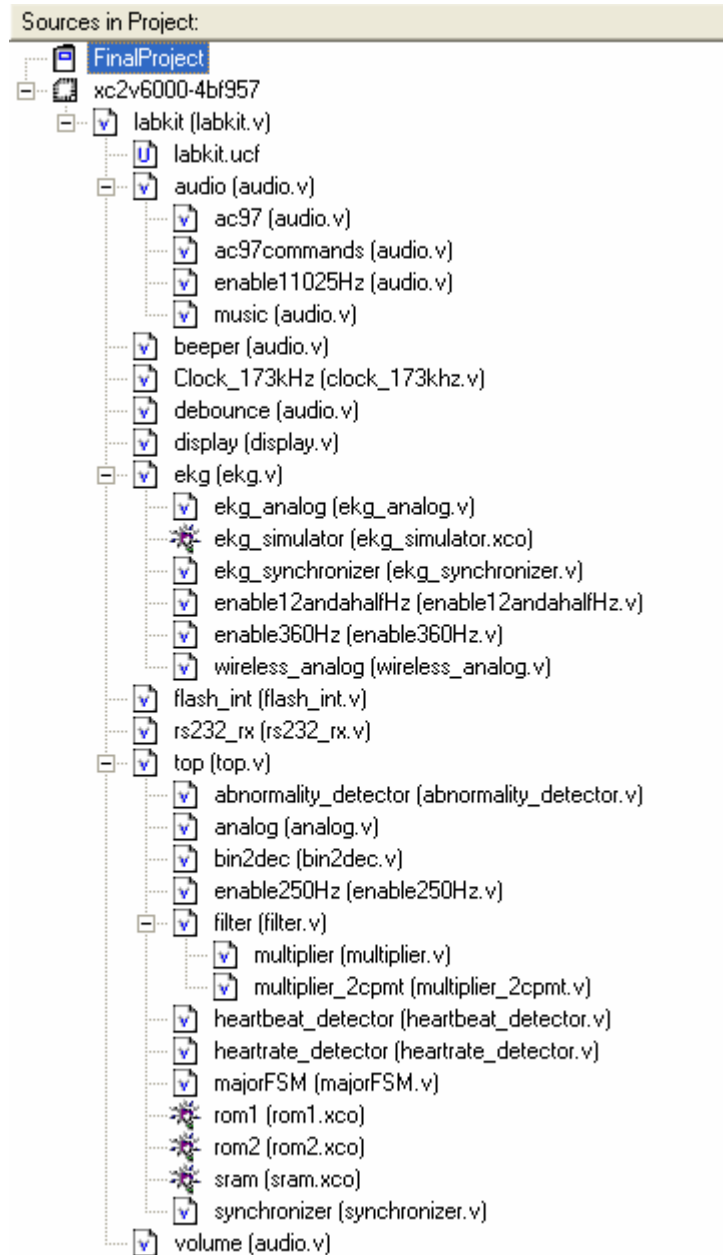
Appendix C: Wireless Calibration Algorithm



Control Flow of a Calibration Instantiation of a Transceiver (Transmittance and Receiving) System

Appendix D: Verilog/C Codes

All the codes are sorted alphabetically. Below is the hierarchy of the overall system.



There are other five files that are used to program the Flash ROM, namely: *flashtest*, *romsound*, *test_fsm_no_erase*, *test_fsm_old*, and *test_fsm_read_ROM*. The C codes used to implement the wireless system can be found in *wireless1*, *wireless2*, and *wireless3*.

```
1 module abnormality_detector(clk, reset_sync, heartrate, abnormal);
2
3 input clk, reset_sync;
4 input[8:0] heartrate;
5 output abnormal;
6
7 wire abnormal;
8 reg[21:0] count;
9
10 always @ (posedge clk)
11 begin
12     count <= count + 1;
13 end
14
15 assign abnormal = (heartrate <= 9'd50 || (heartrate >= 9'd200 && heartrate != 9'b111111111)) ? count[21] : 1'b0;
16
17 endmodule
```

```
1 module analog(clk, reset_sync, ADC_start, DAC_start, status_sync, int_data, analog_busy, r_wbar, cs_bar, cs_bar_
2 DA, ext_data, read_data);
3 input clk, reset_sync, ADC_start, DAC_start, status_sync;
4 input[7:0] int_data;
5 output analog_busy, r_wbar, cs_bar, cs_bar_DA;
6 output[7:0] read_data;
7 inout[7:0] ext_data;
8 wire[7:0] ext_data;
9
10 reg[3:0] state, nextstate;
11 reg r_wbar_int, r_wbar;
12 reg cs_bar_int, cs_bar;
13 reg cs_bar_DA_int, cs_bar_DA;
14 reg LE_ADC, LE_DAC, LE_DAC_int;
15 reg analog_busy, analog_busy_int;
16 reg[7:0] read_data;
17
18 assign ext_data = LE_DAC ? int_data : 8'hz;
19
20 // State declarations
21 parameter IDLE = 0;
22 parameter DAC0 = 1;
23 parameter DAC1 = 2;
24 parameter DAC2 = 3;
25 parameter DAC3 = 4;
26 parameter WRITE0 = 5;
27 parameter WRITE1 = 6;
28 parameter WRITE2 = 7;
29 parameter WAITSTATUSHIGH = 8;
30 parameter WAITSTATUSLOW = 9;
31 parameter READ0 = 10;
32 parameter READ1 = 11;
33 parameter READ2 = 12;
34 parameter READ3 = 13;
35
36 always @ (posedge clk)
37 begin
38     if (!reset_sync) state <= IDLE;
39     else state <= nextstate;
40
41     r_wbar <= r_wbar_int;
42     cs_bar <= cs_bar_int;
43     cs_bar_DA <= cs_bar_DA_int;
44     LE_DAC <= LE_DAC_int;
45     analog_busy <= analog_busy_int;
46
47     if (LE_ADC) read_data <= ext_data;
48 end
49
50 always @ (state or status_sync or DAC_start or ADC_start)
51 begin
52     r_wbar_int = 1;
53     cs_bar_int = 1;
54     cs_bar_DA_int = 1;
55     LE_ADC = 0;
56     LE_DAC_int = 0;
57     analog_busy_int = 1;
58
59     case (state)
60         IDLE:    begin
61             analog_busy_int = 0;
62             if (DAC_start) nextstate = DAC0;
63             else if (ADC_start) nextstate = WRITE0;
64             else nextstate = IDLE;
65         end
66
67         DAC0:    begin
68             LE_DAC_int = 1;
69             nextstate = DAC1;
70         end
71
72         DAC1:    begin
73             LE_DAC_int = 1;
74             cs_bar_DA_int = 0;
75             nextstate = DAC2;
76         end
77
78         DAC2:    begin
```

```
79         LE_DAC_int = 1;
80         nextstate = DAC3;
81     end
82
83     DAC3: begin
84         LE_DAC_int = 0;
85         nextstate = IDLE;
86     end
87
88     WRITE0: begin
89         r_wbar_int = 0;
90         cs_bar_int = 0;
91         nextstate = WRITE1;
92     end
93
94     WRITE1: begin
95         r_wbar_int = 0;
96         cs_bar_int = 0;
97         nextstate = WRITE2;
98     end
99
100    WRITE2: begin
101        r_wbar_int = 0;
102        cs_bar_int = 0;
103        nextstate = WAITSTATUSHIGH;
104    end
105
106    WAITSTATUSHIGH:
107        begin
108            cs_bar_int = 0;
109            if (status_sync) nextstate = WAITSTATUSLOW;
110            else nextstate = WAITSTATUSHIGH;
111        end
112
113    WAITSTATUSLOW:
114        begin
115            cs_bar_int = 0;
116            if (!status_sync) nextstate = READ0;
117            else nextstate = WAITSTATUSLOW;
118        end
119
120    READ0: begin
121        cs_bar_int = 0;
122        nextstate = READ1;
123    end
124
125    READ1: begin
126        cs_bar_int = 0;
127        nextstate = READ2;
128    end
129
130    READ2: begin
131        cs_bar_int = 0;
132        LE_ADC = 1;
133        nextstate = READ3;
134    end
135
136    READ3: begin
137        nextstate = IDLE;
138    end
139
140    default: nextstate = IDLE;
141 endcase
142 end
143
144 endmodule
```



```
1 module audio (reset, clock_27mhz, audio_reset_b, ac97_sdata_out, ac97_sdata_in,
2             ac97_synch, ac97_bit_clock, volume, mode, flash_data, flash_address, heartrate);
3
4     input reset, clock_27mhz;
5     output audio_reset_b;
6     output ac97_sdata_out;
7     input ac97_sdata_in;
8     output ac97_synch;
9     input ac97_bit_clock;
10    input [4:0] volume;
11    input mode;
12    input [15:0] flash_data;
13    output [22:0] flash_address;
14    input [8:0] heartrate;
15
16    wire ready;
17    wire [7:0] command_address;
18    wire [15:0] command_data;
19    wire command_valid;
20    reg [19:0] left_out_data, right_out_data;
21    wire [19:0] left_in_data, right_in_data, music_data;
22    wire [15:0] flash_data;
23    wire [22:0] flash_address;
24    wire enable11025Hz;
25
26    ///////////////////////////////////////////////////////////////////
27    //
28    // Reset Controller
29    //
30    ///////////////////////////////////////////////////////////////////
31
32    reg audio_reset_b;
33    reg [9:0] reset_count;
34
35    always @(posedge clock_27mhz) begin
36        if (reset)
37            begin
38                audio_reset_b = 1'b0;
39                reset_count = 0;
40            end
41        else if (reset_count == 1023)
42            audio_reset_b = 1'b1;
43        else
44            reset_count = reset_count+1;
45        end
46
47    ac97 ac97(ready, command_address, command_data, command_valid,
48            left_out_data, 1'b1, right_out_data, 1'b1, left_in_data,
49            right_in_data, ac97_sdata_out, ac97_sdata_in, ac97_synch,
50            ac97_bit_clock);
51
52    ac97commands cmds(clock_27mhz, ready, command_address, command_data,
53                    command_valid, volume);
54
55    enable11025Hz enable11025Hz1(clock_27mhz, reset, enable11025Hz);
56
57    music music1(clock_27mhz, enable11025Hz, music_data, mode, flash_data, flash_address, heartrate);
58
59    always @(mode or left_in_data or right_in_data or music_data)
60        case (mode)
61            1'b0: begin
62                left_out_data = 20'h00000;
63                right_out_data = 20'h00000;
64            end
65            1'b1: begin
66                left_out_data = music_data;
67                right_out_data = music_data;
68            end
69        endcase
70
71    endmodule
72
73    ///////////////////////////////////////////////////////////////////
74    //
75    // Beeper
76    //
77    ///////////////////////////////////////////////////////////////////
78
79    module beeper (reset, clock_27mhz, beep, enable);
```

```
80
81 input reset, clock_27mhz, enable;
82 output beep;
83
84 reg [15:0] count;
85 reg clock_1khz;
86
87 always @(posedge clock_27mhz)
88     if (reset)
89         begin
90             count <= 0;
91             clock_1khz <= 0;
92         end
93     else if (count == 13499)
94         begin
95             clock_1khz <= ~clock_1khz;
96             count <= 0;
97         end
98     else
99         count <= count+1;
100
101 assign beep = enable && clock_1khz;
102
103 endmodule
104
105 ///////////////////////////////////////////////////////////////////
106 //
107 // Volume Controller
108 //
109 ///////////////////////////////////////////////////////////////////
110
111 module volume (reset, clock, up, down, vol, disp);
112
113     input reset, clock, up, down;
114     output [3:0] vol;
115     output [39:0] disp;
116
117     reg [3:0] vol;
118     reg [39:0] disp;
119     reg old_up, old_down;
120
121     always @(posedge clock)
122         if (reset)
123             begin
124                 vol <= 0;
125                 old_up <= 0;
126                 old_down <= 0;
127             end
128         else
129             begin
130                 if ((up == 1) && (old_up == 0) && (vol < 15))
131                     vol <= vol+1;
132                 else if ((down == 1) && (old_down == 0) && (vol > 0))
133                     vol <= vol-1;
134                 old_up <= up;
135                 old_down <= down;
136             end
137
138     always @(vol)
139         case (vol[3:1])
140             0: disp <= { 5{8'b00000000}};
141             1: disp <= { 5{8'b01000000}};
142             2: disp <= { 5{8'b01100000}};
143             3: disp <= { 5{8'b01110000}};
144             4: disp <= { 5{8'b01111000}};
145             5: disp <= { 5{8'b01111100}};
146             6: disp <= { 5{8'b01111110}};
147             7: disp <= { 5{8'b01111111}};
148         endcase
149
150 endmodule
151
152 ///////////////////////////////////////////////////////////////////
153 //
154 // AC97
155 //
156 ///////////////////////////////////////////////////////////////////
157
158 module ac97 (ready,
```

```
159     command_address, command_data, command_valid,
160     left_data, left_valid,
161     right_data, right_valid,
162     left_in_data, right_in_data,
163     ac97_sdata_out, ac97_sdata_in, ac97_synch, ac97_bit_clock);
164
165 output ready;
166 input [7:0] command_address;
167 input [15:0] command_data;
168 input command_valid;
169 input [19:0] left_data, right_data;
170 input left_valid, right_valid;
171 output [19:0] left_in_data, right_in_data;
172
173 input ac97_sdata_in;
174 input ac97_bit_clock;
175 output ac97_sdata_out;
176 output ac97_synch;
177
178 reg ready;
179
180 reg ac97_sdata_out;
181 reg ac97_synch;
182
183 reg [7:0] bit_count;
184
185 reg [19:0] l_cmd_addr;
186 reg [19:0] l_cmd_data;
187 reg [19:0] l_left_data, l_right_data;
188 reg l_cmd_v, l_left_v, l_right_v;
189 reg [19:0] left_in_data, right_in_data;
190
191 initial begin
192     ready <= 1'b0;
193     // synthesis attribute init of ready is "0";
194     ac97_sdata_out <= 1'b0;
195     // synthesis attribute init of ac97_sdata_out is "0";
196     ac97_synch <= 1'b0;
197     // synthesis attribute init of ac97_synch is "0";
198
199     bit_count <= 8'h00;
200     // synthesis attribute init of bit_count is "0000";
201     l_cmd_v <= 1'b0;
202     // synthesis attribute init of l_cmd_v is "0";
203     l_left_v <= 1'b0;
204     // synthesis attribute init of l_left_v is "0";
205     l_right_v <= 1'b0;
206     // synthesis attribute init of l_right_v is "0";
207
208     left_in_data <= 20'h00000;
209     // synthesis attribute init of left_in_data is "00000";
210     right_in_data <= 20'h00000;
211     // synthesis attribute init of right_in_data is "00000";
212 end
213
214 always @(posedge ac97_bit_clock) begin
215     // Generate the sync signal
216     if (bit_count == 255)
217         ac97_synch <= 1'b1;
218     if (bit_count == 15)
219         ac97_synch <= 1'b0;
220
221     // Generate the ready signal
222     if (bit_count == 128)
223         ready <= 1'b1;
224     if (bit_count == 2)
225         ready <= 1'b0;
226
227     // Latch user data at the end of each frame. This ensures that the
228     // first frame after reset will be empty.
229     if (bit_count == 255)
230         begin
231             l_cmd_addr <= {command_address, 12'h000};
232             l_cmd_data <= {command_data, 4'h0};
233             l_cmd_v <= command_valid;
234             l_left_data <= left_data;
235             l_left_v <= left_valid;
236             l_right_data <= right_data;
237             l_right_v <= right_valid;
```

```

238     end
239
240     if ((bit_count >= 0) && (bit_count <= 15))
241 // Slot 0: Tags
242 case (bit_count[3:0])
243     4'h0: ac97_sdata_out <= 1'b1; // Frame valid
244     4'h1: ac97_sdata_out <= l_cmd_v; // Command address valid
245     4'h2: ac97_sdata_out <= l_cmd_v; // Command data valid
246     4'h3: ac97_sdata_out <= l_left_v; // Left data valid
247     4'h4: ac97_sdata_out <= l_right_v; // Right data valid
248     default: ac97_sdata_out <= 1'b0;
249 endcase
250
251     else if ((bit_count >= 16) && (bit_count <= 35))
252 // Slot 1: Command address (8-bits, left justified)
253 ac97_sdata_out <= l_cmd_v ? l_cmd_addr[35-bit_count] : 1'b0;
254
255     else if ((bit_count >= 36) && (bit_count <= 55))
256 // Slot 2: Command data (16-bits, left justified)
257 ac97_sdata_out <= l_cmd_v ? l_cmd_data[55-bit_count] : 1'b0;
258
259     else if ((bit_count >= 56) && (bit_count <= 75))
260 begin
261 // Slot 3: Left channel
262 ac97_sdata_out <= l_left_v ? l_left_data[19] : 1'b0;
263 l_left_data <= { l_left_data[18:0], l_left_data[19] };
264 end
265     else if ((bit_count >= 76) && (bit_count <= 95))
266 // Slot 4: Right channel
267 ac97_sdata_out <= l_right_v ? l_right_data[95-bit_count] : 1'b0;
268     else
269 ac97_sdata_out <= 1'b0;
270
271     bit_count <= bit_count+1;
272
273 end // always @ (posedge ac97_bit_clock)
274
275 always @(negedge ac97_bit_clock) begin
276     if ((bit_count >= 57) && (bit_count <= 76))
277 // Slot 3: Left channel
278 left_in_data <= { left_in_data[18:0], ac97_sdata_in };
279     else if ((bit_count >= 77) && (bit_count <= 96))
280 // Slot 4: Right channel
281 right_in_data <= { right_in_data[18:0], ac97_sdata_in };
282 end
283
284 endmodule
285
286 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
287 //
288 // AC97commands
289 //
290 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
291
292 module ac97commands (clock, ready, command_address, command_data,
293                     command_valid, volume);
294
295     input clock;
296     input ready;
297     output [7:0] command_address;
298     output [15:0] command_data;
299     output command_valid;
300     input [4:0] volume;
301
302     reg [23:0] command;
303     reg command_valid;
304
305     reg old_ready;
306     reg done;
307     reg [3:0] state;
308
309     initial begin
310         command <= 4'h0;
311         // synthesis attribute init of command is "0";
312         command_valid <= 1'b0;
313         // synthesis attribute init of command_valid is "0";
314         done <= 1'b0;
315         // synthesis attribute init of done is "0";
316         old_ready <= 1'b0;

```

```
317 // synthesis attribute init of old_ready is "0";
318 state <= 16'h0000;
319 // synthesis attribute init of state is "0000";
320 end
321
322 assign command_address = command[23:16];
323 assign command_data = command[15:0];
324
325 wire [4:0] vol;
326 assign vol = 31-volume;
327
328 always @(posedge clock) begin
329     if (ready && (!old_ready))
330     state <= state+1;
331
332     case (state)
333     4'h0: // Read ID
334         begin
335             command <= 24'h80_0000;
336             command_valid <= 1'b1;
337         end
338     4'h1: // Read ID
339         command <= 24'h80_0000;
340     4'h2: // enable changing sampling rate
341         command <= 24'h2A_0001;
342 //***** Sampling Rate *****//
343     4'h3: // Sampling rate
344         command <= 24'h2C_2B11;
345     4'h4: // Sampling rate
346         command <= 24'h32_2B11;
347     4'h5: // Master volume
348         command <= { 8'h02, 3'b000, vol, 3'b000, vol };
349     4'h6: // Aux volume
350         command <= { 8'h04, 3'b000, vol, 3'b000, vol };
351     4'h7: // Mono volume
352         command <= 24'h06_8000;
353     4'h8: // PCM volume
354         command <= 24'h18_0808;
355     4'h9: // Record source select
356         //if (source)
357         //    command <= 24'h1A_0000; // microphone
358         //else
359         command <= 24'h1A_0404; // line-in
360     4'hA: // Record gain
361         command <= 24'h1C_0000;
362     4'hB: // Line in gain
363         command <= 24'h10_8000;
364     4'hC: // Set beep volume
365         command <= 24'h0A_0000;
366     //4'hF: // Misc control bits
367         //command <= 24'h76_8000;
368     default:
369         command <= 24'h80_0000;
370         endcase // case(state)
371
372     old_ready <= ready;
373
374     end // always @ (posedge clock)
375
376 endmodule // ac97commands
377
378 ///////////////////////////////////////////////////////////////////
379 //
380 // Switch Debounce Module
381 //
382 ///////////////////////////////////////////////////////////////////
383
384 module debounce (reset, clock, noisy, clean);
385
386     input reset, clock, noisy;
387     output clean;
388
389     reg [18:0] count;
390     reg new, clean;
391
392     always @(posedge clock)
393         if (reset)
394             begin
395                 count <= 0;
```

```
396     new <= noisy;
397     clean <= noisy;
398     end
399     else if (noisy != new)
400     begin
401     new <= noisy;
402     count <= 0;
403     end
404     else if (count == 270000)
405     clean <= new;
406     else
407     count <= count+1;
408
409 endmodule
410
411
412 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
413 //
414 // Music Module
415 //
416 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
417
418 module music(clock, ready, pcm_data, start, flash_data, flash_address, heartrate);
419
420     input clock;
421     input ready;
422     input start;
423     output [19:0] pcm_data;
424     input [15:0] flash_data;
425     output [22:0] flash_address;
426     input [8:0] heartrate;
427
428     reg rdy, old_ready;
429     reg [19:0] pcm_data;
430     reg [22:0] flash_address;
431     reg [10:0] count;
432     reg [22:0] remove;
433
434     initial begin
435         old_ready <= 1'b0;
436         // synthesis attribute init of old_ready is "0";
437         pcm_data <= 20'h00000;
438         // synthesis attribute init of pcm_data is "00000";
439         count <= 0;
440     end
441
442     always @(posedge clock)
443     begin
444         if (heartrate == 9'b111111111 || heartrate < 9'd80) remove <= 1;
445         else if (heartrate < 9'd100) remove <= 23'd352;
446         else if (heartrate < 9'd120) remove <= 23'd705;
447         else if (heartrate < 9'd140) remove <= 23'd882;
448         else remove <= 23'd1058;
449
450         if (!start) begin
451             old_ready <= 1'b0;
452             pcm_data <= 20'h00000;
453             flash_address <= 0;
454             count <= 0;
455         end
456         else begin
457             if (rdy && ~old_ready) begin
458                 pcm_data <= {flash_data[7:0], 12'h000};
459                 if (count > 11'd1764) begin
460                     count <= 1;
461                     if (flash_address > 23'h00FFFF-23'd1764-remove) flash_address <= 0;
462                     else flash_address <= flash_address + remove;
463                 end
464                 else begin
465                     count <= count + 1;
466                     if (flash_address == 23'h00FFFF) flash_address <= 0;
467                     else flash_address <= flash_address+1;
468                 end
469             end
470             old_ready <= rdy;
471             rdy <= ready;
472         end
473     end
474
```

```
475 endmodule
476
477
478 ///////////////////////////////////////////////////////////////////
479 //
480 // Enable 11.025 kHz
481 //
482 ///////////////////////////////////////////////////////////////////
483
484 module enable11025Hz(clk, reset, enable11025Hz);
485
486 input clk, reset;
487 output enable11025Hz;
488 reg enable11025Hz;
489 reg[14:0] counter;
490
491 always @ (posedge clk)
492 begin
493     if (reset)
494         begin
495             enable11025Hz <= 1'b0;
496             counter <= 15'd0;
497         end
498     else if(counter == 16'd2448) // 27 MHz --> 11.025 kHz
499         begin
500             enable11025Hz <= 1'b1;
501             counter <= 15'd0;
502         end
503     else
504         begin
505             enable11025Hz <= 1'b0;
506             counter <= counter + 1;
507         end
508 end
509
510 endmodule
```

```
1 module bin2dec(clk, reset_sync, convert, heartrate, digit2, digit1, digit0);
2
3 input clk, reset_sync, convert;
4 input[8:0] heartrate;
5 output[3:0] digit2, digit1, digit0;
6
7 reg[8:0] count;
8 reg[3:0] digit2, digit1, digit0;
9 reg[3:0] digit2_int, digit1_int, digit0_int;
10 reg[1:0] state, nextstate;
11
12 parameter IDLE = 2'b00;
13 parameter WAIT = 2'b01;
14 parameter DO_CONVERT = 2'b10;
15 parameter DO_CONVERT2 = 2'b11;
16
17 always @ (posedge clk)
18 begin
19     if (!reset_sync) state <= IDLE;
20     else state <= nextstate;
21
22     if (state == IDLE) begin
23         digit2 <= 4'b1111;
24         digit1 <= 4'b1111;
25         digit0 <= 4'b1111;
26         count <= 0;
27     end
28     else if (state == WAIT) begin
29         count <= 0;
30         digit2_int <= 0;
31         digit1_int <= 0;
32         digit0_int <= 0;
33     end
34     else if (state == DO_CONVERT) begin
35         if (count == heartrate) begin
36             digit2 <= digit2_int;
37             digit1 <= digit1_int;
38             digit0 <= digit0_int;
39         end
40         else if (digit1_int == 4'b1001 && digit0_int == 4'b1001) begin
41             digit2_int <= digit2_int + 1;
42             digit1_int <= 0;
43             digit0_int <= 0;
44         end
45         else if (digit0_int == 4'b1001) begin
46             digit1_int <= digit1_int + 1;
47             digit0_int <= 0;
48         end
49         else begin
50             digit0_int <= digit0_int + 1;
51         end
52     end
53     else if (state == DO_CONVERT2)
54         count <= count + 1;
55
56 end
57
58 always @ (state or convert)
59 begin
60     case (state)
61     IDLE: begin
62         nextstate = WAIT;
63     end
64
65     WAIT: begin
66         if (convert) nextstate = DO_CONVERT;
67         else nextstate = WAIT;
68     end
69
70     DO_CONVERT: begin
71         if (count == heartrate) nextstate = WAIT;
72         else nextstate = DO_CONVERT2;
73     end
74
75     DO_CONVERT2: begin
76         nextstate = DO_CONVERT;
77     end
78
79     default: nextstate = IDLE;
```



```
80     endcase
81 end
82
83 endmodule
```

```
1
2
3 // Clock of 173 kHz //
4
5 // Uses 27MHz clock input to generate 173kHz Clock
6 // The 173kHz clock is used for the different states in
7 // FSM
8
9 module Clock_173kHz (clk, reset, enable);
10
11     input clk, reset;
12     output enable;
13
14     reg enable;
15     reg [20:0] count;
16
17     always @ (posedge clk)
18     begin
19         if (!reset)
20             begin
21                 count <= 21'd0;
22                 enable <= 0;
23             end
24         else if (count == 21'd156) // 27000000/156
25             begin
26                 enable <= 1;
27                 count <= 21'd0;
28             end
29         else
30             begin
31                 enable <= 0;
32                 count <= count + 1;
33             end
34     end
35
36 endmodule
37
38
```

```
1 ///////////////////////////////////////////////////////////////////
2 //
3 // 6.111 FPGA Labkit -- Alphanumeric Display Interface
4 //
5 //
6 // Created: November 5, 2003
7 // Author: Nathan Ickes
8 //
9 ///////////////////////////////////////////////////////////////////
10
11 module display (reset, clock_27mhz,
12               disp_blank, disp_clock, disp_rs, disp_ce_b,
13               disp_reset_b, disp_data_out, dots);
14
15   input  reset, clock_27mhz;
16   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
17          disp_reset_b;
18   input [639:0] dots;
19
20   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;
21
22   ///////////////////////////////////////////////////////////////////
23   //
24   // Display Clock
25   //
26   // Generate a 500kHz clock for driving the displays.
27   //
28   ///////////////////////////////////////////////////////////////////
29
30   reg [4:0] count;
31   reg [7:0] reset_count;
32   reg clock;
33   wire dreset;
34
35   always @(posedge clock_27mhz)
36     begin
37     if (reset)
38       begin
39         count = 0;
40         clock = 0;
41       end
42     else if (count == 26)
43       begin
44         clock = ~clock;
45         count = 5'h00;
46       end
47     else
48       count = count+1;
49     end
50
51   always @(posedge clock_27mhz)
52     if (reset)
53       reset_count <= 100;
54     else
55       reset_count <= (reset_count==0) ? 0 : reset_count-1;
56
57   assign dreset = (reset_count != 0);
58
59   assign disp_clock = ~clock;
60
61   ///////////////////////////////////////////////////////////////////
62   //
63   // Display State Machine
64   //
65   ///////////////////////////////////////////////////////////////////
66
67   reg [7:0] state;
68   reg [9:0] dot_index;
69   reg [31:0] control;
70
71   assign disp_blank = 1'b0; // low <= not blanked
72
73   always @(posedge clock)
74     if (dreset)
75       begin
76         state <= 0;
77         dot_index <= 0;
78         control <= 32'h7F7F7F7F;
79       end
```

```
80     else
81         casex (state)
82     8'h00:
83         begin
84             // Reset displays
85             disp_data_out <= 1'b0;
86             disp_rs <= 1'b0; // dot register
87             disp_ce_b <= 1'b1;
88             disp_reset_b <= 1'b0;
89             dot_index <= 0;
90             state <= state+1;
91         end
92
93     8'h01:
94         begin
95             // End reset
96             disp_reset_b <= 1'b1;
97             state <= state+1;
98         end
99
100    8'h02:
101        begin
102            // Initialize dot register
103            disp_ce_b <= 1'b0;
104            disp_data_out <= 1'b0; // dot_index[0];
105            if (dot_index == 639)
106                state <= state+1;
107            else
108                dot_index <= dot_index+1;
109        end
110
111    8'h03:
112        begin
113            // Latch dot data
114            disp_ce_b <= 1'b1;
115            dot_index <= 31;
116            state <= state+1;
117        end
118
119    8'h04:
120        begin
121            // Setup the control register
122            disp_rs <= 1'b1; // Select the control register
123            disp_ce_b <= 1'b0;
124            disp_data_out <= control[31];
125            control <= {control[30:0], 1'b0};
126            if (dot_index == 0)
127                state <= state+1;
128            else
129                dot_index <= dot_index-1;
130        end
131
132    8'h05:
133        begin
134            // Latch the control register data
135            disp_ce_b <= 1'b1;
136            dot_index <= 639;
137            state <= state+1;
138        end
139
140    8'h06:
141        begin
142            // Load the user's dot data into the dot register
143            disp_rs <= 1'b0; // Select the dot register
144            disp_ce_b <= 1'b0;
145            disp_data_out <= dots[dot_index];
146            if (dot_index == 0)
147                state <= 5;
148            else
149                dot_index <= dot_index-1;
150        end
151    endcase
152 endmodule
153
154
```

```
1 module ekg(clk, reset, cs_bar_DA_ekg, data, rxdata, wireless_dataout, cs_bar_DA_wireless);
2
3     input clk, reset;
4     output cs_bar_DA_ekg, cs_bar_DA_wireless;
5     output[7:0] data, wireless_dataout;
6     input[7:0] rxdata;
7
8     wire reset_sync, sample_ekg, sample_wireless;
9     wire[11:0] ekg_addr;
10    wire[7:0] ekg_q;
11
12    ekg_synchronizer ekg_synchronizer1(clk, reset, reset_sync);
13    enable360Hz enable360Hz1(clk, reset_sync, sample_ekg);
14    ekg_simulator ekg_simulator1(ekg_addr, ekg_q);
15    ekg_analog ekg_analog1(clk, reset_sync, sample_ekg, cs_bar_DA_ekg, ekg_addr, ekg_q, data);
16    wireless_analog wireless_analog1(clk, reset_sync, sample_wireless, cs_bar_DA_wireless, rxdata, wireless_dataout);
17    enable12andahalfHz enable12andahalfHz1(clk, reset_sync, sample_wireless);
18
19 endmodule
```

```
1 module ekg_analog(clk, reset_sync, sample_ekg, cs_bar_DA_ekg, count, ekg_q, data);
2
3 input clk, reset_sync, sample_ekg;
4 input[7:0] ekg_q;
5 output cs_bar_DA_ekg;
6 output[11:0] count;
7 output[7:0] data;
8
9 reg[2:0] state, nextstate;
10 reg cs_bar_DA_ekg, cs_bar_DA_ekg_int, LE_DAC, LE_DAC_int;
11 reg[11:0] count, count_int;
12 wire[7:0] data;
13
14 assign data = LE_DAC ? ekg_q : 8'hz;
15
16 parameter IDLE = 0;
17 parameter DAC0 = 1;
18 parameter DAC1 = 2;
19 parameter DAC2 = 3;
20 parameter DAC3 = 4;
21 parameter WAIT = 5;
22
23 always @ (posedge clk)
24 begin
25     count <= count_int;
26     cs_bar_DA_ekg <= cs_bar_DA_ekg_int;
27     LE_DAC <= LE_DAC_int;
28
29     if (!reset_sync) begin state <= IDLE; count_int <= 0; end
30     else state <= nextstate;
31
32     if (state == IDLE) count_int <= 0;
33     else if (state == DAC3 && count == 4095) count_int <= 0;
34     else if (state == DAC3) count_int <= count_int + 1;
35 end
36
37 always @ (state or sample_ekg)
38 begin
39     cs_bar_DA_ekg_int = 1;
40     LE_DAC_int = 0;
41
42     case (state)
43     IDLE: begin
44         nextstate = WAIT;
45     end
46
47     WAIT: begin
48         if (sample_ekg) nextstate = DAC0;
49         else nextstate = WAIT;
50     end
51
52     DAC0: begin
53         LE_DAC_int = 1;
54         nextstate = DAC1;
55     end
56
57     DAC1: begin
58         LE_DAC_int = 1;
59         cs_bar_DA_ekg_int = 0;
60         nextstate = DAC2;
61     end
62
63     DAC2: begin
64         LE_DAC_int = 1;
65         nextstate = DAC3;
66     end
67
68     DAC3: begin
69         LE_DAC_int = 0;
70         nextstate = WAIT;
71     end
72
73     default: nextstate = IDLE;
74 endcase
75 end
76
77 endmodule
```

```
1 /*****
2 * This file is owned and controlled by Xilinx and must be used *
3 * solely for design, simulation, implementation and creation of *
4 * design files limited to Xilinx devices or technologies. Use *
5 * with non-Xilinx devices or technologies is expressly prohibited *
6 * and immediately terminates your license. *
7 *
8 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" *
9 * SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR *
10 * XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION *
11 * AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION *
12 * OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS *
13 * IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, *
14 * AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE *
15 * FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY *
16 * WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE *
17 * IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR *
18 * REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF *
19 * INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS *
20 * FOR A PARTICULAR PURPOSE. *
21 *
22 * Xilinx products are not intended for use in life support *
23 * appliances, devices, or systems. Use in such applications are *
24 * expressly prohibited. *
25 *
26 * (c) Copyright 1995-2004 Xilinx, Inc. *
27 * All rights reserved. *
28 *****/
29 // The synopsys directives "translate_off/translate_on" specified below are
30 // supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
31 // tools. Ensure they are correct for your synthesis tool(s).
32
33 // You must compile the wrapper file ekg_simulator.v when simulating
34 // the core, ekg_simulator. When compiling the wrapper file, be sure to
35 // reference the XilinxCoreLib Verilog simulation library. For detailed
36 // instructions, please refer to the "CORE Generator Guide".
37
38 module ekg_simulator (
39     A,
40     SPO); // synthesis black_box
41
42 input [11 : 0] A;
43 output [7 : 0] SPO;
44
45 // synopsys translate_off
46
47 C_DIST_MEM_V7_1 #(
48     12, // c_addr_width
49     "0", // c_default_data
50     1, // c_default_data_radix
51     4096, // c_depth
52     1, // c_enable_rlocs
53     1, // c_generate_mif
54     0, // c_has_clk
55     0, // c_has_d
56     0, // c_has_dpo
57     0, // c_has_dpri
58     0, // c_has_i_ce
59     0, // c_has_qdpo
60     0, // c_has_qdpo_ce
61     0, // c_has_qdpo_clk
62     0, // c_has_qdpo_rst
63     0, // c_has_qdpo_srst
64     0, // c_has_qspo
65     0, // c_has_qspo_ce
66     0, // c_has_qspo_rst
67     0, // c_has_qspo_srst
68     0, // c_has_rd_en
69     1, // c_has_spo
70     0, // c_has_spra
71     0, // c_has_we
72     0, // c_latency
73     "ekg_simulator.mif", // c_mem_init_file
74     0, // c_mem_type
75     0, // c_mux_type
76     0, // c_qce_joined
77     0, // c_qualify_we
78     1, // c_read_mif
79     0, // c_reg_a_d_inputs
```

```
80     0, // c_reg_dpra_input
81     0, // c_sync_enable
82     8) // c_width
83 inst (
84     .A(A),
85     .SPO(SPO),
86     .D(),
87     .DPRA(),
88     .SPRA(),
89     .I_CE(),
90     .QSPO_CE(),
91     .WE(),
92     .CLK(),
93     .QDPO_CE(),
94     .QDPO_CLK(),
95     .RD_EN(),
96     .QSPO_RST(),
97     .QDPO_RST(),
98     .QSPO_SRST(),
99     .QDPO_SRST(),
100    .QSPO(),
101    .DPO(),
102    .QDPO());
103
104
105 // synopsys translate_on
106
107 // FPGA Express black box declaration
108 // synopsys attribute fpga_dont_touch "true"
109 // synthesis attribute fpga_dont_touch of ekg_simulator is "true"
110
111 // XST black box declaration
112 // box_type "black_box"
113 // synthesis attribute box_type of ekg_simulator is "black_box"
114
115 endmodule
116
117
```



```
1 module ekg_synchronizer(clk, reset, reset_sync);
2
3 input clk, reset;
4 output reset_sync;
5 reg x1, reset_sync;
6
7 always @ (posedge clk)
8 begin
9     x1 <= reset;
10    reset_sync <= x1;
11 end
12
13 endmodule
```

```
1 module enable12andahalfHz(clk, reset_sync, sample_wireless);
2
3 input clk, reset_sync;
4 output sample_wireless;
5 reg sample_wireless;
6 reg[19:0] counter;
7
8 always @ (posedge clk or negedge reset_sync)
9 begin
10     if(reset_sync == 0)
11     begin
12         sample_wireless <= 1'b0;
13         counter <= 20'd0;
14     end
15     else if(counter == 20'd799999) // 10 MHz --> 12.5 Hz
16     begin
17         sample_wireless <= 1'b1;
18         counter <= 20'd0;
19     end
20     else
21     begin
22         sample_wireless <= 1'b0;
23         counter <= counter + 1;
24     end
25 end
26
27 endmodule
```

```
1 module enable250Hz(clk, reset_sync, sample);
2
3 input clk, reset_sync;
4 output sample;
5 reg sample;
6 reg[15:0] counter;
7
8 always @ (posedge clk or negedge reset_sync)
9 begin
10     if(reset_sync == 0)
11     begin
12         sample <= 1'b0;
13         counter <= 16'd0;
14     end
15     // else if(counter == 9'd4)
16     else if(counter == 16'd40000) // 10 MHz --> 10 kHz
17     begin
18         sample <= 1'b1;
19         counter <= 16'd0;
20     end
21     else
22     begin
23         sample <= 1'b0;
24         counter <= counter + 1;
25     end
26 end
27
28 endmodule
```

```
1 module enable360Hz(clk, reset_sync, sample_ekg);
2
3 input clk, reset_sync;
4 output sample_ekg;
5 reg sample_ekg;
6 reg[14:0] counter;
7
8 always @ (posedge clk or negedge reset_sync)
9 begin
10     if(reset_sync == 0)
11     begin
12         sample_ekg <= 1'b0;
13         counter <= 15'd0;
14     end
15     else if(counter == 15'd27778) //d111115) // 10 MHz --> 360 Hz
16     begin
17         sample_ekg <= 1'b1;
18         counter <= 15'd0;
19     end
20     else
21     begin
22         sample_ekg <= 1'b0;
23         counter <= counter + 1;
24     end
25 end
26
27 endmodule
```

```
1 module filter(clk, reset_sync, filter_start, sram_q, rom1_q, rom2_q, filter_busy, ADC_start, sram_we, sram_addr,
2   rom1_addr, rom2_addr, int_data);
3
4 input clk, reset_sync, filter_start;
5 input[7:0] sram_q, rom1_q, rom2_q;
6 output filter_busy, sram_we, ADC_start;
7 output[5:0] sram_addr, rom1_addr, rom2_addr;
8 output[7:0] int_data;
9
10 reg sram_we, sram_we_int, filter_busy, filter_busy_int, ADC_start, ADC_start_int;
11 reg LD_accum;
12 reg[5:0] sram_addr, sram_addr_recent, rom1_addr, rom2_addr, count, count_int;
13 reg[2:0] state, nextstate;
14 reg[15:0] Q1, Q2;
15 reg[7:0] int_data;
16 wire[15:0] product1, product2, D1, D2, Q;
17
18 parameter IDLE = 0;
19 parameter STORE0 = 1;
20 parameter STORE1 = 2;
21 parameter STORE2 = 3;
22 parameter STARTADC = 4;
23 parameter CONVOLVE0 = 5;
24 parameter CONVOLVE1 = 6;
25
26 always @ (posedge clk)
27 begin
28   sram_we <= sram_we_int;
29   filter_busy <= filter_busy_int;
30   ADC_start <= ADC_start_int;
31   rom1_addr <= count;
32   rom2_addr <= count;
33   sram_addr <= sram_addr_recent + count;
34
35   if (!reset_sync) begin
36     state <= IDLE;
37     sram_addr_recent <= 0;
38     count <= 0;
39   end
40   else state <= nextstate;
41
42   if (LD_accum) begin Q1 <= D1; Q2 <= D2; end
43   else begin Q1 <= Q1; Q2 <= Q2; end
44
45   if (state == IDLE) begin
46     count <= 6'b0;
47     Q1 <= 16'b0;
48     Q2 <= 16'b0;
49   end
50   else if (state == CONVOLVE1 && count == 63) begin
51     count <= 6'b0;
52     sram_addr_recent <= sram_addr_recent - 1;
53   end
54   else if (state == CONVOLVE1)
55     count <= count + 1;
56 end
57
58 always @ (state or reset_sync or filter_start)
59 begin
60   sram_we_int = 0;
61   filter_busy_int = 1;
62   ADC_start_int = 0;
63   LD_accum = 0;
64
65   case (state)
66     IDLE: begin
67       filter_busy_int = 0;
68       if (filter_start) nextstate = STORE0;
69       else nextstate = IDLE;
70     end
71     STORE0: begin
72       nextstate = STORE1;
73     end
74     STORE1: begin
75       sram_we_int = 1;
76       nextstate = STORE2;
77     end
78   end
```

```
79
80     STORE2:   begin
81               nextstate = STARTADC;
82             end
83
84     STARTADC: begin
85               ADC_start_int = 1;
86               nextstate = CONVOLVE0;
87             end
88
89     CONVOLVE0: begin
90               LD_accum = 1;
91               nextstate = CONVOLVE1;
92             end
93
94     CONVOLVE1: begin
95               if (count == 6'b111111) begin
96                 if (!Q[15] && (Q[14] || Q[13] || Q[12] || Q[11])) int_data = 8'b11111111;
97                 else if (Q[15] && (!Q[14] || !Q[13] || !Q[12] || !Q[11])) int_data = 8'b00000000;
98                 else int_data = {~Q[15], Q[10:4]};
99                 nextstate = IDLE;
100                end
101                else nextstate = CONVOLVE0;
102              end
103
104           default: nextstate = IDLE;
105         endcase
106     end
107
108     multiplier multiplier1(sram_q, rom1_q, product1);
109     assign D1 = product1 + Q1;
110     multiplier multiplier2(sram_q, rom2_q, product2);
111     assign D2 = product2 + Q2;
112     multiplier_2cpmt multiplier_2cpmt1(Q1[15:8], Q2[15:8], Q);
113
114 endmodule
```

```
1 ///////////////////////////////////////////////////////////////////
2 //
3 // 6.111 FPGA Labkit -- Flash ROM Interface
4 //
5 // For Labkit Revision 004
6 //
7 //
8 // Created: January 22, 2005
9 // Author: Nathan Ickes
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 `define FLASHOP_IDLE 2'b00
14 `define FLASHOP_READ 2'b01
15 `define FLASHOP_WRITE 2'b10
16
17 module flash_int(reset, clock, op, address, wdata, rdata, busy, flash_data,
18 flash_address, flash_oe_b, flash_oe_b, flash_we_b,
19 flash_reset_b, flash_sts, flash_byte_b);
20
21 parameter access_cycles = 5;
22 parameter reset_assert_cycles = 1000;
23 parameter reset_recovery_cycles = 30;
24
25 input reset, clock; // Reset and clock for the flash interface
26 input [1:0] op; // Flash operation select (read, write, idle)
27 input [22:0] address;
28 input [15:0] wdata;
29 output [15:0] rdata;
30 output busy;
31 inout [15:0] flash_data;
32 output [23:0] flash_address;
33 output flash_oe_b, flash_oe_b, flash_we_b;
34 output flash_reset_b, flash_byte_b;
35 input flash_sts;
36
37 reg [1:0] lop;
38 reg [15:0] rdata;
39 reg busy;
40 reg [15:0] flash_wdata;
41 reg flash_ddata;
42 reg [23:0] flash_address;
43 reg flash_oe_b, flash_we_b, flash_reset_b;
44
45 assign flash_oe_b = flash_oe_b && flash_we_b;
46 assign flash_byte_b = 1; // 1 = 16-bit mode (A0 ignored)
47
48 assign flash_data = flash_ddata ? flash_wdata : 16'hZ;
49
50 ///////////////////////////////////////////////////////////////////
51 //
52 //
53 //
54 ///////////////////////////////////////////////////////////////////
55
56 initial
57 flash_reset_b <= 1'b1;
58
59 reg [9:0] state;
60
61 always @(posedge clock)
62 if (reset)
63 begin
64 state <= 0;
65 flash_reset_b <= 0;
66 flash_we_b <= 1;
67 flash_oe_b <= 1;
68 flash_ddata <= 0;
69 busy <= 1;
70 end
71 else if (flash_reset_b == 0)
72 if (state == reset_assert_cycles)
73 begin
74 flash_reset_b <= 1;
75 state <= 1023-reset_recovery_cycles;
76 end
77 else
78 state <= state+1;
79 else if ((state == 0) && !busy)
```

```
80 // The flash chip and this state machine are both idle. Latch the user's
81 // address and write data inputs. Deassert OE and WE, and stop driving
82 // the data buss ourselves. If a flash operation (read or write) is
83 // requested, move to the next state.
84 begin
85 flash_address <= {address, 1'b0};
86 flash_we_b <= 1;
87 flash_oe_b <= 1;
88 flash_ddata <= 0;
89 flash_wdata <= wdata;
90 lop <= op;
91 if (op != `FLASHOP_IDLE)
92 begin
93 busy <= 1;
94 state <= state+1;
95 end
96 else
97 busy <= 0;
98 end
99 else if ((state==0) && flash_sts)
100 busy <= 0;
101 else if (state == 1)
102 // The first stage of a flash operation. The address bus is already set,
103 // so, if this is a read, we assert OE. For a write, we start driving
104 // the user's data onto the flash databus (the value was latched in the
105 // previous state.
106 begin
107 if (lop == `FLASHOP_WRITE)
108 flash_ddata <= 1;
109 else if (lop == `FLASHOP_READ)
110 flash_oe_b <= 0;
111 state <= state+1;
112 end
113 else if (state == 2)
114 // The second stage of a flash operation. Nothing to do for a read. For
115 // a write, we assert WE.
116 begin
117 if (lop == `FLASHOP_WRITE)
118 flash_we_b <= 0;
119 state <= state+1;
120 end
121 else if (state == access_cycles+1)
122 // The third stage of a flash operation. For a read, we latch the data
123 // from the flash chip. For a write, we deassert WE.
124 begin
125 if (lop == `FLASHOP_WRITE)
126 flash_we_b <= 1;
127 if (lop == `FLASHOP_READ)
128 rdata <= flash_data;
129 state <= 0;
130 end
131 else
132 begin
133 if (!flash_sts)
134 busy <= 1;
135 state <= state+1;
136 end
137 endmodule
138
139
```



```
1 ///////////////////////////////////////////////////////////////////
2 //
3 // 6.111 FPGA Labkit -- Flash Test Program
4 //
5 // For Labkit Revision 004
6 //
7 //
8 // Created: January 23, 2005, during a blizzard
9 // Author: Nathan Ickes
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 `timescale 1ns/1ns
14 `include "flash_int.v"
15 `include "test_fsm.v"
16 `include "display.v"
17
18 module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
19               ac97_bit_clock,
20
21               vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
22               vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
23               vga_out_vsync,
24
25               tv_out_ycrCb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
26               tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
27               tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
28
29               tv_in_ycrCb, tv_in_data_valid, tv_in_line_clock1,
30               tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
31               tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
32               tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
33
34               ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
35               ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
36
37               ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
38               ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
39
40               clock_feedback_out, clock_feedback_in,
41
42               flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
43               flash_reset_b, flash_sts, flash_byte_b,
44
45               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
46
47               mouse_clock, mouse_data, keyboard_clock, keyboard_data,
48
49               clock_27mhz, clock1, clock2,
50
51               disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
52               disp_reset_b, disp_data_in,
53
54               button0, button1, button2, button3, button_enter, button_right,
55               button_left, button_down, button_up,
56
57               switch,
58
59               led,
60
61               user1, user2, user3, user4,
62
63               daughtercard,
64
65               systemace_data, systemace_address, systemace_ce_b,
66               systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
67
68               analyzer1_data, analyzer1_clock,
69               analyzer2_data, analyzer2_clock,
70               analyzer3_data, analyzer3_clock,
71               analyzer4_data, analyzer4_clock);
72
73 output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
74 input ac97_bit_clock, ac97_sdata_in;
75
76 output [7:0] vga_out_red, vga_out_green, vga_out_blue;
77 output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
78        vga_out_hsync, vga_out_vsync;
79
```

```
80 output [9:0] tv_out_ycrcb;
81 output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
82 tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
83 tv_out_subcar_reset;
84
85 input [19:0] tv_in_ycrcb;
86 input tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
87 tv_in_hff, tv_in_aff;
88 output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
89 tv_in_reset_b, tv_in_clock;
90 inout tv_in_i2c_data;
91
92 inout [35:0] ram0_data;
93 output [18:0] ram0_address;
94 output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
95 output [3:0] ram0_bwe_b;
96
97 inout [35:0] ram1_data;
98 output [18:0] ram1_address;
99 output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
100 output [3:0] ram1_bwe_b;
101
102 input clock_feedback_in;
103 output clock_feedback_out;
104
105 inout [15:0] flash_data;
106 output [23:0] flash_address;
107 output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
108 input flash_sts;
109
110 output rs232_txd, rs232_rts;
111 input rs232_rxd, rs232_cts;
112
113 input mouse_clock, mouse_data, keyboard_clock, keyboard_data;
114
115 input clock_27mhz, clock1, clock2;
116
117 output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
118 input disp_data_in;
119 output disp_data_out;
120
121 input button0, button1, button2, button3, button_enter, button_right,
122 button_left, button_down, button_up;
123 input [7:0] switch;
124 output [7:0] led;
125
126 inout [31:0] user1, user2, user3, user4;
127
128 inout [43:0] daughtercard;
129
130 inout [15:0] systemace_data;
131 output [6:0] systemace_address;
132 output systemace_ce_b, systemace_we_b, systemace_oe_b;
133 input systemace_irq, systemace_mpbrdy;
134
135 output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
136 analyzer4_data;
137 output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;
138
139 ///////////////////////////////////////////////////////////////////
140 //
141 // Reset Generation
142 //
143 // A shift register primitive is used to generate an active-high reset
144 // signal that remains high for 16 clock cycles after configuration finishes
145 // and the FPGA's internal clocks begin toggling.
146 //
147 ///////////////////////////////////////////////////////////////////
148
149 wire reset;
150 SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(reset),
151 .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
152 defparam reset_sr.INIT = 16'hFFFF;
153
154 ///////////////////////////////////////////////////////////////////
155 //
156 // Flash ROM Tester
157 //
158 ///////////////////////////////////////////////////////////////////
```

```
159
160 wire [1:0] fop;
161 wire [22:0] faddress;
162 wire [15:0] fwdata, frdata;
163 wire fbusy;
164 wire [639:0] dots;
165 wire [7:0] FPGA_ROM_data;
166 wire [12:0] FPGA_ROM_addr;
167
168 flash_int flashint1 (reset, clock_27mhz, fop, faddress, fwdata, frdata,
169     fbusy, flash_data, flash_address, flash_ce_b,
170     flash_oe_b, flash_we_b, flash_reset_b, 1'b1,
171     flash_byte_b);
172
173 test_fsm fsm1 (reset, clock_27mhz, fop, faddress, fwdata, frdata, fbusy,
174     dots, FPGA_ROM_data, FPGA_ROM_addr);
175
176 display displ (reset, clock_27mhz, disp_blank, disp_clock, disp_rs,
177     disp_ce_b, disp_reset_b, disp_data_out, dots);
178
179 romsound romsound1 (FPGA_ROM_addr, FPGA_ROM_data);
180
181 assign analyzer1_clock = clock_27mhz;
182
183 assign analyzer1_data = {7'b0000000, // 15-9
184     fbusy, // 8
185     flash_ce_b, // 7
186     flash_oe_b, // 6
187     flash_we_b, // 5
188     flash_reset_b, // 4
189     flash_sts, // 3
190     fop, // 2-1
191     reset}; // 0
192
193 assign analyzer2_data = flash_data;
194
195 assign user1 = {flash_data, // 31-16
196     6'h00, // 15-10
197     fbusy, // 9
198     flash_ce_b, // 8
199     flash_oe_b, // 7
200     flash_we_b, // 6
201     flash_reset_b, // 5
202     flash_sts, // 4
203     fop, // 3-2
204     reset, // 1
205     clock_27mhz}; // 0
206
207 assign analyzer3_data = {9'b000000000,
208     flash_address[22:16]};
209
210 assign analyzer4_data = flash_address[15:0];
211
212 assign user2 = {disp_blank, // 31
213     disp_clock, // 30
214     disp_rs, // 29
215     disp_ce_b, // 28
216     disp_reset_b, // 27
217     disp_data_out, // 26
218     3'b000, // 27-23
219     flash_address}; // 22-0
220
221 //assign led = {6'b111111, ~fop};
222 assign led = {dots[7:0]};
223
224 ///////////////////////////////////////////////////////////////////
225 //
226 // Default I/O Assignments
227 //
228 ///////////////////////////////////////////////////////////////////
229
230 // Audio Input and Output
231 assign beep= 1'b0;
232 assign audio_reset_b = 1'b0;
233 assign ac97_synch = 1'b0;
234
235 // VGA Output
236 assign vga_out_red = 10'h0;
237 assign vga_out_green = 10'h0;
```

```
238     assign vga_out_blue = 10'h0;
239     assign vga_out_sync_b = 1'b1;
240     assign vga_out_blank_b = 1'b1;
241     assign vga_out_pixel_clock = 1'b0;
242     assign vga_out_hsync = 1'b0;
243     assign vga_out_vsync = 1'b0;
244
245     // Video Output
246     assign tv_out_ycrcb = 10'h0;
247     assign tv_out_reset_b = 1'b0;
248     assign tv_out_clock = 1'b0;
249     assign tv_out_i2c_clock = 1'b0;
250     assign tv_out_i2c_data = 1'b0;
251     assign tv_out_pal_ntsc = 1'b0;
252     assign tv_out_hsync_b = 1'b1;
253     assign tv_out_vsync_b = 1'b1;
254     assign tv_out_blank_b = 1'b1;
255     assign tv_out_subcar_reset = 1'b0;
256
257     // Video Input
258     assign tv_in_i2c_clock = 1'b0;
259     assign tv_in_fifo_read = 1'b0;
260     assign tv_in_fifo_clock = 1'b0;
261     assign tv_in_iso = 1'b0;
262     assign tv_in_reset_b = 1'b0;
263     assign tv_in_clock = 1'b0;
264     assign tv_in_i2c_data = 1'bZ;
265
266     // SRAMs
267     assign ram0_data = 36'hZ;
268     assign ram0_address = 19'h0;
269     assign ram0_adv_ld = 1'b0;
270     assign ram0_clk = 1'b0;
271     assign ram0_cen_b = 1'b1;
272     assign ram0_ce_b = 1'b1;
273     assign ram0_oe_b = 1'b1;
274     assign ram0_we_b = 1'b1;
275     assign ram0_bwe_b = 4'hF;
276     assign ram1_data = 36'hZ;
277     assign ram1_address = 19'h0;
278     assign ram1_adv_ld = 1'b0;
279     assign ram1_clk = 1'b0;
280     assign ram1_cen_b = 1'b1;
281     assign ram1_ce_b = 1'b1;
282     assign ram1_oe_b = 1'b1;
283     assign ram1_we_b = 1'b1;
284     assign ram1_bwe_b = 4'hF;
285     assign clock_feedback_out = 1'b0;
286
287     // RS-232 Interface
288     assign rs232_txd = 1'b1;
289     assign rs232_rts = 1'b1;
290
291     // User I/Os
292     assign user3 = 32'hZ;
293     assign user4 = 32'hZ;
294
295     // Daughtercard Connectors
296     assign daughtercard = 44'hZ;
297
298     // SystemACE Microprocessor Port
299     assign systemace_data = 16'hZ;
300     assign systemace_address = 7'h0;
301     assign systemace_ce_b = 1'b1;
302     assign systemace_we_b = 1'b1;
303     assign systemace_oe_b = 1'b1;
304
305 endmodule
306
```

```
1 module heartbeat_detector(clk, reset_sync, int_data, heartbeat);
2
3 input clk, reset_sync;
4 input[7:0] int_data;
5 output heartbeat;
6 reg heartbeat;
7
8 always @ (posedge clk)
9 begin
10     if (!reset_sync) heartbeat <= 0;
11     else if ((int_data[7] && int_data[6]) || (int_data[7] && !int_data[6] && int_data[5])) heartbeat <= 1;
12     else heartbeat <= 0;
13 end
14
15 endmodule
```

```
1 module heartrate_detector(clk, reset_sync, heartbeat, heartrate, convert);
2
3 input clk, reset_sync, heartbeat;
4 output[8:0] heartrate;
5 output convert;
6
7 wire[2:0] six;
8 wire[11:0] heartrate1;
9 reg convert;
10 reg[8:0] heartrate_int, heartrate;
11 reg[26:0] count10sec;
12 reg reset_count;
13
14 assign six = 3'b110;
15 assign heartrate1 = heartrate_int * six;
16
17 always @ (posedge clk)
18 begin
19     if (!reset_sync) begin
20         count10sec <= 0;
21         reset_count <= 1;
22         heartrate <= 9'b111111111;
23         convert <= 0;
24     end
25     else if (count10sec == 27'd99999999) begin // 10 MHz -> 0.1 Hz
26         count10sec <= 0;
27         heartrate <= heartrate1[8:0];
28         reset_count <= 1;
29         convert <= 1;
30     end
31     else begin
32         count10sec <= count10sec + 1;
33         reset_count <= 0;
34         convert <= 0;
35     end
36 end
37
38 always @ (posedge heartbeat or posedged reset_count)
39 begin
40     if (reset_count) heartrate_int <= 0;
41     else heartrate_int <= heartrate_int + 1;
42 end
43
44 endmodule
```

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // 6.111 FPGA Labkit -- Template Toplevel Module
4 //
5 // For Labkit Revision 004
6 //
7 //
8 // Created: October 31, 2004, from revision 003 file
9 // Author: Nathan Ickes
10 //
11 ////////////////////////////////////////////////////////////////////
12 //
13 // CHANGES FOR BOARD REVISION 004
14 //
15 // 1) Added signals for logic analyzer pods 2-4.
16 // 2) Expanded "tv_in_ycrCb" to 20 bits.
17 // 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
18 //     "tv_out_i2c_clock".
19 // 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
20 //     output of the FPGA, and "in" is an input.
21 //
22 // CHANGES FOR BOARD REVISION 003
23 //
24 // 1) Combined flash chip enables into a single signal, flash_ce_b.
25 //
26 // CHANGES FOR BOARD REVISION 002
27 //
28 // 1) Added SRAM clock feedback path input and output
29 // 2) Renamed "mousedata" to "mouse_data"
30 // 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
31 //     the data bus, and the byte write enables have been combined into the
32 //     4-bit ram#_bwe_b bus.
33 // 4) Removed the "systemace_clock" net, since the SystemACE clock is now
34 //     hardwired on the PCB to the oscillator.
35 //
36 ////////////////////////////////////////////////////////////////////
37 //
38 // Complete change history (including bug fixes)
39 //
40 // 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
41 //              actually populated on the boards. (The boards support up to
42 //              256Mb devices, with 25 address lines.)
43 //
44 // 2004-Apr-29: Change history started
45 //
46 // 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
47 //              actually populated on the boards. (The boards support up to
48 //              72Mb devices, with 21 address lines.)
49 //
50 // 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
51 //              value. (Previous versions of this file declared this port to
52 //              be an input.)
53 //
54 // 2004-Oct-31: Adapted to new revision 004 board.
55 //
56 ////////////////////////////////////////////////////////////////////
57
58 module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
59               ac97_bit_clock,
60
61               vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
62               vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
63               vga_out_vsync,
64
65               tv_out_ycrCb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
66               tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
67               tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
68
69               tv_in_ycrCb, tv_in_data_valid, tv_in_line_clock1,
70               tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
71               tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
72               tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
73
74               ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
75               ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
76
77               ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
78               ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
79
```

```
80     clock_feedback_out, clock_feedback_in,
81
82     flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
83     flash_reset_b, flash_sts, flash_byte_b,
84
85     rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
86
87     mouse_clock, mouse_data, keyboard_clock, keyboard_data,
88
89     clock_27mhz, clock1, clock2,
90
91     disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
92     disp_reset_b, disp_data_in,
93
94     button0, button1, button2, button3, button_enter, button_right,
95     button_left, button_down, button_up,
96
97     switch,
98
99     led,
100
101     user1, user2, user3, user4,
102
103     daughtercard,
104
105     systemace_data, systemace_address, systemace_ce_b,
106     systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbdrdy,
107
108     analyzer1_data, analyzer1_clock,
109     analyzer2_data, analyzer2_clock,
110     analyzer3_data, analyzer3_clock,
111     analyzer4_data, analyzer4_clock);
112
113 output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
114 input  ac97_bit_clock, ac97_sdata_in;
115
116 output [7:0] vga_out_red, vga_out_green, vga_out_blue;
117 output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
118        vga_out_hsync, vga_out_vsync;
119
120 output [9:0] tv_out_ycrcb;
121 output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
122        tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
123        tv_out_subcar_reset;
124
125 input  [19:0] tv_in_ycrcb;
126 input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
127        tv_in_hff, tv_in_aff;
128 output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
129        tv_in_reset_b, tv_in_clock;
130 inout  tv_in_i2c_data;
131
132 inout  [35:0] ram0_data;
133 output [18:0] ram0_address;
134 output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
135 output [3:0] ram0_bwe_b;
136
137 inout  [35:0] ram1_data;
138 output [18:0] ram1_address;
139 output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
140 output [3:0] ram1_bwe_b;
141
142 input  clock_feedback_in;
143 output clock_feedback_out;
144
145 inout  [15:0] flash_data;
146 output [23:0] flash_address;
147 output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
148 input  flash_sts;
149
150 output rs232_txd, rs232_rts;
151 input  rs232_rxd, rs232_cts;
152
153 input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
154
155 input  clock_27mhz, clock1, clock2;
156
157 output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
158 input  disp_data_in;
```



```
159     output  disp_data_out;
160
161     input   button0, button1, button2, button3, button_enter, button_right,
162           button_left, button_down, button_up;
163     input   [7:0] switch;
164     output  [7:0] led;
165
166     inout  [31:0] user1, user2, user3, user4;
167
168     inout  [43:0] daughtercard;
169
170     inout  [15:0] systemace_data;
171     output [6:0]  systemace_address;
172     output systemace_ce_b, systemace_we_b, systemace_oe_b;
173     input  systemace_irq, systemace_mprbdy;
174
175     output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
176           analyzer4_data;
177     output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;
178
179     ///////////////////////////////////////////////////////////////////
180     //
181     // I/O Assignments
182     //
183     ///////////////////////////////////////////////////////////////////
184
185     // VGA Output
186     assign vga_out_red = 10'h0;
187     assign vga_out_green = 10'h0;
188     assign vga_out_blue = 10'h0;
189     assign vga_out_sync_b = 1'b1;
190     assign vga_out_blank_b = 1'b1;
191     assign vga_out_pixel_clock = 1'b0;
192     assign vga_out_hsync = 1'b0;
193     assign vga_out_vsync = 1'b0;
194
195     // Video Output
196     assign tv_out_ycrcb = 10'h0;
197     assign tv_out_reset_b = 1'b0;
198     assign tv_out_clock = 1'b0;
199     assign tv_out_i2c_clock = 1'b0;
200     assign tv_out_i2c_data = 1'b0;
201     assign tv_out_pal_ntsc = 1'b0;
202     assign tv_out_hsync_b = 1'b1;
203     assign tv_out_vsync_b = 1'b1;
204     assign tv_out_blank_b = 1'b1;
205     assign tv_out_subcar_reset = 1'b0;
206
207     // Video Input
208     assign tv_in_i2c_clock = 1'b0;
209     assign tv_in_fifo_read = 1'b0;
210     assign tv_in_fifo_clock = 1'b0;
211     assign tv_in_iso = 1'b0;
212     assign tv_in_reset_b = 1'b0;
213     assign tv_in_clock = 1'b0;
214     assign tv_in_i2c_data = 1'bZ;
215     // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
216     // tv_in_aef, tv_in_hff, and tv_in_aff are inputs
217
218     // SRAMs
219     assign ram0_data = 36'hZ;
220     assign ram0_address = 19'h0;
221     assign ram0_adv_ld = 1'b0;
222     assign ram0_clk = 1'b0;
223     assign ram0_cen_b = 1'b1;
224     assign ram0_ce_b = 1'b1;
225     assign ram0_oe_b = 1'b1;
226     assign ram0_we_b = 1'b1;
227     assign ram0_bwe_b = 4'hF;
228     assign ram1_data = 36'hZ;
229     assign ram1_address = 19'h0;
230     assign ram1_adv_ld = 1'b0;
231     assign ram1_clk = 1'b0;
232     assign ram1_cen_b = 1'b1;
233     assign ram1_ce_b = 1'b1;
234     assign ram1_oe_b = 1'b1;
235     assign ram1_we_b = 1'b1;
236     assign ram1_bwe_b = 4'hF;
237     assign clock_feedback_out = 1'b0;
```

```
238 // clock_feedback_in is an input
239
240 // User I/Os
241 // assign user1 = 32'hZ;
242 assign user2 = 32'hZ;
243 assign user3 = 32'hZ;
244 assign user4 = 32'hZ;
245
246 // Daughtercard Connectors
247 assign daughtercard = 44'hZ;
248
249 // SystemACE Microprocessor Port
250 assign systemace_data = 16'hZ;
251 assign systemace_address = 7'h0;
252 assign systemace_ce_b = 1'b1;
253 assign systemace_we_b = 1'b1;
254 assign systemace_oe_b = 1'b1;
255 // systemace_irq and systemace_mprdy are inputs
256
257 ///////////////////////////////////////////////////////////////////
258 //
259 // Reset Generation for Wireless Module
260 //
261 // A shift register primitive is used to generate an active-high reset
262 // signal that remains high for 16 clock cycles after configuration finishes
263 // and the FPGA's internal clocks begin toggling.
264 //
265 ///////////////////////////////////////////////////////////////////
266
267 wire reset_pos, reset_wireless;
268 SRL16 reset_sr111 (.D(1'b0), .CLK(clock_27mhz), .Q(reset_pos),
269 .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
270 defparam reset_sr.INIT = 16'hFFFF;
271 assign reset_wireless = ~reset_pos;
272
273
274 ///////////////////////////////////////////////////////////////////
275 //
276 // RS-232 Interface Tester
277 //
278 ///////////////////////////////////////////////////////////////////
279
280 assign rs232_txd = 1'b1;
281 assign rs232_rts = 1'b1;
282
283 wire [7:0] rxdata;
284 wire [5:0] rx_state;
285 wire clk_173khz;
286
287 Clock_173kHz clock_173khz1(clock_27mhz, reset_wireless, clk_173khz);
288 rs232_rx rs232_rx1(reset_wireless, clk_173khz, switch[5], rs232_rxd, rxdata, rx_state);
289 // rs232_rxd and rs232_cts are inputs
290
291
292 ///////////////////////////////////////////////////////////////////
293 //
294 // EKG System
295 //
296 ///////////////////////////////////////////////////////////////////
297
298 wire reset, status, reset_ekg;
299 wire cs_bar, r_wbar, cs_bar_DA, cs_bar_DA_ekg, heartbeat, abnormal;
300 wire[3:0] digit2, digit1, digit0;
301 wire[8:0] heartrate;
302
303 assign status = user1[0];
304 assign reset = switch[1];
305 assign reset_ekg = switch[0];
306
307 top top1(clock1, reset, status, cs_bar, r_wbar, cs_bar_DA, user1[31:24], heartbeat, heartrate,
308 digit2, digit1, digit0, abnormal);
309 ekg ekg1(clock1, reset_ekg, cs_bar_DA_ekg, user4[8:1], rxdata, user4[28:21], user4[20]);
310
311 assign user1[1] = cs_bar;
312 assign user1[2] = r_wbar;
313 assign user1[3] = cs_bar_DA;
314 assign user4[0] = cs_bar_DA_ekg;
315 assign user4[31] = heartbeat;
316
```

```
317
318 ////////////////////////////////////////////////////
319 //
320 // Logic Analyzer
321 //
322 ////////////////////////////////////////////////////
323
324 // Logic Analyzer
325 assign analyzer1_data = {1'b0, rx_state, rs232_rxd, rxdata}; //16 bits
326 assign analyzer1_clock = clk_173khz;
327
328 ////////////////////////////////////////////////////
329 //
330 // Reset Generation for Audio + Flash ROM
331 //
332 // A shift register primitive is used to generate an active-high reset
333 // signal that remains high for 16 clock cycles after configuration finishes
334 // and the FPGA's internal clocks begin toggling.
335 //
336 ////////////////////////////////////////////////////
337
338
339 wire reset_audio;
340 SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(reset_audio),
341               .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
342 defparam reset_sr.INIT = 16'hFFFF;
343
344 ////////////////////////////////////////////////////
345 //
346 // Flash ROM
347 //
348 ////////////////////////////////////////////////////
349
350 wire [22:0] faddress;
351 wire [15:0] fwdata, frdata;
352 wire fbusy;
353
354 flash_int flashint1 (reset_audio, clock_27mhz, 2'b01, faddress, fwdata, frdata,
355                    fbusy, flash_data, flash_address, flash_ce_b,
356                    flash_oe_b, flash_we_b, flash_reset_b, 1'b1,
357                    flash_byte_b);
358
359 ////////////////////////////////////////////////////
360 //
361 // Audio Input and Output
362 //
363 ////////////////////////////////////////////////////
364
365 wire[3:0] vol;
366 wire[39:0] vdisp;
367 wire volume_up, volume_down, heart_beep;
368 //assign reset_audio = ~(switch[2] || switch[3]);
369 assign heart_beep = switch[2] ? (switch[3] ? (abnormal || heartbeat) : abnormal)
370                    : (switch[3] ? heartbeat : 1'b0);
371 audio audiol (reset_audio, clock_27mhz, audio_reset_b, ac97_sdata_out,
372             ac97_sdata_in, ac97_synch, ac97_bit_clock, {vol, 1'b0}, switch[4], frdata, faddress, heartrate);
373 beeper beep1 (reset_audio, clock_27mhz, beep, heart_beep);
374 debounce vol_up (reset_audio, clock_27mhz, button_up, volume_up);
375 debounce vol_down (reset_audio, clock_27mhz, button_down, volume_down);
376 volume voll (reset_audio, clock_27mhz, volume_up, volume_down, vol, vdisp);
377
378 assign led[7:0] = {~abnormal, ~abnormal, ~abnormal, ~abnormal, ~abnormal, ~abnormal, ~abnormal, ~abnormal};
379
380 ////////////////////////////////////////////////////
381 //
382 // Display
383 //
384 ////////////////////////////////////////////////////
385
386 wire[639:0] dots, dots_heart, dots_vol;
387 reg[79:0] volume_dots;
388 wire[39:0] one, two, three, four, five, six, seven, eight, nine, zero, dash, blank, qmark;
389 reg[39:0] digit2_display, digit1_display, digit0_display;
390
391 assign zero = 40'b00111110_01000001_01000001_01000001_00111110;
392 assign one = 40'b00000000_01000010_01111111_01000000_00000000;
393 assign two = 40'b01100010_01010001_01001001_01001001_01000110;
394 assign three = 40'b00100010_01000001_01001001_01001001_00110110;
```

```
396 assign four = 40'b00011000_00010100_00010010_01111111_00010000;
397 assign five = 40'b00100111_01000101_01000101_01000101_00111001;
398 assign six = 40'b00111100_01001010_01001001_01001001_00110000;
399 assign seven = 40'b00000001_01110001_00001001_00000101_00000011;
400 assign eight = 40'b00110110_01001001_01001001_01001001_00110110;
401 assign nine = 40'b00000110_01001001_01001001_00101001_00011110;
402 assign dash = 40'b00001000_00001000_00001000_00001000_00001000;
403 assign blank = 40'b00000000_00000000_00000000_00000000_00000000;
404 assign qmark = 40'b00000010_00000001_01010001_00001001_00000110; //'?'
405
406 assign dots = switch[7] ? dots_heart : dots_vol;
407
408 always @ (digit2 or digit1 or digit0)
409 begin
410     case (digit2)
411         4'b0000: digit2_display <= blank;          4'b0001: digit2_display <= one;
412         4'b0010: digit2_display <= two;          4'b0011: digit2_display <= three;
413         4'b0100: digit2_display <= four;        4'b0101: digit2_display <= five;
414         4'b0110: digit2_display <= six;        4'b0111: digit2_display <= seven;
415         4'b1000: digit2_display <= eight;      4'b1001: digit2_display <= nine;
416         default: digit2_display <= dash;
417     endcase
418
419     case (digit1)
420         4'b0000: digit1_display <= zero;       4'b0001: digit1_display <= one;
421         4'b0010: digit1_display <= two;       4'b0011: digit1_display <= three;
422         4'b0100: digit1_display <= four;     4'b0101: digit1_display <= five;
423         4'b0110: digit1_display <= six;     4'b0111: digit1_display <= seven;
424         4'b1000: digit1_display <= eight;   4'b1001: digit1_display <= nine;
425         default: digit1_display <= dash;
426     endcase
427
428     case (digit0)
429         4'b0000: digit0_display <= zero;     4'b0001: digit0_display <= one;
430         4'b0010: digit0_display <= two;     4'b0011: digit0_display <= three;
431         4'b0100: digit0_display <= four;   4'b0101: digit0_display <= five;
432         4'b0110: digit0_display <= six;   4'b0111: digit0_display <= seven;
433         4'b1000: digit0_display <= eight;  4'b1001: digit0_display <= nine;
434         default: digit0_display <= dash;
435     endcase
436 end
437
438 always @(vol)
439 case (vol)
440     5'd15: volume_dots <= {one, five};
441     5'd14: volume_dots <= {one, four};
442     5'd13: volume_dots <= {one, three};
443     5'd12: volume_dots <= {one, two};
444     5'd11: volume_dots <= {one, one};
445     5'd10: volume_dots <= {one, zero};
446     5'd09: volume_dots <= {blank, nine};
447     5'd08: volume_dots <= {blank, eight};
448     5'd07: volume_dots <= {blank, seven};
449     5'd06: volume_dots <= {blank, six};
450     5'd05: volume_dots <= {blank, five};
451     5'd04: volume_dots <= {blank, four};
452     5'd03: volume_dots <= {blank, three};
453     5'd02: volume_dots <= {blank, two};
454     5'd01: volume_dots <= {blank, one};
455     5'd00: volume_dots <= {blank, zero};
456     default: volume_dots <= {blank, qmark};
457 endcase
458
459 assign dots_heart = { 40'b01111111_00001000_00001000_00001000_01111111, // 'H'
460     40'b01111111_01001001_01001001_01001001_01000001, // 'E'
461     40'b01111100_00010010_00010001_00010010_01111100, // 'A'
462     40'b01111111_00001001_00011001_00101001_01000110, // 'R'
463     40'b00000001_00000001_01111111_00000001_00000001, // 'T'
464     blank,
465     40'b01111111_00001001_00011001_00101001_01000110, // 'R'
466     40'b01111100_00010010_00010001_00010010_01111100, // 'A'
467     40'b00000001_00000001_01111111_00000001_00000001, // 'T'
468     40'b01111111_01001001_01001001_01001001_01000001, // 'E'
469     40'b00000000_00110110_00110110_00000000_00000000, // ':'
470     blank,
471     digit2_display,
472     digit1_display,
473     digit0_display,
474     blank
```

```
475     };
476
477     assign dots_vol = { 40'b00000111_00011000_01100000_00011000_00000111, // 'V'
478                       40'b00111110_01000001_01000001_01000001_00111110, // 'O'
479                       40'b01111111_01000000_01000000_01000000_01000000, // 'L'
480                       40'b00111111_01000000_01000000_01000000_00111111, // 'U'
481                       40'b01111111_00000010_00001100_00000010_01111111, // 'M'
482                       40'b01111111_01001001_01001001_01001001_01000001, // 'E'
483                       40'b00000000_00110110_00110110_00000000_00000000, // ':'
484                       40'b00000000_00000000_00000000_00000000_00000000, // ' '
485                       volume_dots,
486                       blank, blank, blank, blank, blank
487     };
488
489     display disp(1'b0, clock_27mhz, disp_blank, disp_clock, disp_rs,
490               disp_ce_b, disp_reset_b, disp_data_out, dots);
491
492 endmodule
```

```
1 module majorFSM(clk, reset_sync, sample, filter_busy, analog_busy, filter_start, DAC_start);
2
3 input clk, reset_sync, sample, filter_busy, analog_busy;
4 output filter_start, DAC_start;
5 reg DAC_start, filter_start;
6 reg[2:0] state, nextstate;
7
8 parameter IDLE = 0;
9 parameter WAIT = 1;
10 parameter DO_DAC0 = 2;
11 parameter DO_DAC1 = 3;
12 parameter DO_DAC2 = 4;
13 parameter DO_MEM0 = 5;
14 parameter DO_MEM1 = 6;
15 parameter DO_MEM2 = 7;
16
17 always @ (posedge clk)
18 begin
19     if (!reset_sync) state <= IDLE;
20     else state <= nextstate;
21 end
22
23 always @ (state or sample or analog_busy or filter_busy)
24 begin
25     DAC_start = 0;
26     filter_start = 0;
27
28     case (state)
29         IDLE: begin
30             nextstate = WAIT;
31         end
32
33         WAIT: begin
34             if (sample) nextstate = DO_DAC0;
35             else nextstate = WAIT;
36         end
37
38         DO_DAC0: begin
39             if (!analog_busy) nextstate = DO_DAC1;
40             else nextstate = DO_DAC0;
41         end
42
43         DO_DAC1: begin
44             DAC_start = 1;
45             if (analog_busy) nextstate = DO_DAC2;
46             else nextstate = DO_DAC1;
47         end
48
49         DO_DAC2: begin
50             if (!analog_busy) nextstate = DO_MEM0;
51             else nextstate = DO_DAC2;
52         end
53
54         DO_MEM0: begin
55             if (!filter_busy) nextstate = DO_MEM1;
56             else nextstate = DO_MEM0;
57         end
58
59         DO_MEM1: begin
60             filter_start = 1;
61             if (filter_busy) nextstate = DO_MEM2;
62             else nextstate = DO_MEM1;
63         end
64
65         DO_MEM2: begin
66             if (!filter_busy && !analog_busy) nextstate = WAIT;
67             else nextstate = DO_MEM2;
68         end
69
70         default: nextstate = IDLE;
71     endcase
72 end
73
74 endmodule
```

```
1 // 8bits*8bits Multiplier
2 // x is in twos complement format
3 // y is in sign magnitude format
4
5 module multiplier(x, y, z);
6
7 input[7:0] x, y;
8 // 16-bit output
9 // the biggest output magnitude is 2^14 (15 bits)
10 output[15:0] z;
11 wire[7:0] x_neg, y_neg;
12 wire[15:0] z_pos, z_neg, z;
13
14 // Use unsigned multiplication operation * in Verilog
15 // then account for the sign on the MSB
16 assign x_neg = (x == 8'b0) ? 8'b0 : (~x + 1);
17 assign y_neg = (y == 8'b0 || y == 8'b10000000) ? 8'b0 : (~y[7], y[6:0]);
18 assign z_pos = (x[7] ? x_neg : x) * (y[7] ? y_neg : y);
19 assign z_neg = (z_pos == 8'b0) ? 8'b0 : (~z_pos + 1);
20 assign z = (x[7] ^ y[7]) ? z_neg : z_pos;
21
22 endmodule
```

```
1 // 8bits*8bits Multiplier
2 // x,y,z are in twos complement format
3
4 module multiplier_2cpmt(x, y, z);
5
6 input[7:0] x, y;
7 // 16-bit output
8 // the biggest output magnitude is 2^14 (15 bits)
9 output[15:0] z;
10 wire[7:0] x_neg, y_neg;
11 wire[15:0] z_pos, z_neg, z;
12
13 // Use unsigned multiplication operation * in Verilog
14 // then account for the sign on the MSB
15 assign x_neg = (x == 8'b0) ? 8'b0 : (~x + 1);
16 assign y_neg = (y == 8'b0 || y == 8'b10000000) ? 8'b0 : (~y + 1);
17 assign z_pos = (x[7] ? x_neg : x) * (y[7] ? y_neg : y);
18 assign z_neg = (z_pos == 8'b0) ? 8'b0 : (~z_pos + 1);
19 assign z = (x[7] ^ y[7]) ? z_neg : z_pos;
20
21 endmodule
```



```
1 /*****
2 * This file is owned and controlled by Xilinx and must be used *
3 * solely for design, simulation, implementation and creation of *
4 * design files limited to Xilinx devices or technologies. Use *
5 * with non-Xilinx devices or technologies is expressly prohibited *
6 * and immediately terminates your license. *
7 *
8 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" *
9 * SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR *
10 * XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION *
11 * AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION *
12 * OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS *
13 * IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, *
14 * AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE *
15 * FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY *
16 * WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE *
17 * IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR *
18 * REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF *
19 * INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS *
20 * FOR A PARTICULAR PURPOSE. *
21 *
22 * Xilinx products are not intended for use in life support *
23 * appliances, devices, or systems. Use in such applications are *
24 * expressly prohibited. *
25 *
26 * (c) Copyright 1995-2004 Xilinx, Inc. *
27 * All rights reserved. *
28 *****/
29 // The synopsys directives "translate_off/translate_on" specified below are
30 // supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
31 // tools. Ensure they are correct for your synthesis tool(s).
32
33 // You must compile the wrapper file rom1.v when simulating
34 // the core, rom1. When compiling the wrapper file, be sure to
35 // reference the XilinxCoreLib Verilog simulation library. For detailed
36 // instructions, please refer to the "CORE Generator Guide".
37
38 module rom1 (
39     A,
40     SPO); // synthesis black_box
41
42 input [5 : 0] A;
43 output [7 : 0] SPO;
44
45 // synopsys translate_off
46
47 C_DIST_MEM_V7_1 #(
48     6, // c_addr_width
49     "0", // c_default_data
50     1, // c_default_data_radix
51     64, // c_depth
52     1, // c_enable_rlocs
53     1, // c_generate_mif
54     0, // c_has_clk
55     0, // c_has_d
56     0, // c_has_dpo
57     0, // c_has_dpri
58     0, // c_has_i_ce
59     0, // c_has_qdpo
60     0, // c_has_qdpo_ce
61     0, // c_has_qdpo_clk
62     0, // c_has_qdpo_rst
63     0, // c_has_qdpo_srst
64     0, // c_has_qspo
65     0, // c_has_qspo_ce
66     0, // c_has_qspo_rst
67     0, // c_has_qspo_srst
68     0, // c_has_rd_en
69     1, // c_has_spo
70     0, // c_has_spra
71     0, // c_has_we
72     0, // c_latency
73     "rom1.mif", // c_mem_init_file
74     0, // c_mem_type
75     0, // c_mux_type
76     0, // c_qce_joined
77     0, // c_qualify_we
78     1, // c_read_mif
79     0, // c_reg_a_d_inputs
```

```
80     0, // c_reg_dpra_input
81     0, // c_sync_enable
82     8) // c_width
83 inst (
84     .A(A),
85     .SPO(SPO),
86     .D(),
87     .DPRP(),
88     .SPRA(),
89     .I_CE(),
90     .QSPO_CE(),
91     .WE(),
92     .CLK(),
93     .QDPO_CE(),
94     .QDPO_CLK(),
95     .RD_EN(),
96     .QSPO_RST(),
97     .QDPO_RST(),
98     .QSPO_SRST(),
99     .QDPO_SRST(),
100    .QSPO(),
101    .DPO(),
102    .QDPO());
103
104
105 // synopsys translate_on
106
107 // FPGA Express black box declaration
108 // synopsys attribute fpga_dont_touch "true"
109 // synthesis attribute fpga_dont_touch of rom1 is "true"
110
111 // XST black box declaration
112 // box_type "black_box"
113 // synthesis attribute box_type of rom1 is "black_box"
114
115 endmodule
116
117
```

```
1 /*****
2 * This file is owned and controlled by Xilinx and must be used *
3 * solely for design, simulation, implementation and creation of *
4 * design files limited to Xilinx devices or technologies. Use *
5 * with non-Xilinx devices or technologies is expressly prohibited *
6 * and immediately terminates your license. *
7 *
8 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" *
9 * SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR *
10 * XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION *
11 * AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION *
12 * OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS *
13 * IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, *
14 * AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE *
15 * FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY *
16 * WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE *
17 * IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR *
18 * REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF *
19 * INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS *
20 * FOR A PARTICULAR PURPOSE. *
21 *
22 * Xilinx products are not intended for use in life support *
23 * appliances, devices, or systems. Use in such applications are *
24 * expressly prohibited. *
25 *
26 * (c) Copyright 1995-2004 Xilinx, Inc. *
27 * All rights reserved. *
28 *****/
29 // The synopsys directives "translate_off/translate_on" specified below are
30 // supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
31 // tools. Ensure they are correct for your synthesis tool(s).
32
33 // You must compile the wrapper file rom2.v when simulating
34 // the core, rom2. When compiling the wrapper file, be sure to
35 // reference the XilinxCoreLib Verilog simulation library. For detailed
36 // instructions, please refer to the "CORE Generator Guide".
37
38 module rom2 (
39     A,
40     SPO); // synthesis black_box
41
42 input [5 : 0] A;
43 output [7 : 0] SPO;
44
45 // synopsys translate_off
46
47 C_DIST_MEM_V7_1 #(
48     6, // c_addr_width
49     "0", // c_default_data
50     1, // c_default_data_radix
51     64, // c_depth
52     1, // c_enable_rlocs
53     1, // c_generate_mif
54     0, // c_has_clk
55     0, // c_has_d
56     0, // c_has_dpo
57     0, // c_has_dpri
58     0, // c_has_i_ce
59     0, // c_has_qdpo
60     0, // c_has_qdpo_ce
61     0, // c_has_qdpo_clk
62     0, // c_has_qdpo_rst
63     0, // c_has_qdpo_srst
64     0, // c_has_qspo
65     0, // c_has_qspo_ce
66     0, // c_has_qspo_rst
67     0, // c_has_qspo_srst
68     0, // c_has_rd_en
69     1, // c_has_spo
70     0, // c_has_spra
71     0, // c_has_we
72     0, // c_latency
73     "rom2.mif", // c_mem_init_file
74     0, // c_mem_type
75     0, // c_mux_type
76     0, // c_qce_joined
77     0, // c_qualify_we
78     1, // c_read_mif
79     0, // c_reg_a_d_inputs
```

```
80     0, // c_reg_dpra_input
81     0, // c_sync_enable
82     8) // c_width
83 inst (
84     .A(A),
85     .SPO(SPO),
86     .D(),
87     .DPRP(),
88     .SPRA(),
89     .I_CE(),
90     .QSPO_CE(),
91     .WE(),
92     .CLK(),
93     .QDPO_CE(),
94     .QDPO_CLK(),
95     .RD_EN(),
96     .QSPO_RST(),
97     .QDPO_RST(),
98     .QSPO_SRST(),
99     .QDPO_SRST(),
100    .QSPO(),
101    .DPO(),
102    .QDPO());
103
104
105 // synopsys translate_on
106
107 // FPGA Express black box declaration
108 // synopsys attribute fpga_dont_touch "true"
109 // synthesis attribute fpga_dont_touch of rom2 is "true"
110
111 // XST black box declaration
112 // box_type "black_box"
113 // synthesis attribute box_type of rom2 is "black_box"
114
115 endmodule
116
117
```

```
1 /*****
2 * This file is owned and controlled by Xilinx and must be used *
3 * solely for design, simulation, implementation and creation of *
4 * design files limited to Xilinx devices or technologies. Use *
5 * with non-Xilinx devices or technologies is expressly prohibited *
6 * and immediately terminates your license. *
7 *
8 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" *
9 * SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR *
10 * XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION *
11 * AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION *
12 * OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS *
13 * IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, *
14 * AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE *
15 * FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY *
16 * WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE *
17 * IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR *
18 * REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF *
19 * INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS *
20 * FOR A PARTICULAR PURPOSE. *
21 *
22 * Xilinx products are not intended for use in life support *
23 * appliances, devices, or systems. Use in such applications are *
24 * expressly prohibited. *
25 *
26 * (c) Copyright 1995-2004 Xilinx, Inc. *
27 * All rights reserved. *
28 *****/
29 // The synopsys directives "translate_off/translate_on" specified below are
30 // supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
31 // tools. Ensure they are correct for your synthesis tool(s).
32
33 // You must compile the wrapper file romsound.v when simulating
34 // the core, romsound. When compiling the wrapper file, be sure to
35 // reference the XilinxCoreLib Verilog simulation library. For detailed
36 // instructions, please refer to the "CORE Generator Guide".
37
38 module romsound (
39     A,
40     SPO); // synthesis black_box
41
42 input [12 : 0] A;
43 output [7 : 0] SPO;
44
45 // synopsys translate_off
46
47 C_DIST_MEM_V7_1 #(
48     13, // c_addr_width
49     "0", // c_default_data
50     1, // c_default_data_radix
51     8192, // c_depth
52     1, // c_enable_rlocs
53     1, // c_generate_mif
54     0, // c_has_clk
55     0, // c_has_d
56     0, // c_has_dpo
57     0, // c_has_dpri
58     0, // c_has_i_ce
59     0, // c_has_qdpo
60     0, // c_has_qdpo_ce
61     0, // c_has_qdpo_clk
62     0, // c_has_qdpo_rst
63     0, // c_has_qdpo_srst
64     0, // c_has_qsop
65     0, // c_has_qsop_ce
66     0, // c_has_qsop_rst
67     0, // c_has_qsop_srst
68     0, // c_has_rd_en
69     1, // c_has_spo
70     0, // c_has_spra
71     0, // c_has_we
72     0, // c_latency
73     "romsound.mif", // c_mem_init_file
74     0, // c_mem_type
75     0, // c_mux_type
76     0, // c_qce_joined
77     0, // c_qualify_we
78     1, // c_read_mif
79     0, // c_reg_a_d_inputs
```

```
80     0, // c_reg_dpra_input
81     0, // c_sync_enable
82     8) // c_width
83 inst (
84     .A(A),
85     .SPO(SPO),
86     .D(),
87     .DPRA(),
88     .SPRA(),
89     .I_CE(),
90     .QSPO_CE(),
91     .WE(),
92     .CLK(),
93     .QDPO_CE(),
94     .QDPO_CLK(),
95     .RD_EN(),
96     .QSPO_RST(),
97     .QDPO_RST(),
98     .QSPO_SRST(),
99     .QDPO_SRST(),
100    .QSPO(),
101    .DPO(),
102    .QDPO());
103
104
105 // synopsys translate_on
106
107 // FPGA Express black box declaration
108 // synopsys attribute fpga_dont_touch "true"
109 // synthesis attribute fpga_dont_touch of romsound is "true"
110
111 // XST black box declaration
112 // box_type "black_box"
113 // synthesis attribute box_type of romsound is "black_box"
114
115 endmodule
116
117
```

```
1 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 //
3 // 6.111 FPGA Labkit -- RS232 State Machine
4 //
5 //
6 // Created: May 9, 2005
7 // Author: Amy Tang
8 //
9 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
10 `define STATUS_RESET          4'h0
11 `define STATUS_READ_ID       4'h1
12 `define STATUS_CLEAR_LOCKS   4'h2
13
14
15 module rs232_rx (reset, clock, sw5, rs232_rxd, rxdata, state);
16
17     input reset, clock;
18     input rs232_rxd, sw5;
19     output [7:0] rxdata;
20     output [5:0] state;
21
22
23     reg [7:0] rxdata;
24     reg [2:0] ptr; // serial bit ptr
25     reg b0, b1, b2, b3, b4, b5, b6, b7;
26
27 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
28 //
29 // State Machine
30 //
31 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
32
33 // internal state
34     reg [5:0] state;
35
36 parameter IDLE = 0;
37 parameter START1 = 1;
38 parameter START2 = 2;
39 parameter WRITE1 = 3;
40 parameter WRITE2 = 4;
41 parameter WRITE3 = 5;
42 parameter STOP1 = 6;
43 parameter STOP2 = 7;
44 parameter STOP3 = 8;
45
46 /*initial begin
47     state <= IDLE;
48     rxdata <= 0; //rxd changes once all 8 bits are received
49     ptr<= 0;
50     b0 <= 0;    b1 <= 0; b2 <= 0;    b3 <= 0;
51     b4 <= 0;    b5 <= 0; b6 <= 0;    b7 <= 0;
52 end*/
53
54 // Combination Block for always next block
55 always @ (posedge clock or negedge reset) begin
56     if (!reset) begin
57         state <= IDLE;
58         rxdata <= 0;
59         ptr <= 0;
60         b0 <= 0;    b1 <= 0; b2 <= 0;    b3 <= 0;
61         b4 <= 0;    b5 <= 0; b6 <= 0;    b7 <= 0;
62     end
63     else if (sw5)
64     case (state)
65         IDLE:
66             begin
67                 if (!rs232_rxd) state <= START1; // start bit detected; active low
68                 else state <= IDLE;
69             end
70
71         START1:
72             begin
73                 state <= START2;
74             end
75         START2:
76             begin
77                 state <= WRITE1;
78             end
79     end
```

```
80     WRITE1:
81         begin
82             state <= WRITE2;
83         end
84
85     WRITE2:
86         begin
87             if (ptr == 0) b0 <= rs232_rxd;
88             else if (ptr == 1) b1 <= rs232_rxd;
89             else if (ptr == 2) b2 <= rs232_rxd;
90             else if (ptr == 3) b3 <= rs232_rxd;
91             else if (ptr == 4) b4 <= rs232_rxd;
92             else if (ptr == 5) b5 <= rs232_rxd;
93             else if (ptr == 6) b6 <= rs232_rxd;
94             else if (ptr == 7) b7 <= rs232_rxd;
95
96             state <= WRITE3;
97         end
98
99     WRITE3:
100        begin
101            if (ptr == 3'd7) begin
102                rxdata <= {b7, b6, b5, b4, b3, b2, b1, b0};
103                ptr <= 0;
104                state <= STOP1;
105            end
106            else begin
107                ptr <= ptr +1;
108                state <= WRITE1;
109            end
110        end
111    STOP1:
112        begin
113            state <= STOP2;
114        end
115
116    STOP2:
117        begin
118            state <= STOP3;
119        end
120
121    STOP3:
122        begin
123            state <= IDLE;
124        end
125    default: state <= IDLE;
126 endcase // case(state)
127 end // always @ (clock)
128 endmodule
129
```



```
1 /*****
2 * This file is owned and controlled by Xilinx and must be used *
3 * solely for design, simulation, implementation and creation of *
4 * design files limited to Xilinx devices or technologies. Use *
5 * with non-Xilinx devices or technologies is expressly prohibited *
6 * and immediately terminates your license. *
7 *
8 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" *
9 * SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR *
10 * XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION *
11 * AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION *
12 * OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS *
13 * IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, *
14 * AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE *
15 * FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY *
16 * WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE *
17 * IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR *
18 * REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF *
19 * INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS *
20 * FOR A PARTICULAR PURPOSE. *
21 *
22 * Xilinx products are not intended for use in life support *
23 * appliances, devices, or systems. Use in such applications are *
24 * expressly prohibited. *
25 *
26 * (c) Copyright 1995-2004 Xilinx, Inc. *
27 * All rights reserved. *
28 *****/
29 // The synopsys directives "translate_off/translate_on" specified below are
30 // supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
31 // tools. Ensure they are correct for your synthesis tool(s).
32
33 // You must compile the wrapper file sram.v when simulating
34 // the core, sram. When compiling the wrapper file, be sure to
35 // reference the XilinxCoreLib Verilog simulation library. For detailed
36 // instructions, please refer to the "CORE Generator Guide".
37
38 module sram (
39     A,
40     CLK,
41     D,
42     WE,
43     SPO); // synthesis black_box
44
45 input [5 : 0] A;
46 input CLK;
47 input [7 : 0] D;
48 input WE;
49 output [7 : 0] SPO;
50
51 // synopsys translate_off
52
53 C_DIST_MEM_V7_1 #(
54     6, // c_addr_width
55     "0", // c_default_data
56     1, // c_default_data_radix
57     64, // c_depth
58     1, // c_enable_rlocs
59     1, // c_generate_mif
60     1, // c_has_clk
61     1, // c_has_d
62     0, // c_has_dpo
63     0, // c_has_dpra
64     0, // c_has_i_ce
65     0, // c_has_qdpo
66     0, // c_has_qdpo_ce
67     0, // c_has_qdpo_clk
68     0, // c_has_qdpo_rst
69     0, // c_has_qdpo_srst
70     0, // c_has_qspos
71     0, // c_has_qspos_ce
72     0, // c_has_qspos_rst
73     0, // c_has_qspos_srst
74     0, // c_has_rd_en
75     1, // c_has_spo
76     0, // c_has_spra
77     1, // c_has_we
78     0, // c_latency
79     "sram.mif", // c_mem_init_file
```

```
80     1, // c_mem_type
81     0, // c_mux_type
82     0, // c_qce_joined
83     0, // c_qualify_we
84     0, // c_read_mif
85     0, // c_reg_a_d_inputs
86     0, // c_reg_dpra_input
87     0, // c_sync_enable
88     8) // c_width
89 inst (
90     .A(A),
91     .CLK(CLK),
92     .D(D),
93     .WE(WE),
94     .SPO(SPO),
95     .DPRA(),
96     .SPRA(),
97     .I_CE(),
98     .QSPO_CE(),
99     .QDPO_CE(),
100    .QDPO_CLK(),
101    .RD_EN(),
102    .QSPO_RST(),
103    .QDPO_RST(),
104    .QSPO_SRST(),
105    .QDPO_SRST(),
106    .QSPO(),
107    .DPO(),
108    .QDPO());
109
110
111 // synopsys translate_on
112
113 // FPGA Express black box declaration
114 // synopsys attribute fpga_dont_touch "true"
115 // synthesis attribute fpga_dont_touch of sram is "true"
116
117 // XST black box declaration
118 // box_type "black_box"
119 // synthesis attribute box_type of sram is "black_box"
120
121 endmodule
122
123
```

```
1 module synchronizer(clk, reset, status, reset_sync, status_sync);
2
3 input clk, reset, status;
4 output reset_sync, status_sync;
5 reg x1, x2, reset_sync, status_sync;
6
7 always @ (posedge clk)
8 begin
9     x1 <= reset;
10    x2 <= status;
11    reset_sync <= x1;
12    status_sync <= x2;
13 end
14
15 endmodule
```

```
1 ///////////////////////////////////////////////////////////////////
2 //
3 // 6.111 FPGA Labkit -- Flash Tester State Machine
4 //
5 //
6 // Created: January 23, 2005
7 // Author: Nathan Ickes
8 //
9 ///////////////////////////////////////////////////////////////////
10
11 `define FLASHOP_IDLE 2'b00
12 `define FLASHOP_READ 2'b01
13 `define FLASHOP_WRITE 2'b10
14
15 `define STATUS_RESET 4'h0
16 `define STATUS_READ_ID 4'h1
17 `define STATUS_CLEAR_LOCKS 4'h2
18 `define STATUS_ERASING 4'h3
19 `define STATUS_WRITING 4'h4
20 `define STATUS_READING 4'h5
21 `define STATUS_SUCCESS 4'h6
22 `define STATUS_BAD_MANUFACTURER 4'h7
23 `define STATUS_BAD_SIZE 4'h8
24 `define STATUS_LOCK_BIT_ERROR 4'h9
25 `define STATUS_ERASE_BLOCK_ERROR 4'ha
26 `define STATUS_WRITE_ERROR 4'hB
27 `define STATUS_READ_WRONG_DATA 4'hC
28
29 //`define NUM_BLOCKS 128
30 `define NUM_BLOCKS 128
31 `define BLOCK_SIZE 64*1024
32 `define LAST_BLOCK_ADDRESS ((`NUM_BLOCKS-1)*`BLOCK_SIZE)
33 `define LAST_ADDRESS (`NUM_BLOCKS*`BLOCK_SIZE-1)
34
35 `define NUM_BLOCKS2 2
36 `define BLOCK_SIZE2 64*1024
37 `define LAST_BLOCK_ADDRESS2 ((`NUM_BLOCKS2-1)*`BLOCK_SIZE2)
38 `define LAST_ADDRESS2 (`NUM_BLOCKS2*`BLOCK_SIZE2-1)
39
40
41 `define INIT_FLASH_ADDRESS 23'd57351 // 23'h0, 23'd8192
42 `define LAST_FLASH_ADDRESS (`INIT_FLASH_ADDRESS+23'd8192)
43
44
45 module test_fsm (reset, clock, fop, faddress, fwdata, frdata, fbusy, dots, FPGA_ROM_data, FPGA_ROM_addr);
46
47     input reset, clock;
48     output [1:0] fop;
49     output [22:0] faddress;
50     output [15:0] fwdata;
51     input [15:0] frdata;
52     input fbusy;
53     output [639:0] dots;
54     input [7:0] FPGA_ROM_data;
55     output [12:0] FPGA_ROM_addr;
56
57     reg [1:0] fop;
58     reg [22:0] faddress;
59     reg [15:0] fwdata;
60     reg [639:0] dots;
61     reg [12:0] FPGA_ROM_addr;
62
63     ///////////////////////////////////////////////////////////////////
64     //
65     // State Machine
66     //
67     ///////////////////////////////////////////////////////////////////
68
69     reg [7:0] state;
70     reg [3:0] status;
71
72     always @(posedge clock)
73     if (reset)
74     begin
75         state <= 0;
76         status <= `STATUS_RESET;
77         faddress <= 0;
78         FPGA_ROM_addr <= 0;
79         fop <= `FLASHOP_IDLE;
```

```
80     end
81     else if (!fbusy && (fop == `FLASHOP_IDLE))
82     case (state)
83 8'h00:
84     begin
85         // Issue "read id codes" command
86         status <= `STATUS_READ_ID;
87         faddress <= 0;
88         fwdata <= 16'h0090;
89         fop <= `FLASHOP_WRITE;
90         state <= state+1;
91     end
92
93 8'h01:
94     begin
95         // Read manufacturer code
96         faddress <= 0;
97         fop <= `FLASHOP_READ;
98         state <= state+1;
99     end
100
101 8'h02:
102     if (frdata != 16'h0089) // 16'h0089 = Intel
103         status <= `STATUS_BAD_MANUFACTURER;
104     else
105     begin
106         // Read the device size code
107         faddress <= 1;
108         fop <= `FLASHOP_READ;
109         state <= state+1;
110     end
111
112 8'h03:
113     if (frdata != 16'h0018) // 16'h0018 = 128Mbit
114         status <= `STATUS_BAD_SIZE;
115     else
116     begin
117         faddress <= 0;
118         fwdata <= 16'hFF;
119         fop <= `FLASHOP_WRITE;
120         state <= state+1;
121     end
122 8'h04:
123     begin
124         // Issue "clear lock bits" command
125         status <= `STATUS_CLEAR_LOCKS;
126         faddress <= 0;
127         fwdata <= 16'h60;
128         fop <= `FLASHOP_WRITE;
129         state <= state+1;
130     end
131 8'h05:
132     begin
133         // Issue "confirm clear lock bits" command
134         faddress <= 0;
135         fwdata <= 16'hD0;
136         fop <= `FLASHOP_WRITE;
137         state <= state+1;
138     end
139 8'h06:
140     begin
141         // Read status
142         faddress <= 0;
143         fop <= `FLASHOP_READ;
144         state <= state+1;
145     end
146 8'h07:
147     if (frdata[7] == 1) // Done clearing lock bits
148         if (frdata[6:1] == 0) // No errors
149         begin
150             faddress <= `INIT_FLASH_ADDRESS;
151             fop <= `FLASHOP_IDLE;
152             state <= state+5;
153         end
154     else
155         status <= `STATUS_LOCK_BIT_ERROR;
156     else // Still busy, reread status register
157     begin
158         faddress <= 0;
```

```

159     fop <= `FLASHOP_READ;
160     end
161 */
162 ////////////////////////////////////////////////////
163 // Block Erase Sequence
164 ////////////////////////////////////////////////////
165
166 8'h08:
167     begin
168         status <= `STATUS_ERASING;
169         fwdata <= 16'h20; // Issue "erase block" command
170         fop <= `FLASHOP_WRITE;
171         state <= state+1;
172     end
173 8'h09:
174     begin
175         fwdata <= 16'hD0; // Issue "confirm erase" command
176         fop <= `FLASHOP_WRITE;
177         state <= state+1;
178     end
179 8'h0A:
180     begin
181         fop <= `FLASHOP_READ;
182         state <= state+1;
183     end
184 8'h0B:
185     if (frdata[7] == 1) // Done erasing block
186         if (frdata[6:1] == 0) // No errors
187             if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS
188                 //if (faddress != `LAST_BLOCK_ADDRESS) // `LAST_BLOCK_ADDRESS)
189                 //if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS)
190                 //if (faddress != 23'h100) // `LAST_BLOCK_ADDRESS)
191             begin
192                 faddress <= faddress+`BLOCK_SIZE;
193                 fop <= `FLASHOP_IDLE;
194                 state <= state-3;
195             end
196             else
197             begin
198                 faddress <= `INIT_FLASH_ADDRESS;
199                 FPGA_ROM_addr <= 0;
200                 fop <= `FLASHOP_IDLE;
201                 state <= state+1;
202             end
203             else // Erase error detected
204                 status <= `STATUS_ERASE_BLOCK_ERROR;
205         else // Still busy
206             fop <= `FLASHOP_READ;
207
208 */
209 ////////////////////////////////////////////////////
210 // Write Addresses to All Locations
211 ////////////////////////////////////////////////////
212
213 8'h0C:
214     begin
215         status <= `STATUS_WRITING;
216         fwdata <= 16'h40; // Issue "setup write" command
217         fop <= `FLASHOP_WRITE;
218         state <= state+1;
219     end
220
221 8'h0D:
222     begin
223         //fwdata <= 16'b10101; // Finish write
224         fwdata <= {8'b0, FPGA_ROM_data}; // Finish write
225         fop <= `FLASHOP_WRITE;
226         state <= state+1;
227     end
228 8'h0E:
229     begin
230         // Read status register
231         fop <= `FLASHOP_READ;
232         state <= state+1;
233     end
234 8'h0F:
235     if (frdata[7] == 1) // Done writing
236         if (frdata[6:1] == 0) // No errors
237             if (faddress != `LAST_FLASH_ADDRESS) // `LAST_ADDRESS)

```

```
238 //if (faddress != {20'b0, FPGA_ROM_addr}) // `LAST_ADDRESS)
239 //if (faddress != 23'h7FFFFF) // `LAST_ADDRESS) , if (faddress != 23'h20000)
240 begin
241     faddress <= faddress+1;
242     FPGA_ROM_addr <= FPGA_ROM_addr+1;
243     fop <= `FLASHOP_IDLE;
244     state <= state-3;
245 end
246 else
247 begin
248     faddress <= `INIT_FLASH_ADDRESS;
249     FPGA_ROM_addr <= 0;
250     fop <= `FLASHOP_IDLE;
251     state <= state+1;
252 end
253 else // Write error detected
254     status <= `STATUS_WRITE_ERROR;
255 else // Still busy
256     fop <= `FLASHOP_READ;
257
258 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
259 // Read back data
260 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
261
262 8'h10:
263 begin
264     status <= `STATUS_READING;
265     fwdata <= 16'hFF; // Issue "read array" command
266     fop <= `FLASHOP_WRITE;
267     faddress <= `INIT_FLASH_ADDRESS;
268     FPGA_ROM_addr <= 0;
269     state <= state+1;
270 end
271 8'h11:
272 begin
273     fop <= `FLASHOP_READ;
274     state <= state+1;
275 end
276 8'h12:
277 if (frdata == {8'b0, FPGA_ROM_data})
278     if (faddress == `LAST_FLASH_ADDRESS)
279 //if (faddress == `LAST_ADDRESS2)
280 //if (faddress == 23'h7FFFFF)
281     begin
282         fop <= `FLASHOP_IDLE;
283         faddress <= `INIT_FLASH_ADDRESS;
284         FPGA_ROM_addr <= 0;
285         state <= state+5;
286     end
287     else
288     begin
289         faddress <= faddress+1;
290         FPGA_ROM_addr <= FPGA_ROM_addr+1;
291         fop <= `FLASHOP_READ;
292     end
293 else
294     status <= `STATUS_READ_WRONG_DATA;
295 /*
296 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
297 // Erase the chip again
298 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
299
300 8'h13:
301 begin
302     status <= `STATUS_ERASING;
303     fwdata <= 16'h20; // Issue "erase block" command
304     fop <= `FLASHOP_WRITE;
305     state <= state+1;
306 end
307 8'h14:
308 begin
309     fwdata <= 16'hD0; // Issue "confirm erase" command
310     fop <= `FLASHOP_WRITE;
311     state <= state+1;
312 end
313 8'h15:
314 begin
315     fop <= `FLASHOP_READ;
316     state <= state+1;
```

```

317     end
318     8'h16:
319     if (frdata[7] == 1) // Done erasing block
320     if (frdata[6:1] == 0) // No errors
321     if (faddress != 23'h20000) // `LAST_BLOCK_ADDRESS)
322     //if (faddress != `LAST_BLOCK_ADDRESS) // `LAST_BLOCK_ADDRESS)
323     // if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS)
324     // if (faddress != 23'h100) // `LAST_BLOCK_ADDRESS)
325     begin
326     faddress <= faddress+`BLOCK_SIZE;
327     fop <= `FLASHOP_IDLE;
328     state <= state-3;
329     end
330     else
331     begin
332     faddress <= 0;
333     fop <= `FLASHOP_IDLE;
334     state <= state+1;
335     end
336     else // Erase error detected
337     status <= `STATUS_ERASE_BLOCK_ERROR;
338     else // Still busy
339     fop <= `FLASHOP_READ;
340 */
341 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
342 // End of test: declare success
343 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
344
345 8'h17:
346     begin
347     status <= `STATUS_SUCCESS;
348     fop <= `FLASHOP_IDLE;
349     end
350
351     endcase
352     else
353     fop <= `FLASHOP_IDLE;
354
355 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
356 //
357 // Status display
358 //
359 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
360 //
361 // "Reset    -----" --> During reset
362 // "Read ID  000000" --> While reading ID codes
363 // "Clr locks 000000" --> While clearing block locks
364 // "Erase    000000" --> While erasing
365 // "Write    000000" --> While writing
366 // "Read     000000" --> While reading
367 // " *** PASSED *** " --> If the entire test completes with no errors
368 // "Err: Manuf 0000" --> If an incorrect manufacturer code is read
369 // "Err: Size  0000" --> If an incorrect size code is read
370 // "Err: Locks 0000" --> If an error is detected when clearing the block lock bits
371 // "Err: Erase 0000"
372 // "Err: 000000 0000"
373
374
375 // Rd  000000  0000
376 // Wr  000000  0000
377 // Id  000000  0000
378 //
379
380 function [39:0] nib2char;
381     input [3:0] nib;
382     begin
383     case (nib)
384     4'h0: nib2char = 40'b00111110_01010001_01001001_01000101_00111110;
385     4'h1: nib2char = 40'b00000000_01000010_01111111_01000000_00000000;
386     4'h2: nib2char = 40'b01100010_01010001_01001001_01001001_01000110;
387     4'h3: nib2char = 40'b00100010_01000001_01001001_01001001_00110110;
388     4'h4: nib2char = 40'b00011000_00010100_00010010_01111111_00010000;
389     4'h5: nib2char = 40'b00100111_01000101_01000101_01000101_00111100;
390     4'h6: nib2char = 40'b00111100_01001010_01001001_01001001_00110000;
391     4'h7: nib2char = 40'b00000001_01110001_00001001_00000101_00000011;
392     4'h8: nib2char = 40'b00110110_01001001_01001001_01001001_00110110;
393     4'h9: nib2char = 40'b00000110_01001001_01001001_00101001_00011110;
394     4'hA: nib2char = 40'b01111110_00001001_00001001_00001001_01111110;
395     4'hB: nib2char = 40'b01111111_01001001_01001001_01001001_00110110;

```



```
396     4'hC: nib2char = 40'b00111110_01000001_01000001_01000001_00100010;
397     4'hD: nib2char = 40'b01111111_01000001_01000001_01000001_00111110;
398     4'hE: nib2char = 40'b01111111_01001001_01001001_01001001_01000001;
399     4'hF: nib2char = 40'b01111111_00001001_00001001_00001001_00000001;
400   endcase
401   end
402 endfunction
403
404 wire [159:0] data_dots;
405 assign data_dots = {nib2char(frdata[15:12]), nib2char(frdata[11:8]),
406                   nib2char(frdata[7:4]), nib2char(frdata[3:0])};
407
408 wire [239:0] address_dots;
409 assign address_dots = {nib2char({ 1'b0, faddress[22:20]}),
410                      nib2char(faddress[19:16]),
411                      nib2char(faddress[15:12]),
412                      nib2char(faddress[11:8]),
413                      nib2char(faddress[7:4]),
414                      nib2char(faddress[3:0])};
415
416 always @(status or address_dots or data_dots)
417   case (status)
418     `STATUS_RESET:
419       dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
420              40'b01111111_01001001_01001001_01001001_01000001, // E
421              40'b00100110_01001001_01001001_01001001_00110010, // S
422              40'b01111111_01001001_01001001_01001001_01000001, // E
423              40'b00000001_00000001_01111111_00000001_00000001, // T
424              40'b00000000_00000000_00000000_00000000_00000000, //
425              40'b00000000_00000000_00000000_00000000_00000000, //
426              40'b00000000_00000000_00000000_00000000_00000000, //
427              40'b00000000_00000000_00000000_00000000_00000000, //
428              40'b00000000_00000000_00000000_00000000_00000000, //
429              40'b00001000_00001000_00001000_00001000_00001000, // -
430              40'b00001000_00001000_00001000_00001000_00001000, // -
431              40'b00001000_00001000_00001000_00001000_00001000, // -
432              40'b00001000_00001000_00001000_00001000_00001000, // -
433              40'b00001000_00001000_00001000_00001000_00001000, // -
434              40'b00001000_00001000_00001000_00001000_00001000}; // -
435     `STATUS_READ_ID:
436       dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
437              40'b01111111_01001001_01001001_01001001_01000001, // E
438              40'b01111110_00001001_00001001_00001001_01111110, // A
439              40'b01111111_01000001_01000001_01000001_00111110, // D
440              40'b00000000_00000000_00000000_00000000_00000000, //
441              40'b00000000_01000001_01111111_01000001_00000000, // I
442              40'b01111111_01000001_01000001_01000001_00111110, // D
443              40'b00000000_00000000_00000000_00000000_00000000, //
444              40'b00000000_00000000_00000000_00000000_00000000, //
445              40'b00000000_00000000_00000000_00000000_00000000, //
446              address_dots};
447     `STATUS_CLEAR_LOCKS:
448       dots <= {40'b00111110_01000001_01000001_01000001_00100010, // C
449              40'b01111111_01000000_01000000_01000000_01000000, // L
450              40'b01111111_00001001_00011001_00101001_01000110, // R
451              40'b00000000_00000000_00000000_00000000_00000000, //
452              40'b01111111_01000000_01000000_01000000_01000000, // L
453              40'b00111110_01000001_01000001_01000001_00111110, // O
454              40'b00111110_01000001_01000001_01000001_00100010, // C
455              40'b01111111_00001000_00010100_00100010_01000001, // K
456              40'b00100110_01001001_01001001_01001001_00110010, // S
457              40'b00000000_00000000_00000000_00000000_00000000, //
458              address_dots};
459     `STATUS_ERASING:
460       dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
461              40'b01111111_00001001_00011001_00101001_01000110, // R
462              40'b01111110_00001001_00001001_00001001_01111110, // A
463              40'b00100110_01001001_01001001_01001001_00110010, // S
464              40'b00000000_01000001_01111111_01000001_00000000, // I
465              40'b01111111_00000010_00000100_00001000_01111111, // N
466              40'b00111110_01000001_01001001_01001001_00111010, // G
467              40'b00000000_00000000_00000000_00000000_00000000, //
468              40'b00000000_00000000_00000000_00000000_00000000, //
469              40'b00000000_00000000_00000000_00000000_00000000, //
470              address_dots};
471     `STATUS_WRITING:
472       dots <= {40'b01111111_00100000_00011000_00100000_01111111, // W
473              40'b01111111_00001001_00011001_00101001_01000110, // R
474              40'b00000000_01000001_01111111_01000001_00000000, // I
```

```
475     40'b00000001_00000001_01111111_00000001_00000001, // T
476     40'b00000000_01000001_01111111_01000001_00000000, // I
477     40'b01111111_00000010_00000100_00001000_01111111, // N
478     40'b00111110_01000001_01001001_01001001_00111010, // G
479     40'b00000000_00000000_00000000_00000000_00000000, //
480     40'b00000000_00000000_00000000_00000000_00000000, //
481     40'b00000000_00000000_00000000_00000000_00000000, //
482     address_dots};
483     `STATUS_READING:
484     dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
485             40'b01111111_01001001_01001001_01001001_01000001, // E
486             40'b01111110_00001001_00001001_00001001_01111110, // A
487             40'b01111111_01000001_01000001_01000001_00111110, // D
488             40'b00000000_01000001_01111111_01000001_00000000, // I
489             40'b01111111_00000010_00000100_00001000_01111111, // N
490             40'b00111110_01000001_01001001_01001001_00111010, // G
491             40'b00000000_00000000_00000000_00000000_00000000, //
492             40'b00000000_00000000_00000000_00000000_00000000, //
493             40'b00000000_00000000_00000000_00000000_00000000, //
494             address_dots};
495     `STATUS_SUCCESS:
496     dots <= {40'b00000000_00000000_00000000_00000000_00000000, //
497             40'b00101010_00011100_01111111_00011100_00101010, // *
498             40'b00101010_00011100_01111111_00011100_00101010, // *
499             40'b00101010_00011100_01111111_00011100_00101010, // *
500             40'b00000000_00000000_00000000_00000000_00000000, //
501             40'b01111111_00001001_00001001_00001001_00000110, // P
502             40'b01111110_00001001_00001001_00001001_01111110, // A
503             40'b00100110_01001001_01001001_01001001_00110010, // S
504             40'b00100110_01001001_01001001_01001001_00110010, // S
505             40'b01111111_01001001_01001001_01001001_01000001, // E
506             40'b01111111_01000001_01000001_01000001_00111110, // D
507             40'b00000000_00000000_00000000_00000000_00000000, //
508             40'b00101010_00011100_01111111_00011100_00101010, // *
509             40'b00101010_00011100_01111111_00011100_00101010, // *
510             40'b00101010_00011100_01111111_00011100_00101010, // *
511             40'b00000000_00000000_00000000_00000000_00000000}; //
512     `STATUS_BAD_MANUFACTURER:
513     dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
514             40'b01111111_00001001_00011001_00101001_01000110, // R
515             40'b01111111_00001001_00011001_00101001_01000110, // R
516             40'b00000000_00110110_00110110_00000000_00000000, // :
517             40'b00000000_00000000_00000000_00000000_00000000, //
518             40'b01111111_00000010_00001100_00000010_01111111, // M
519             40'b01111110_00001001_00001001_00001001_01111110, // A
520             40'b01111111_00000010_00000100_00001000_01111111, // N
521             40'b01111111_00001001_00001001_00001001_00000001, // U
522             40'b01111111_00001001_00001001_00001001_00000001, // F
523             40'b00000000_00000000_00000000_00000000_00000000, //
524             40'b00000000_00000000_00000000_00000000_00000000, //
525             data_dots};
526     `STATUS_BAD_SIZE:
527     dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
528             40'b01111111_00001001_00011001_00101001_01000110, // R
529             40'b01111111_00001001_00011001_00101001_01000110, // R
530             40'b00000000_00110110_00110110_00000000_00000000, // :
531             40'b00000000_00000000_00000000_00000000_00000000, //
532             40'b00100110_01001001_01001001_01001001_00110010, // S
533             40'b00000000_01000001_01111111_01000001_00000000, // I
534             40'b01100001_01010001_01001001_01000101_01000011, // Z
535             40'b01111111_01001001_01001001_01001001_01000001, // E
536             40'b00000000_00000000_00000000_00000000_00000000, //
537             40'b00000000_00000000_00000000_00000000_00000000, //
538             40'b00000000_00000000_00000000_00000000_00000000, //
539             data_dots};
540     `STATUS_LOCK_BIT_ERROR:
541     dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
542             40'b01111111_00001001_00011001_00101001_01000110, // R
543             40'b01111111_00001001_00011001_00101001_01000110, // R
544             40'b00000000_00110110_00110110_00000000_00000000, // :
545             40'b00000000_00000000_00000000_00000000_00000000, //
546             40'b01111111_01000000_01000000_01000000_01000000, // L
547             40'b00111110_01000001_01000001_01000001_00111110, // O
548             40'b00111110_01000001_01000001_01000001_00100010, // C
549             40'b01111111_00001000_00010100_00100010_01000001, // K
550             40'b00100110_01001001_01001001_01001001_00110010, // S
551             40'b00000000_00000000_00000000_00000000_00000000, //
552             40'b00000000_00000000_00000000_00000000_00000000, //
553             data_dots};
```

```

554 `STATUS_ERASE_BLOCK_ERROR:
555 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
556 40'b01111111_00001001_00011001_00101001_01000110, // R
557 40'b01111111_00001001_00011001_00101001_01000110, // R
558 40'b00000000_00110110_00110110_00000000_00000000, // :
559 40'b00000000_00000000_00000000_00000000_00000000, //
560 40'b01111111_01001001_01001001_01001001_01000001, // E
561 40'b01111111_00001001_00011001_00101001_01000110, // R
562 40'b01111110_00001001_00001001_00001001_01111110, // A
563 40'b00100110_01001001_01001001_01001001_00110010, // S
564 40'b01111111_01001001_01001001_01001001_01000001, // E
565 40'b00000000_00000000_00000000_00000000_00000000,
566 40'b00000000_00000000_00000000_00000000_00000000,
567 data_dots};
568 `STATUS_WRITE_ERROR:
569 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
570 40'b01111111_00001001_00011001_00101001_01000110, // R
571 40'b01111111_00001001_00011001_00101001_01000110, // R
572 40'b00000000_00110110_00110110_00000000_00000000, // :
573 40'b00000000_00000000_00000000_00000000_00000000, //
574 40'b01111111_00100000_00011000_00100000_01111111, // W
575 40'b01111111_00001001_00011001_00101001_01000110, // R
576 40'b00000000_01000001_01111111_01000001_00000000, // I
577 40'b00000001_00000001_01111111_00000001_00000001, // T
578 40'b01111111_01001001_01001001_01001001_01000001, // E
579 40'b00000000_00000000_00000000_00000000_00000000,
580 40'b00000000_00000000_00000000_00000000_00000000,
581 data_dots};
582 `STATUS_READ_WRONG_DATA:
583 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
584 40'b01111111_00001001_00011001_00101001_01000110, // R
585 40'b01111111_00001001_00011001_00101001_01000110, // R
586 40'b00000000_00110110_00110110_00000000_00000000, // :
587 40'b00000000_00000000_00000000_00000000_00000000,
588 address_dots,
589 40'b00000000_00000000_00000000_00000000_00000000,
590 data_dots};
591 default:
592 dots <= {16{40'b01010101_00101010_01010101_00101010_01010101}};
593 endcase
594
595 endmodule
596
597
598
599
600
601
602

```

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // 6.111 FPGA Labkit -- Flash Tester State Machine
4 //
5 //
6 // Created: January 23, 2005
7 // Author: Nathan Ickes
8 //
9 ////////////////////////////////////////////////////////////////////
10
11 `define FLASHOP_IDLE 2'b00
12 `define FLASHOP_READ 2'b01
13 `define FLASHOP_WRITE 2'b10
14
15 `define STATUS_RESET 4'h0
16 `define STATUS_READ_ID 4'h1
17 `define STATUS_CLEAR_LOCKS 4'h2
18 `define STATUS_ERASING 4'h3
19 `define STATUS_WRITING 4'h4
20 `define STATUS_READING 4'h5
21 `define STATUS_SUCCESS 4'h6
22 `define STATUS_BAD_MANUFACTURER 4'h7
23 `define STATUS_BAD_SIZE 4'h8
24 `define STATUS_LOCK_BIT_ERROR 4'h9
25 `define STATUS_ERASE_BLOCK_ERROR 4'ha
26 `define STATUS_WRITE_ERROR 4'hB
27 `define STATUS_READ_WRONG_DATA 4'hC
28
29 //`define NUM_BLOCKS 128
30 `define NUM_BLOCKS 128
31 `define BLOCK_SIZE 64*1024
32 `define LAST_BLOCK_ADDRESS ((`NUM_BLOCKS-1)*`BLOCK_SIZE)
33 `define LAST_ADDRESS (`NUM_BLOCKS*`BLOCK_SIZE-1)
34
35 `define NUM_BLOCKS2 2
36 `define BLOCK_SIZE2 64*1024
37 `define LAST_BLOCK_ADDRESS2 ((`NUM_BLOCKS2-1)*`BLOCK_SIZE2)
38 `define LAST_ADDRESS2 (`NUM_BLOCKS2*`BLOCK_SIZE2-1)
39
40
41 `define INIT_FLASH_ADDRESS 23'h0 // next one 23'd8192
42 `define LAST_FLASH_ADDRESS (`INIT_FLASH_ADDRESS+23'd8192)
43
44
45 module test_fsm (reset, clock, fop, faddress, fwdata, frdata, fbusy, dots, FPGA_ROM_data, FPGA_ROM_addr);
46
47     input reset, clock;
48     output [1:0] fop;
49     output [22:0] faddress;
50     output [15:0] fwdata;
51     input [15:0] frdata;
52     input fbusy;
53     output [639:0] dots;
54     input [7:0] FPGA_ROM_data;
55     output [12:0] FPGA_ROM_addr;
56
57     reg [1:0] fop;
58     reg [22:0] faddress;
59     reg [15:0] fwdata;
60     reg [639:0] dots;
61     reg [12:0] FPGA_ROM_addr;
62
63     ////////////////////////////////////////////////////////////////////
64     //
65     // State Machine
66     //
67     ////////////////////////////////////////////////////////////////////
68
69     reg [7:0] state;
70     reg [3:0] status;
71
72     always @(posedge clock)
73     if (reset)
74     begin
75         state <= 0;
76         status <= `STATUS_RESET;
77         faddress <= 0;
78         FPGA_ROM_addr <= 0;
79         fop <= `FLASHOP_IDLE;
```

```
80     end
81     else if (!fbusy && (fop == `FLASHOP_IDLE))
82         case (state)
83     8'h00:
84         begin
85             // Issue "read id codes" command
86             status <= `STATUS_READ_ID;
87             faddress <= 0;
88             fwdata <= 16'h0090;
89             fop <= `FLASHOP_WRITE;
90             state <= state+1;
91         end
92
93     8'h01:
94         begin
95             // Read manufacturer code
96             faddress <= 0;
97             fop <= `FLASHOP_READ;
98             state <= state+1;
99         end
100
101     8'h02:
102         if (frdata != 16'h0089) // 16'h0089 = Intel
103             status <= `STATUS_BAD_MANUFACTURER;
104         else
105             begin
106                 // Read the device size code
107                 faddress <= 1;
108                 fop <= `FLASHOP_READ;
109                 state <= state+1;
110             end
111
112     8'h03:
113         if (frdata != 16'h0018) // 16'h0018 = 128Mbit
114             status <= `STATUS_BAD_SIZE;
115         else
116             begin
117                 faddress <= 0;
118                 fwdata <= 16'hFF;
119                 fop <= `FLASHOP_WRITE;
120                 state <= state+1;
121             end
122     8'h04:
123         begin
124             // Issue "clear lock bits" command
125             status <= `STATUS_CLEAR_LOCKS;
126             faddress <= 0;
127             fwdata <= 16'h60;
128             fop <= `FLASHOP_WRITE;
129             state <= state+1;
130         end
131     8'h05:
132         begin
133             // Issue "confirm clear lock bits" command
134             faddress <= 0;
135             fwdata <= 16'hD0;
136             fop <= `FLASHOP_WRITE;
137             state <= state+1;
138         end
139     8'h06:
140         begin
141             // Read status
142             faddress <= 0;
143             fop <= `FLASHOP_READ;
144             state <= state+1;
145         end
146     8'h07:
147         if (frdata[7] == 1) // Done clearing lock bits
148             if (frdata[6:1] == 0) // No errors
149                 begin
150                     faddress <= 0;
151                     fop <= `FLASHOP_IDLE;
152                     state <= state+1;
153                 end
154             else
155                 status <= `STATUS_LOCK_BIT_ERROR;
156         else // Still busy, reread status register
157             begin
158                 faddress <= 0;
```

```
159     fop <= `FLASHOP_READ;
160     end
161
162     //////////////////////////////////////
163     // Block Erase Sequence
164     //////////////////////////////////////
165
166     8'h08:
167     begin
168         status <= `STATUS_ERASING;
169         fwdata <= 16'h20; // Issue "erase block" command
170         fop <= `FLASHOP_WRITE;
171         state <= state+1;
172     end
173     8'h09:
174     begin
175         fwdata <= 16'hD0; // Issue "confirm erase" command
176         fop <= `FLASHOP_WRITE;
177         state <= state+1;
178     end
179     8'h0A:
180     begin
181         fop <= `FLASHOP_READ;
182         state <= state+1;
183     end
184     8'h0B:
185     if (frdata[7] == 1) // Done erasing block
186         if (frdata[6:1] == 0) // No errors
187             if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS)
188                 //if (faddress != `LAST_BLOCK_ADDRESS) // `LAST_BLOCK_ADDRESS)
189                 //if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS)
190                 //if (faddress != 23'h100) // `LAST_BLOCK_ADDRESS)
191             begin
192                 faddress <= faddress+`BLOCK_SIZE;
193                 fop <= `FLASHOP_IDLE;
194                 state <= state-3;
195             end
196             else
197             begin
198                 faddress <= `INIT_FLASH_ADDRESS;
199                 FPGA_ROM_addr <= 0;
200                 fop <= `FLASHOP_IDLE;
201                 state <= state+1;
202             end
203             else // Erase error detected
204                 status <= `STATUS_ERASE_BLOCK_ERROR;
205         else // Still busy
206             fop <= `FLASHOP_READ;
207
208     //////////////////////////////////////
209     // Write Addresses to All Locations
210     //////////////////////////////////////
211
212     8'h0C:
213     begin
214         status <= `STATUS_WRITING;
215         fwdata <= 16'h40; // Issue "setup write" command
216         fop <= `FLASHOP_WRITE;
217         state <= state+1;
218     end
219
220     8'h0D:
221     begin
222         //fwdata <= faddress[15:0]^faddress[22:16]; // Finish write
223         //fwdata <= 16'b10101; // Finish write
224         fwdata <= {8'b0, FPGA_ROM_data}; // Finish write
225         fop <= `FLASHOP_WRITE;
226         state <= state+1;
227     end
228     8'h0E:
229     begin
230         // Read status register
231         fop <= `FLASHOP_READ;
232         state <= state+1;
233     end
234     8'h0F:
235     if (frdata[7] == 1) // Done writing
236         if (frdata[6:1] == 0) // No errors
237             if (faddress != `LAST_FLASH_ADDRESS) // `LAST_ADDRESS)
```

```
238 //if (faddress != {20'b0, FPGA_ROM_addr}) // `LAST_ADDRESS)
239 //if (faddress != 23'h7FFFF) // `LAST_ADDRESS) , if (faddress != 23'h20000)
240 begin
241     faddress <= faddress+1;
242     FPGA_ROM_addr <= FPGA_ROM_addr+1;
243     fop <= `FLASHOP_IDLE;
244     state <= state-3;
245 end
246 else
247 begin
248     faddress <= `INIT_FLASH_ADDRESS;
249     FPGA_ROM_addr <= 0;
250     fop <= `FLASHOP_IDLE;
251     state <= state+1;
252 end
253 else // Write error detected
254     status <= `STATUS_WRITE_ERROR;
255 else // Still busy
256     fop <= `FLASHOP_READ;
257
258 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
259 // Read back data
260 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
261
262 8'h10:
263 begin
264     status <= `STATUS_READING;
265     fwdata <= 16'hFF; // Issue "read array" command
266     fop <= `FLASHOP_WRITE;
267     faddress <= `INIT_FLASH_ADDRESS;
268     FPGA_ROM_addr <= 0;
269     state <= state+1;
270 end
271 8'h11:
272 begin
273     fop <= `FLASHOP_READ;
274     state <= state+1;
275 end
276 8'h12:
277 if (frdata == {8'b0, FPGA_ROM_data})
278     if (faddress == `LAST_FLASH_ADDRESS)
279 //if (faddress == `LAST_ADDRESS2)
280 //if (faddress == 23'h7FFFF)
281     begin
282         fop <= `FLASHOP_IDLE;
283         faddress <= `INIT_FLASH_ADDRESS;
284         FPGA_ROM_addr <= 0;
285         state <= state+5;
286     end
287     else
288     begin
289         faddress <= faddress+1;
290         FPGA_ROM_addr <= FPGA_ROM_addr+1;
291         fop <= `FLASHOP_READ;
292     end
293 else
294     status <= `STATUS_READ_WRONG_DATA;
295 /*
296 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
297 // Erase the chip again
298 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
299
300 8'h13:
301 begin
302     status <= `STATUS_ERASING;
303     fwdata <= 16'h20; // Issue "erase block" command
304     fop <= `FLASHOP_WRITE;
305     state <= state+1;
306 end
307 8'h14:
308 begin
309     fwdata <= 16'hD0; // Issue "confirm erase" command
310     fop <= `FLASHOP_WRITE;
311     state <= state+1;
312 end
313 8'h15:
314 begin
315     fop <= `FLASHOP_READ;
316     state <= state+1;
```

```

317     end
318     8'h16:
319     if (frdata[7] == 1) // Done erasing block
320     if (frdata[6:1] == 0) // No errors
321     if (faddress != 23'h20000) // `LAST_BLOCK_ADDRESS)
322     //if (faddress != `LAST_BLOCK_ADDRESS) // `LAST_BLOCK_ADDRESS)
323     // if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS)
324     // if (faddress != 23'h100) // `LAST_BLOCK_ADDRESS)
325     begin
326     faddress <= faddress+`BLOCK_SIZE;
327     fop <= `FLASHOP_IDLE;
328     state <= state-3;
329     end
330     else
331     begin
332     faddress <= 0;
333     fop <= `FLASHOP_IDLE;
334     state <= state+1;
335     end
336     else // Erase error detected
337     status <= `STATUS_ERASE_BLOCK_ERROR;
338     else // Still busy
339     fop <= `FLASHOP_READ;
340 */
341 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
342 // End of test: declare success
343 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
344
345 8'h17:
346     begin
347     status <= `STATUS_SUCCESS;
348     fop <= `FLASHOP_IDLE;
349     end
350
351     endcase
352     else
353     fop <= `FLASHOP_IDLE;
354
355 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
356 //
357 // Status display
358 //
359 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
360 //
361 // "Reset    -----" --> During reset
362 // "Read ID  000000" --> While reading ID codes
363 // "Clr locks 000000" --> While clearing block locks
364 // "Erase    000000" --> While erasing
365 // "Write    000000" --> While writing
366 // "Read     000000" --> While reading
367 // " *** PASSED *** " --> If the entire test completes with no errors
368 // "Err: Manuf 0000" --> If an incorrect manufacturer code is read
369 // "Err: Size  0000" --> If an incorrect size code is read
370 // "Err: Locks 0000" --> If an error is detected when clearing the block lock bits
371 // "Err: Erase 0000"
372 // "Err: 000000 0000"
373
374
375 // Rd  000000  0000
376 // Wr  000000  0000
377 // Id  000000  0000
378 //
379
380 function [39:0] nib2char;
381     input [3:0] nib;
382     begin
383     case (nib)
384     4'h0: nib2char = 40'b00111110_01010001_01001001_01000101_00111110;
385     4'h1: nib2char = 40'b00000000_01000010_01111111_01000000_00000000;
386     4'h2: nib2char = 40'b01100010_01010001_01001001_01001001_01000110;
387     4'h3: nib2char = 40'b00100010_01000001_01001001_01001001_00110110;
388     4'h4: nib2char = 40'b00011000_00010100_00010010_01111111_00010000;
389     4'h5: nib2char = 40'b00100111_01000101_01000101_01000101_00111001;
390     4'h6: nib2char = 40'b00111100_01001010_01001001_01001001_00110000;
391     4'h7: nib2char = 40'b00000001_01110001_00001001_00000101_00000011;
392     4'h8: nib2char = 40'b00110110_01001001_01001001_01001001_00110110;
393     4'h9: nib2char = 40'b00000110_01001001_01001001_00101001_00011110;
394     4'hA: nib2char = 40'b01111110_00001001_00001001_00001001_01111110;
395     4'hB: nib2char = 40'b01111111_01001001_01001001_01001001_00110110;

```



```
396     4'hC: nib2char = 40'b00111110_01000001_01000001_01000001_00100010;
397     4'hD: nib2char = 40'b01111111_01000001_01000001_01000001_00111110;
398     4'hE: nib2char = 40'b01111111_01001001_01001001_01001001_01000001;
399     4'hF: nib2char = 40'b01111111_00001001_00001001_00001001_00000001;
400   endcase
401   end
402 endfunction
403
404 wire [159:0] data_dots;
405 assign data_dots = {nib2char(frdata[15:12]), nib2char(frdata[11:8]),
406                   nib2char(frdata[7:4]), nib2char(frdata[3:0])};
407
408 wire [239:0] address_dots;
409 assign address_dots = {nib2char({ 1'b0, faddress[22:20]}),
410                      nib2char(faddress[19:16]),
411                      nib2char(faddress[15:12]),
412                      nib2char(faddress[11:8]),
413                      nib2char(faddress[7:4]),
414                      nib2char(faddress[3:0])};
415
416 always @(status or address_dots or data_dots)
417   case (status)
418     `STATUS_RESET:
419       dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
420              40'b01111111_01001001_01001001_01001001_01000001, // E
421              40'b00100110_01001001_01001001_01001001_00110010, // S
422              40'b01111111_01001001_01001001_01001001_01000001, // E
423              40'b00000001_00000001_01111111_00000001_00000001, // T
424              40'b00000000_00000000_00000000_00000000_00000000, //
425              40'b00000000_00000000_00000000_00000000_00000000, //
426              40'b00000000_00000000_00000000_00000000_00000000, //
427              40'b00000000_00000000_00000000_00000000_00000000, //
428              40'b00000000_00000000_00000000_00000000_00000000, //
429              40'b00001000_00001000_00001000_00001000_00001000, // -
430              40'b00001000_00001000_00001000_00001000_00001000, // -
431              40'b00001000_00001000_00001000_00001000_00001000, // -
432              40'b00001000_00001000_00001000_00001000_00001000, // -
433              40'b00001000_00001000_00001000_00001000_00001000, // -
434              40'b00001000_00001000_00001000_00001000_00001000}; // -
435     `STATUS_READ_ID:
436       dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
437              40'b01111111_01001001_01001001_01001001_01000001, // E
438              40'b01111110_00001001_00001001_00001001_01111110, // A
439              40'b01111111_01000001_01000001_01000001_00111110, // D
440              40'b00000000_00000000_00000000_00000000_00000000, //
441              40'b00000000_01000001_01111111_01000001_00000000, // I
442              40'b01111111_01000001_01000001_01000001_00111110, // D
443              40'b00000000_00000000_00000000_00000000_00000000, //
444              40'b00000000_00000000_00000000_00000000_00000000, //
445              40'b00000000_00000000_00000000_00000000_00000000, //
446              address_dots};
447     `STATUS_CLEAR_LOCKS:
448       dots <= {40'b00111110_01000001_01000001_01000001_00100010, // C
449              40'b01111111_01000000_01000000_01000000_01000000, // L
450              40'b01111111_00001001_00011001_00101001_01000110, // R
451              40'b00000000_00000000_00000000_00000000_00000000, //
452              40'b01111111_01000000_01000000_01000000_01000000, // L
453              40'b00111110_01000001_01000001_01000001_00111110, // O
454              40'b00111110_01000001_01000001_01000001_00100010, // C
455              40'b01111111_00001000_00010100_00100010_01000001, // K
456              40'b00100110_01001001_01001001_01001001_00110010, // S
457              40'b00000000_00000000_00000000_00000000_00000000, //
458              address_dots};
459     `STATUS_ERASING:
460       dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
461              40'b01111111_00001001_00011001_00101001_01000110, // R
462              40'b01111110_00001001_00001001_00001001_01111110, // A
463              40'b00100110_01001001_01001001_01001001_00110010, // S
464              40'b00000000_01000001_01111111_01000001_00000000, // I
465              40'b01111111_00000010_00000100_00001000_01111111, // N
466              40'b00111110_01000001_01001001_01001001_00111010, // G
467              40'b00000000_00000000_00000000_00000000_00000000, //
468              40'b00000000_00000000_00000000_00000000_00000000, //
469              40'b00000000_00000000_00000000_00000000_00000000, //
470              address_dots};
471     `STATUS_WRITING:
472       dots <= {40'b01111111_00100000_00011000_00100000_01111111, // W
473              40'b01111111_00001001_00011001_00101001_01000110, // R
474              40'b00000000_01000001_01111111_01000001_00000000, // I
```

```
475     40'b00000001_00000001_01111111_00000001_00000001, // T
476     40'b00000000_01000001_01111111_01000001_00000000, // I
477     40'b01111111_00000010_00000100_00001000_01111111, // N
478     40'b00111110_01000001_01001001_01001001_00111010, // G
479     40'b00000000_00000000_00000000_00000000_00000000, //
480     40'b00000000_00000000_00000000_00000000_00000000, //
481     40'b00000000_00000000_00000000_00000000_00000000, //
482     address_dots};
483     `STATUS_READING:
484     dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
485             40'b01111111_01001001_01001001_01001001_01000001, // E
486             40'b01111110_00001001_00001001_00001001_01111110, // A
487             40'b01111111_01000001_01000001_01000001_00111110, // D
488             40'b00000000_01000001_01111111_01000001_00000000, // I
489             40'b01111111_00000010_00000100_00001000_01111111, // N
490             40'b00111110_01000001_01001001_01001001_00111010, // G
491             40'b00000000_00000000_00000000_00000000_00000000, //
492             40'b00000000_00000000_00000000_00000000_00000000, //
493             40'b00000000_00000000_00000000_00000000_00000000, //
494             address_dots};
495     `STATUS_SUCCESS:
496     dots <= {40'b00000000_00000000_00000000_00000000_00000000, //
497             40'b00101010_00011100_01111111_00011100_00101010, // *
498             40'b00101010_00011100_01111111_00011100_00101010, // *
499             40'b00101010_00011100_01111111_00011100_00101010, // *
500             40'b00000000_00000000_00000000_00000000_00000000, //
501             40'b01111111_00001001_00001001_00001001_00000110, // P
502             40'b01111110_00001001_00001001_00001001_01111110, // A
503             40'b00100110_01001001_01001001_01001001_00110010, // S
504             40'b00100110_01001001_01001001_01001001_00110010, // S
505             40'b01111111_01001001_01001001_01001001_01000001, // E
506             40'b01111111_01000001_01000001_01000001_00111110, // D
507             40'b00000000_00000000_00000000_00000000_00000000, //
508             40'b00101010_00011100_01111111_00011100_00101010, // *
509             40'b00101010_00011100_01111111_00011100_00101010, // *
510             40'b00101010_00011100_01111111_00011100_00101010, // *
511             40'b00000000_00000000_00000000_00000000_00000000}; //
512     `STATUS_BAD_MANUFACTURER:
513     dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
514             40'b01111111_00001001_00011001_00101001_01000110, // R
515             40'b01111111_00001001_00011001_00101001_01000110, // R
516             40'b00000000_00110110_00110110_00000000_00000000, // :
517             40'b00000000_00000000_00000000_00000000_00000000, //
518             40'b01111111_00000010_00001100_00000010_01111111, // M
519             40'b01111110_00001001_00001001_00001001_01111110, // A
520             40'b01111111_00000010_00000100_00001000_01111111, // N
521             40'b01111111_00001001_00001001_00001001_00000001, // U
522             40'b01111111_00001001_00001001_00001001_00000001, // F
523             40'b00000000_00000000_00000000_00000000_00000000, //
524             40'b00000000_00000000_00000000_00000000_00000000, //
525             data_dots};
526     `STATUS_BAD_SIZE:
527     dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
528             40'b01111111_00001001_00011001_00101001_01000110, // R
529             40'b01111111_00001001_00011001_00101001_01000110, // R
530             40'b00000000_00110110_00110110_00000000_00000000, // :
531             40'b00000000_00000000_00000000_00000000_00000000, //
532             40'b00100110_01001001_01001001_01001001_00110010, // S
533             40'b00000000_01000001_01111111_01000001_00000000, // I
534             40'b01100001_01010001_01001001_01000101_01000011, // Z
535             40'b01111111_01001001_01001001_01001001_01000001, // E
536             40'b00000000_00000000_00000000_00000000_00000000, //
537             40'b00000000_00000000_00000000_00000000_00000000, //
538             40'b00000000_00000000_00000000_00000000_00000000, //
539             data_dots};
540     `STATUS_LOCK_BIT_ERROR:
541     dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
542             40'b01111111_00001001_00011001_00101001_01000110, // R
543             40'b01111111_00001001_00011001_00101001_01000110, // R
544             40'b00000000_00110110_00110110_00000000_00000000, // :
545             40'b00000000_00000000_00000000_00000000_00000000, //
546             40'b01111111_01000000_01000000_01000000_01000000, // L
547             40'b00111110_01000001_01000001_01000001_00111110, // O
548             40'b00111110_01000001_01000001_01000001_00100010, // C
549             40'b01111111_00001000_00010100_00100010_01000001, // K
550             40'b00100110_01001001_01001001_01001001_00110010, // S
551             40'b00000000_00000000_00000000_00000000_00000000, //
552             40'b00000000_00000000_00000000_00000000_00000000, //
553             data_dots};
```

```
554 `STATUS_ERASE_BLOCK_ERROR:
555 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
556 40'b01111111_00001001_00011001_00101001_01000110, // R
557 40'b01111111_00001001_00011001_00101001_01000110, // R
558 40'b00000000_00110110_00110110_00000000_00000000, // :
559 40'b00000000_00000000_00000000_00000000_00000000, //
560 40'b01111111_01001001_01001001_01001001_01000001, // E
561 40'b01111111_00001001_00011001_00101001_01000110, // R
562 40'b01111110_00001001_00001001_00001001_01111110, // A
563 40'b00100110_01001001_01001001_01001001_00110010, // S
564 40'b01111111_01001001_01001001_01001001_01000001, // E
565 40'b00000000_00000000_00000000_00000000_00000000,
566 40'b00000000_00000000_00000000_00000000_00000000,
567 data_dots};
568 `STATUS_WRITE_ERROR:
569 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
570 40'b01111111_00001001_00011001_00101001_01000110, // R
571 40'b01111111_00001001_00011001_00101001_01000110, // R
572 40'b00000000_00110110_00110110_00000000_00000000, // :
573 40'b00000000_00000000_00000000_00000000_00000000, //
574 40'b01111111_00100000_00011000_00100000_01111111, // W
575 40'b01111111_00001001_00011001_00101001_01000110, // R
576 40'b00000000_01000001_01111111_01000001_00000000, // I
577 40'b00000001_00000001_01111111_00000001_00000001, // T
578 40'b01111111_01001001_01001001_01001001_01000001, // E
579 40'b00000000_00000000_00000000_00000000_00000000,
580 40'b00000000_00000000_00000000_00000000_00000000,
581 data_dots};
582 `STATUS_READ_WRONG_DATA:
583 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
584 40'b01111111_00001001_00011001_00101001_01000110, // R
585 40'b01111111_00001001_00011001_00101001_01000110, // R
586 40'b00000000_00110110_00110110_00000000_00000000, // :
587 40'b00000000_00000000_00000000_00000000_00000000,
588 address_dots,
589 40'b00000000_00000000_00000000_00000000_00000000,
590 data_dots};
591 default:
592 dots <= {16{40'b01010101_00101010_01010101_00101010_01010101}};
593 endcase
594
595 endmodule
596
597
598
599
600
601
602
```

```

1 ////////////////////////////////////////////////////////////////////
2 //
3 // 6.111 FPGA Labkit -- Flash Tester State Machine
4 //
5 //
6 // Created: January 23, 2005
7 // Author: Nathan Ickes
8 //
9 ////////////////////////////////////////////////////////////////////
10
11 `define FLASHOP_IDLE 2'b00
12 `define FLASHOP_READ 2'b01
13 `define FLASHOP_WRITE 2'b10
14
15 `define STATUS_RESET 4'h0
16 `define STATUS_READ_ID 4'h1
17 `define STATUS_CLEAR_LOCKS 4'h2
18 `define STATUS_ERASING 4'h3
19 `define STATUS_WRITING 4'h4
20 `define STATUS_READING 4'h5
21 `define STATUS_SUCCESS 4'h6
22 `define STATUS_BAD_MANUFACTURER 4'h7
23 `define STATUS_BAD_SIZE 4'h8
24 `define STATUS_LOCK_BIT_ERROR 4'h9
25 `define STATUS_ERASE_BLOCK_ERROR 4'hA
26 `define STATUS_WRITE_ERROR 4'hB
27 `define STATUS_READ_WRONG_DATA 4'hC
28 `define STATUS_READOUT_DATA 4'hD
29
30 //`define NUM_BLOCKS 128
31 `define NUM_BLOCKS 128
32 `define BLOCK_SIZE 64*1024
33 `define LAST_BLOCK_ADDRESS ((`NUM_BLOCKS-1)*`BLOCK_SIZE)
34 `define LAST_ADDRESS (`NUM_BLOCKS*`BLOCK_SIZE-1)
35
36 `define NUM_BLOCKS2 2
37 `define BLOCK_SIZE2 64*1024
38 `define LAST_BLOCK_ADDRESS2 ((`NUM_BLOCKS2-1)*`BLOCK_SIZE2)
39 `define LAST_ADDRESS2 (`NUM_BLOCKS2*`BLOCK_SIZE2-1)
40
41
42 `define INIT_FLASH_ADDRESS 23'd32772 // 23'h0, 23'd8192
43 `define LAST_FLASH_ADDRESS (`INIT_FLASH_ADDRESS+23'd20)
44
45
46 module test_fsm (reset, clock, fop, faddress, fwdata, frdata, fbusy, dots, FPGA_ROM_data, FPGA_ROM_addr);
47
48     input reset, clock;
49     output [1:0] fop;
50     output [22:0] faddress;
51     output [15:0] fwdata;
52     input [15:0] frdata;
53     input fbusy;
54     output [639:0] dots;
55     input [7:0] FPGA_ROM_data;
56     output [12:0] FPGA_ROM_addr;
57
58     reg [1:0] fop;
59     reg [22:0] faddress;
60     reg [15:0] fwdata;
61     reg [639:0] dots;
62     reg [12:0] FPGA_ROM_addr;
63     reg [25:0] count;
64
65 ////////////////////////////////////////////////////////////////////
66 //
67 // State Machine
68 //
69 ////////////////////////////////////////////////////////////////////
70
71     reg [7:0] state;
72     reg [3:0] status;
73
74     always @(posedge clock)
75     if (reset)
76     begin
77         state <= 0;
78         status <= `STATUS_RESET;
79         faddress <= 0;

```

```
80     FPGA_ROM_addr <= 0;
81     count <= 0;
82     fop <= `FLASHOP_IDLE;
83     end
84     else if (!fbusy && (fop == `FLASHOP_IDLE))
85         case (state)
86     8'h00:
87         begin
88             // Issue "read id codes" command
89             status <= `STATUS_READ_ID;
90             faddress <= 0;
91             fwdata <= 16'h0090;
92             fop <= `FLASHOP_WRITE;
93             state <= state+1;
94         end
95
96     8'h01:
97         begin
98             // Read manufacturer code
99             faddress <= 0;
100            fop <= `FLASHOP_READ;
101            state <= state+1;
102        end
103
104     8'h02:
105         if (frdata != 16'h0089) // 16'h0089 = Intel
106             status <= `STATUS_BAD_MANUFACTURER;
107         else
108             begin
109                 // Read the device size code
110                 faddress <= 1;
111                 fop <= `FLASHOP_READ;
112                 state <= state+1;
113             end
114
115     8'h03:
116         if (frdata != 16'h0018) // 16'h0018 = 128Mbit
117             status <= `STATUS_BAD_SIZE;
118         else
119             begin
120                 faddress <= 0;
121                 fwdata <= 16'hFF;
122                 fop <= `FLASHOP_WRITE;
123                 state <= state+1;
124             end
125     8'h04:
126         begin
127             // Issue "clear lock bits" command
128             status <= `STATUS_CLEAR_LOCKS;
129             faddress <= 0;
130             fwdata <= 16'h60;
131             fop <= `FLASHOP_WRITE;
132             state <= state+1;
133         end
134     8'h05:
135         begin
136             // Issue "confirm clear lock bits" command
137             faddress <= 0;
138             fwdata <= 16'hD0;
139             fop <= `FLASHOP_WRITE;
140             state <= state+1;
141         end
142     8'h06:
143         begin
144             // Read status
145             faddress <= 0;
146             fop <= `FLASHOP_READ;
147             state <= state+1;
148         end
149     8'h07:
150         if (frdata[7] == 1) // Done clearing lock bits
151             if (frdata[6:1] == 0) // No errors
152                 begin
153                     faddress <= `INIT_FLASH_ADDRESS;    // `INIT_FLASH_ADDRESS
154                     fop <= `FLASHOP_IDLE;
155                     state <= state+ 8'd9;
156                 end
157             else
158                 status <= `STATUS_LOCK_BIT_ERROR;
```

```
159     else // Still busy, reread status register
160     begin
161     faddress <= 0;
162     fop <= `FLASHOP_READ;
163     end
164 /*
165 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
166 // Block Erase Sequence
167 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
168
169 8'h08:
170 begin
171     status <= `STATUS_ERASING;
172     fwdata <= 16'h20; // Issue "erase block" command
173     fop <= `FLASHOP_WRITE;
174     state <= state+1;
175 end
176 8'h09:
177 begin
178     fwdata <= 16'hD0; // Issue "confirm erase" command
179     fop <= `FLASHOP_WRITE;
180     state <= state+1;
181 end
182 8'h0A:
183 begin
184     fop <= `FLASHOP_READ;
185     state <= state+1;
186 end
187 8'h0B:
188 if (frdata[7] == 1) // Done erasing block
189     if (frdata[6:1] == 0) // No errors
190         if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS)
191             //if (faddress != `LAST_BLOCK_ADDRESS) // `LAST_BLOCK_ADDRESS)
192             //if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS)
193             //if (faddress != 23'h100) // `LAST_BLOCK_ADDRESS)
194         begin
195             faddress <= faddress+`BLOCK_SIZE;
196             fop <= `FLASHOP_IDLE;
197             state <= state-3;
198         end
199         else
200         begin
201             faddress <= `INIT_FLASH_ADDRESS;
202             FPGA_ROM_addr <= 0;
203             fop <= `FLASHOP_IDLE;
204             state <= state+1;
205         end
206         else // Erase error detected
207             status <= `STATUS_ERASE_BLOCK_ERROR;
208     else // Still busy
209         fop <= `FLASHOP_READ;
210
211 */
212 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
213 // Write Addresses to All Locations
214 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
215 /*
216 8'h0C:
217 begin
218     status <= `STATUS_WRITING;
219     fwdata <= 16'h40; // Issue "setup write" command
220     fop <= `FLASHOP_WRITE;
221     state <= state+1;
222 end
223
224 8'h0D:
225 begin
226     //fwdata <= 16'b10101; // Finish write
227     fwdata <= {8'b0, FPGA_ROM_data}; // Finish write
228     fop <= `FLASHOP_WRITE;
229     state <= state+1;
230 end
231 8'h0E:
232 begin
233     // Read status register
234     fop <= `FLASHOP_READ;
235     state <= state+1;
236 end
237 8'h0F:
```

238

```
261 ////////////////////////////////////////////////////  
262 // Read back data  
263 ////////////////////////////////////////////////////  
264  
265 8'h10:  
266 begin  
267     status <= `STATUS_READING;  
268     fwrdata <= 16'hFF; // Issue "read array" command  
269     fop <= `FLASHOP_WRITE;  
270     faddress <= `INIT_FLASH_ADDRESS;  
271     FPGA_ROM_addr <= 0;  
272     state <= state+1;  
273 end  
274 8'h11:  
275 begin  
276     fop <= `FLASHOP_READ;  
277     state <= state+1;  
278     status <= `STATUS_READOUT_DATA;  
279 end  
280 8'h12:  
281 begin  
282     fop <= `FLASHOP_IDLE;  
283     count <= count + 1;  
284     //status <= `STATUS_READOUT_DATA;  
285     state <= state+1;  
286 end  
287 8'h13:  
288 begin  
289     fop <= `FLASHOP_IDLE;  
290     //status <= `STATUS_READOUT_DATA;  
291     if (count == 26'd30000000)  
292         state <= state+1;  
293     else  
294         state <= state-1;  
295 end  
296 8'h14:  
297     if (faddress == `LAST_FLASH_ADDRESS)  
298         //if (faddress == 23'h7FFFFFFF)  
299             begin  
300                 fop <= `FLASHOP_IDLE;  
301                 faddress <= `INIT_FLASH_ADDRESS;  
302                 FPGA_ROM_addr <= 0;  
303                 state <= state+5;  
304             end  
305         else  
306             begin  
307                 faddress <= faddress+1;  
308                 FPGA_ROM_addr <= FPGA_ROM_addr+1;  
309                 count <= 0;  
310                 fop <= `FLASHOP_READ;  
311                 state <= state-2;  
312                 status <= `STATUS_READOUT_DATA;  
313             end  
314  
315 /* if (fwrdata == {8'b0, FPGA_ROM_data})  
316     if (faddress == `LAST_FLASH_ADDRESS)
```

317

```
335 */
336 /*
337 // Erase the chip again
338 // Erase the chip again
339
340 8'h13:
341 begin
342     status <= `STATUS_ERASING;
343     fwdata <= 16'h20; // Issue "erase block" command
344     fop <= `FLASHOP_WRITE;
345     state <= state+1;
346 end
347 8'h14:
348 begin
349     fwdata <= 16'hD0; // Issue "confirm erase" command
350     fop <= `FLASHOP_WRITE;
351     state <= state+1;
352 end
353 8'h15:
354 begin
355     fop <= `FLASHOP_READ;
356     state <= state+1;
357 end
358 8'h16:
359 if (frdata[7] == 1) // Done erasing block
360     if (frdata[6:1] == 0) // No errors
361         if (faddress != 23'h20000) // `LAST_BLOCK_ADDRESS
362             //if (faddress != `LAST_BLOCK_ADDRESS) // `LAST_BLOCK_ADDRESS
363             // if (faddress != 23'h7F0000) // `LAST_BLOCK_ADDRESS
364             // if (faddress != 23'h100) // `LAST_BLOCK_ADDRESS
365         begin
366             faddress <= faddress+`BLOCK_SIZE;
367             fop <= `FLASHOP_IDLE;
368             state <= state-3;
369         end
370         else
371         begin
372             faddress <= 0;
373             fop <= `FLASHOP_IDLE;
374             state <= state+1;
375         end
376         else // Erase error detected
377             status <= `STATUS_ERASE_BLOCK_ERROR;
378     else // Still busy
379         fop <= `FLASHOP_READ;
380 */
381 // Erase the chip again
382 // End of test: declare success
383 // Erase the chip again
384
385 8'h19:
386 begin
387     status <= `STATUS_SUCCESS;
388     fop <= `FLASHOP_IDLE;
389 end
390
391 endcase
392 else
393     fop <= `FLASHOP_IDLE;
394
395 // Erase the chip again
```



```

396 //
397 // Status display
398 //
399 ///////////////////////////////////////////////////////////////////
400 //
401 // "Reset    -----" --> During reset
402 // "Read ID  000000" --> While reading ID codes
403 // "Clr locks 000000" --> While clearing block locks
404 // "Erase    000000" --> While erasing
405 // "Write    000000" --> While writing
406 // "Read     000000" --> While reading
407 // " *** PASSED *** " --> If the entire test completes with no errors
408 // "Err: Manuf 0000" --> If an incorrect manufacturer code is read
409 // "Err: Size  0000" --> If an incorrect size code is read
410 // "Err: Locks 0000" --> If an error is detected when clearing the block lock bits
411 // "Err: Erase 0000"
412 // "Err: 000000 0000"
413
414
415 // Rd  000000 0000
416 // Wr  000000 0000
417 // Id  000000 0000
418 //
419
420 function [39:0] nib2char;
421     input [3:0] nib;
422     begin
423     case (nib)
424     4'h0: nib2char = 40'b00111110_01010001_01001001_01000101_00111110;
425     4'h1: nib2char = 40'b00000000_01000010_01111111_01000000_00000000;
426     4'h2: nib2char = 40'b01100010_01010001_01001001_01001001_01000110;
427     4'h3: nib2char = 40'b00100010_01000001_01001001_01001001_00110110;
428     4'h4: nib2char = 40'b00011000_00010100_00010010_01111111_00010000;
429     4'h5: nib2char = 40'b00100111_01000101_01000101_01000101_00111001;
430     4'h6: nib2char = 40'b00111100_01001010_01001001_01001001_00110000;
431     4'h7: nib2char = 40'b00000001_01110001_00001001_00000101_00000011;
432     4'h8: nib2char = 40'b00110110_01001001_01001001_01001001_00110110;
433     4'h9: nib2char = 40'b00000110_01001001_01001001_00101001_00011110;
434     4'hA: nib2char = 40'b01111110_00001001_00001001_00001001_01111110;
435     4'hB: nib2char = 40'b01111111_01001001_01001001_01001001_00110110;
436     4'hC: nib2char = 40'b00111110_01000001_01000001_01000001_00100010;
437     4'hD: nib2char = 40'b01111111_01000001_01000001_01000001_00111110;
438     4'hE: nib2char = 40'b01111111_01001001_01001001_01001001_01000001;
439     4'hF: nib2char = 40'b01111111_00001001_00001001_00001001_00000001;
440     endcase
441     end
442 endfunction
443
444 wire [159:0] data_dots;
445 assign data_dots = {nib2char(frdata[15:12]), nib2char(frdata[11:8]),
446     nib2char(frdata[7:4]), nib2char(frdata[3:0])};
447
448 wire [239:0] address_dots;
449 assign address_dots = {nib2char({ 1'b0, faddress[22:20]}),
450     nib2char(faddress[19:16]),
451     nib2char(faddress[15:12]),
452     nib2char(faddress[11:8]),
453     nib2char(faddress[7:4]),
454     nib2char(faddress[3:0])};
455
456 always @(status or address_dots or data_dots)
457     case (status)
458     `STATUS_RESET:
459     dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
460     40'b01111111_01001001_01001001_01001001_01000001, // E
461     40'b00100110_01001001_01001001_01001001_00110010, // S
462     40'b01111111_01001001_01001001_01001001_01000001, // E
463     40'b00000001_00000001_01111111_00000001_00000001, // T
464     40'b00000000_00000000_00000000_00000000_00000000, //
465     40'b00000000_00000000_00000000_00000000_00000000, //
466     40'b00000000_00000000_00000000_00000000_00000000, //
467     40'b00000000_00000000_00000000_00000000_00000000, //
468     40'b00000000_00000000_00000000_00000000_00000000, //
469     40'b00001000_00001000_00001000_00001000_00001000, // -
470     40'b00001000_00001000_00001000_00001000_00001000, // -
471     40'b00001000_00001000_00001000_00001000_00001000, // -
472     40'b00001000_00001000_00001000_00001000_00001000, // -
473     40'b00001000_00001000_00001000_00001000_00001000, // -
474     40'b00001000_00001000_00001000_00001000_00001000}; // -

```

```
475 `STATUS_READ_ID:
476 dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
477 40'b01111111_01001001_01001001_01001001_01000001, // E
478 40'b01111110_00001001_00001001_00001001_01111110, // A
479 40'b01111111_01000001_01000001_01000001_00111110, // D
480 40'b00000000_00000000_00000000_00000000_00000000, //
481 40'b00000000_01000001_01111111_01000001_00000000, // I
482 40'b01111111_01000001_01000001_01000001_00111110, // D
483 40'b00000000_00000000_00000000_00000000_00000000, //
484 40'b00000000_00000000_00000000_00000000_00000000, //
485 40'b00000000_00000000_00000000_00000000_00000000, //
486 address_dots};
487 `STATUS_CLEAR_LOCKS:
488 dots <= {40'b00111110_01000001_01000001_01000001_00100010, // C
489 40'b01111111_01000000_01000000_01000000_01000000, // L
490 40'b01111111_00001001_00011001_00101001_01000110, // R
491 40'b00000000_00000000_00000000_00000000_00000000, //
492 40'b01111111_01000000_01000000_01000000_01000000, // L
493 40'b00111110_01000001_01000001_01000001_00111110, // O
494 40'b00111110_01000001_01000001_01000001_00100010, // C
495 40'b01111111_00001000_00010100_00100010_01000001, // K
496 40'b00100110_01001001_01001001_01001001_00110010, // S
497 40'b00000000_00000000_00000000_00000000_00000000, //
498 address_dots};
499 `STATUS_ERASING:
500 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
501 40'b01111111_00001001_00011001_00101001_01000110, // R
502 40'b01111110_00001001_00001001_00001001_01111110, // A
503 40'b00100110_01001001_01001001_01001001_00110010, // S
504 40'b00000000_01000001_01111111_01000001_00000000, // I
505 40'b01111111_00000010_00000100_00001000_01111111, // N
506 40'b00111110_01000001_01001001_01001001_00111010, // G
507 40'b00000000_00000000_00000000_00000000_00000000, //
508 40'b00000000_00000000_00000000_00000000_00000000, //
509 40'b00000000_00000000_00000000_00000000_00000000, //
510 address_dots};
511 `STATUS_WRITING:
512 dots <= {40'b01111111_00100000_00011000_00100000_01111111, // W
513 40'b01111111_00001001_00011001_00101001_01000110, // R
514 40'b00000000_01000001_01111111_01000001_00000000, // I
515 40'b00000001_00000001_01111111_00000001_00000001, // T
516 40'b00000000_01000001_01111111_01000001_00000000, // I
517 40'b01111111_00000010_00000100_00001000_01111111, // N
518 40'b00111110_01000001_01001001_01001001_00111010, // G
519 40'b00000000_00000000_00000000_00000000_00000000, //
520 40'b00000000_00000000_00000000_00000000_00000000, //
521 40'b00000000_00000000_00000000_00000000_00000000, //
522 address_dots};
523 `STATUS_READING:
524 dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
525 40'b01111111_01001001_01001001_01001001_01000001, // E
526 40'b01111110_00001001_00001001_00001001_01111110, // A
527 40'b01111111_01000001_01000001_01000001_00111110, // D
528 40'b00000000_01000001_01111111_01000001_00000000, // I
529 40'b01111111_00000010_00000100_00001000_01111111, // N
530 40'b00111110_01000001_01001001_01001001_00111010, // G
531 40'b00000000_00000000_00000000_00000000_00000000, //
532 40'b00000000_00000000_00000000_00000000_00000000, //
533 40'b00000000_00000000_00000000_00000000_00000000, //
534 address_dots};
535 `STATUS_SUCCESS:
536 dots <= {40'b00000000_00000000_00000000_00000000_00000000, //
537 40'b00101010_00011100_01111111_00011100_00101010, // *
538 40'b00101010_00011100_01111111_00011100_00101010, // *
539 40'b00101010_00011100_01111111_00011100_00101010, // *
540 40'b00000000_00000000_00000000_00000000_00000000, //
541 40'b01111111_00001001_00001001_00001001_00000110, // P
542 40'b01111110_00001001_00001001_00001001_01111110, // A
543 40'b00100110_01001001_01001001_01001001_00110010, // S
544 40'b00100110_01001001_01001001_01001001_00110010, // S
545 40'b01111111_01001001_01001001_01001001_01000001, // E
546 40'b01111111_01000001_01000001_01000001_00111110, // D
547 40'b00000000_00000000_00000000_00000000_00000000, //
548 40'b00101010_00011100_01111111_00011100_00101010, // *
549 40'b00101010_00011100_01111111_00011100_00101010, // *
550 40'b00101010_00011100_01111111_00011100_00101010, // *
551 40'b00000000_00000000_00000000_00000000_00000000}; //
552 `STATUS_BAD_MANUFACTURER:
553 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
```

```
554 40'b01111111_00001001_00011001_00101001_01000110, // R
555 40'b01111111_00001001_00011001_00101001_01000110, // R
556 40'b00000000_00110110_00110110_00000000_00000000, // :
557 40'b00000000_00000000_00000000_00000000_00000000, //
558 40'b01111111_00000010_00001100_00000010_01111111, // M
559 40'b01111110_00001001_00001001_00001001_01111110, // A
560 40'b01111111_00000010_00001000_00001000_01111111, // N
561 40'b01111111_00001001_00001001_00001001_00000001, // U
562 40'b01111111_00001001_00001001_00001001_00000001, // F
563 40'b00000000_00000000_00000000_00000000_00000000, //
564 40'b00000000_00000000_00000000_00000000_00000000, //
565 data_dots};
566 `STATUS_BAD_SIZE:
567 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
568 40'b01111111_00001001_00011001_00101001_01000110, // R
569 40'b01111111_00001001_00011001_00101001_01000110, // R
570 40'b00000000_00110110_00110110_00000000_00000000, // :
571 40'b00000000_00000000_00000000_00000000_00000000, //
572 40'b00100110_01001001_01001001_01001001_00110010, // S
573 40'b00000000_01000001_01111111_01000001_00000000, // I
574 40'b01100001_01010001_01001001_01000101_01000011, // Z
575 40'b01111111_01001001_01001001_01001001_01000001, // E
576 40'b00000000_00000000_00000000_00000000_00000000,
577 40'b00000000_00000000_00000000_00000000_00000000,
578 40'b00000000_00000000_00000000_00000000_00000000,
579 data_dots};
580 `STATUS_LOCK_BIT_ERROR:
581 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
582 40'b01111111_00001001_00011001_00101001_01000110, // R
583 40'b01111111_00001001_00011001_00101001_01000110, // R
584 40'b00000000_00110110_00110110_00000000_00000000, // :
585 40'b00000000_00000000_00000000_00000000_00000000, //
586 40'b01111111_01000000_01000000_01000000_01000000, // L
587 40'b00111110_01000001_01000001_01000001_00111110, // O
588 40'b00111110_01000001_01000001_01000001_00100010, // C
589 40'b01111111_00001000_00010100_00100010_01000001, // K
590 40'b00100110_01001001_01001001_01001001_00110010, // S
591 40'b00000000_00000000_00000000_00000000_00000000,
592 40'b00000000_00000000_00000000_00000000_00000000,
593 data_dots};
594 `STATUS_ERASE_BLOCK_ERROR:
595 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
596 40'b01111111_00001001_00011001_00101001_01000110, // R
597 40'b01111111_00001001_00011001_00101001_01000110, // R
598 40'b00000000_00110110_00110110_00000000_00000000, // :
599 40'b00000000_00000000_00000000_00000000_00000000, //
600 40'b01111111_01001001_01001001_01001001_01000001, // E
601 40'b01111111_00001001_00011001_00101001_01000110, // R
602 40'b01111110_00001001_00001001_00001001_01111110, // A
603 40'b00100110_01001001_01001001_01001001_00110010, // S
604 40'b01111111_01001001_01001001_01001001_01000001, // E
605 40'b00000000_00000000_00000000_00000000_00000000,
606 40'b00000000_00000000_00000000_00000000_00000000,
607 data_dots};
608 `STATUS_WRITE_ERROR:
609 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
610 40'b01111111_00001001_00011001_00101001_01000110, // R
611 40'b01111111_00001001_00011001_00101001_01000110, // R
612 40'b00000000_00110110_00110110_00000000_00000000, // :
613 40'b00000000_00000000_00000000_00000000_00000000, //
614 40'b01111111_00100000_00011000_00100000_01111111, // W
615 40'b01111111_00001001_00011001_00101001_01000110, // R
616 40'b00000000_01000001_01111111_01000001_00000000, // I
617 40'b00000001_00000001_01111111_00000001_00000001, // T
618 40'b01111111_01001001_01001001_01001001_01000001, // E
619 40'b00000000_00000000_00000000_00000000_00000000,
620 40'b00000000_00000000_00000000_00000000_00000000,
621 data_dots};
622 `STATUS_READ_WRONG_DATA:
623 dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
624 40'b01111111_00001001_00011001_00101001_01000110, // R
625 40'b01111111_00001001_00011001_00101001_01000110, // R
626 40'b00000000_00110110_00110110_00000000_00000000, // :
627 40'b00000000_00000000_00000000_00000000_00000000, //
628 address_dots,
629 40'b00000000_00000000_00000000_00000000_00000000,
630 data_dots};
631 `STATUS_READOUT_DATA:
632 dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
```

```
633     40'b01111111_01001001_01001001_01001001_01000001, // E
634     40'b01111110_00001001_00001001_00001001_01111110, // A
635     40'b01111111_01000001_01000001_01000001_00111110, // D
636     40'b00000000_00000000_00000000_00000000_00000000, //
637     address_dots, // 6 digits
638     40'b00000000_00000000_00000000_00000000_00000000, //
639     data_dots; // 4 digits
640     default:
641     dots <= {16{40'b01010101_00101010_01010101_00101010_01010101}};
642     endcase
643
644 endmodule
645
646
647
648
649
650
651
```

```
1 module top(clk, reset, status, cs_bar, r_wbar, cs_bar_DA, ext_data, heartbeat, heartrate, digit2, digit1, digit0
, abnormal);
2
3 input clk, reset, status;
4 inout[7:0] ext_data;
5 output cs_bar, r_wbar, cs_bar_DA, heartbeat, abnormal;
6 output[3:0] digit2, digit1, digit0;
7 output[8:0] heartrate;
8
9 wire reset_sync, status_sync, sample, sram_we, heartbeat, convert, abnormal;
10 wire filter_start, filter_busy, DAC_start, analog_busy, ADC_start;
11 wire[7:0] sram_q, rom1_q, rom2_q, read_data, int_data;
12 wire[3:0] sram_addr, rom1_addr, rom2_addr;
13 wire[8:0] heartrate;
14
15 synchronizer synchronizer1(clk, reset, status, reset_sync, status_sync);
16 enable250Hz enable250Hz1(clk, reset_sync, sample);
17 majorFSM majorFSM1(clk, reset_sync, sample, filter_busy, analog_busy, filter_start, DAC_start);
18 rom1 rom11(rom1_addr, rom1_q);
19 rom2 rom22(rom2_addr, rom2_q);
20 sram sram1(sram_addr, clk, read_data, sram_we, sram_q);
21 analog analog1(clk, reset_sync, ADC_start, DAC_start, status_sync, int_data, analog_busy, r_wbar, cs_bar, cs_bar
_DA, ext_data, read_data);
22 filter filter1(clk, reset_sync, filter_start, sram_q, rom1_q, rom2_q, filter_busy, ADC_start, sram_we, sram_addr
, rom1_addr, rom2_addr, int_data);
23 heartbeat_detector heartbeat_detector1(clk, reset_sync, int_data, heartbeat);
24 heartrate_detector heartrate_detector1(clk, reset_sync, heartbeat, heartrate, convert);
25 bin2dec bin2dec1(clk, reset_sync, convert, heartrate, digit2, digit1, digit0);
26 abnormality_detector abnormality_detector1(clk, reset_sync, heartrate, abnormal);
27
28 endmodule
```

```
1 module wireless_analog(clk, reset_sync, sample_wireless, cs_bar_DA_wireless, rxdata, wireless_dataout);
2
3 input clk, reset_sync, sample_wireless;
4 input[7:0] rxdata;
5 output cs_bar_DA_wireless;
6 output[7:0] wireless_dataout;
7
8 reg[2:0] state, nextstate;
9 reg cs_bar_DA_wireless, cs_bar_DA_wireless_int, LE_DAC, LE_DAC_int;
10 wire[7:0] data;
11
12 assign wireless_dataout = LE_DAC ? rxdata : 8'hz;
13
14 parameter IDLE = 0;
15 parameter DAC0 = 1;
16 parameter DAC1 = 2;
17 parameter DAC2 = 3;
18 parameter DAC3 = 4;
19 parameter WAIT = 5;
20
21 always @ (posedge clk)
22 begin
23     cs_bar_DA_wireless <= cs_bar_DA_wireless_int;
24     LE_DAC <= LE_DAC_int;
25
26     if (!reset_sync) state <= IDLE;
27     else state <= nextstate;
28 end
29
30 always @ (state or sample_wireless)
31 begin
32     cs_bar_DA_wireless_int = 1;
33     LE_DAC_int = 0;
34
35     case (state)
36     IDLE: begin
37         nextstate = WAIT;
38     end
39
40     WAIT: begin
41         if (sample_wireless) nextstate = DAC0;
42         else nextstate = WAIT;
43     end
44
45     DAC0: begin
46         LE_DAC_int = 1;
47         nextstate = DAC1;
48     end
49
50     DAC1: begin
51         LE_DAC_int = 1;
52         cs_bar_DA_wireless_int = 0;
53         nextstate = DAC2;
54     end
55
56     DAC2: begin
57         LE_DAC_int = 1;
58         nextstate = DAC3;
59     end
60
61     DAC3: begin
62         LE_DAC_int = 0;
63         nextstate = WAIT;
64     end
65
66     default: nextstate = IDLE;
67 endcase
68 end
69
70 endmodule
```

```

// TRANSMITTER MODULE of Wireless Musical EKG
// Test Data is incrementing in a loop 00-FF
// Transmitting the Test Data at 80ms period

// Notes for myself
// RF Frequency of 915 MHz
// Included pathways.. U:\Keil\C51\INC; U:\Keil\C51\LIB
// I included the testdata used to send over a package of
// three values.

#include <chipcon/regl010.h>
#include <chipcon/ccl010eb.h>
#include <chipcon/hal.h>
#include <stdio.h>

// Protocol constants
#define PREAMBLE_BYTE_COUNT 7

// Test packet
#define TEST_STRING_LENGTH 1
byte test_data[TEST_STRING_LENGTH];

// Initiate halTimer
byte halTimerUsage;

//byte adc_data[TEST_STRING_LENGTH];

void main(void) {

#ifdef FREQ915

// X-tal frequency: 14.745600 MHz
// RF Frequency A: 915.027455 MHz      Rx
// RF Frequency B: 915.027455 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 4 dBm
// IF/RSSI: RSSI Enabled

RF_RTXPAIR_SETTINGS code RF_SETTINGS = {
    0x4B, 0x2F, 0x15, // Modem 0, 1 and 2
    0xAA, 0x80, 0x00, // Freq A
    0x5C, 0xF4, 0x02, // Freq B
    0x01, 0xAB, // FSEP 1 and 0
    0x58, // PLL_RX
    0x30, // PLL_TX
    0x6C, // CURRENT_RX
    0xF3, // CURRENT_TX
    0x32, // FREQD
    0xFF, // PA_POW
    0x00, // MATCH
    0x00, // PRESCALER
};

#endif

#ifdef FREQ868

// X-tal frequency: 14.745600 MHz
// RF frequency A: 868.277200 MHz      Rx
// RF frequency B: 868.277200 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 4 dBm
// IF/RSSI: RSSI Enabled

RF_RTXPAIR_SETTINGS code RF_SETTINGS = {
    0x4B, 0x2F, 0x15, // Modem 0, 1 and 2: Manchester, 2.4 kBaud
    //0x43, 0x2F, 0x15, // Modem 0, 1 and 2: NRZ, 2.4 kBaud
    //0xA1, 0x2F, 0x29, // Modem 0, 1 and 2: NRZ, 38.4 kBaud
    //0xA0, 0x2F, 0x52, // Modem 0, 1 and 2: NRZ, 76.8 kBaud
    0x75, 0xA0, 0x00, // Freq A
    0x58, 0x32, 0x8D, // Freq B
    0x01, 0xAB, // FSEP 1 and 0
    0x40, // PLL_RX
    0x30, // PLL_TX
    0x6C, // CURRENT_RX
    0xF3, // CURRENT_TX
    0x32, // FREQD
    0xFF, // PA_POW
    0x00, // MATCH
    0x00, // PRESCALER
};

#endif

#ifdef FREQ433

// X-tal frequency: 14.745600 MHz
// RF frequency A: 433.302000 MHz      Rx
// RF frequency B: 433.302000 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 10 dBm
// IF/RSSI: RSSI Enabled

RF_RTXPAIR_SETTINGS code RF_SETTINGS = {
    0x4B, 0x2F, 0x0E, // Modem 0, 1 and 2
    0x58, 0x00, 0x00, // Freq A
    0x41, 0xFC, 0x9C, // Freq B
    0x02, 0x80, // FSEP 1 and 0
    0x60, // PLL_RX
    0x48, // PLL_TX
    0x44, // CURRENT_RX
    0x81, // CURRENT_TX
    0x0A, // FREQD
    0xFF, // PA_POW
    0xC0, // MATCH
    0x00, // PRESCALER
};

#endif

// Calibration data
RF_RTXPAIR_CALDATA xdata RF_CALDATA;

// Disable watchdog timer
WDT_ENABLE(FALSE);

// Set optimum settings for speed and low power consumption
MEM_NO_WAIT_STATES();
FLASH_SET_POWER_MODE(FLASH_STANDBY_BETWEEN_READS);

```

```

// Calibrate
halRFCalib(&RF_SETTINGS, &RF_CALDATA);

// Turn on RF for TX, send packet
halRFSetRxTxOff(RF_TX, &RF_SETTINGS, &RF_CALDATA);

// Setup UART0 with polled I/O
UART0_SETUP(57600, CC1010EB_CLKFREQ, UART_NO_PARITY | UART_RX_TX | UART_POLLED);

WDT_ENABLE(FALSE);
RLED_OE(TRUE);
YLED_OE(TRUE);
GLED_OE(TRUE);
BLED_OE(TRUE);

RLED = LED_OFF;
YLED = LED_OFF;
GLED = LED_ON;
BLED = LED_OFF;

//FOR TRANSMITTER
// Setup ADC, turn it on
//halConfigADC(ADC_MODE_SINGLE | ADC_REFERENCE_VDD, CC1010EB_CLKFREQ, 0); // 0: Threshold value (not used in this program)
//ADC_POWER(TRUE); // Power up ADC from sleep mode

test_data[0] = 0;
while (TRUE)
{
    YLED=-YLED;
    /*
    ADC_SELECT_INPUT(ADC_INPUT_AD0);
    adc_data[0] = ADC_GET_SAMPLE_8BIT();
    ADC_SAMPLE_SINGLE();
    printf("building %03X from ADC\n",adc_data[0]);
    halRFSendPacket(PREAMBLE_BYTE_COUNT, &adc_data[0], 1);
    */

    test_data[0]++;
    //test_data[1] = 255;
    //test_data[2] = 128;

    halRFSendPacket(PREAMBLE_BYTE_COUNT, &test_data[0], 1);

    //printf("data_0 %X \n",test_data[0]);
    //printf("data_1 %X \n",test_data[1]);
    //printf("data_2 %X \n",test_data[2]);
}
}

void halConfigADC(byte options, word clkFreq, byte threshold) {
    clkFreq= (clkFreq-250*8)/(250*16);
    ADCON2=(options&0x80)|((byte)clkFreq&0x3F);
    if (options&ADC_INTERRUPT_ENABLE) {
        EXIFR=-0x40;
        ADIE=1;
    }
    ADCON=0x80|(options&0x38);
    ADTRH=threshold;
}

```



```

// RECEIVER MODULE for Wireless Musical EKG

#include <chipcon/regl010.h>
#include <chipcon/cc1010eb.h>
#include <chipcon/hal.h>
#include <stdio.h>

// Sending packages with one test string (8-bit extracted
// value from adc)
#define RF_RX_BUF_SIZE      50
#define TEST_STRING_LENGTH  1

//Define ASCII codes / terminal commands
#define ASCII_LF      0x0A
#define ASCII_CR      0x0D
#define ASCII_NUL     0x00
#define ASCII_ESC     0x1B

// Define RSSI limit for response
#define RSSI_LIM      -75

// Initiated Timer of Hal from Lib
byte halTimer1Usage;

// Define monitor timeouts
#define RSSI_MONITOR_TIMEOUT  0x0110
#define MAIN_MONITOR_TIMEOUT  0x00FF

void RFSetupReceive(void);

int main_monitor = 0;
int rssi_monitor = 0;
char rssi_val;
int i = 0;

byte rf_rx_string[TEST_STRING_LENGTH];
byte rf_rx_buf[RF_RX_BUF_SIZE];
byte rf_rx_index = 0;

byte rf_rx_display = FALSE;
int packet_error_cnt = 0;
byte packet_error = FALSE;

void main() {

#ifdef FREQ868

// X-tal frequency: 14.745600 MHz
// RF Frequency A: 868.277200 MHz      Rx
// RF Frequency B: 868.277200 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 4 dBm
// IF/RSSI: RSSI Enabled

RF_RTXPAIR_SETTINGS code RF_SETTINGS = {
0x4B, 0x2F, 0x15, // Modem 0, 1 and 2: Manchester, 2.4 kBaud
//0x43, 0x2F, 0x15, // Modem 0, 1 and 2: NRZ, 2.4 kBaud
//0xA1, 0x2F, 0x29, // Modem 0, 1 and 2: NRZ, 38.4 kBaud
//0xA0, 0x2F, 0x52, // Modem 0, 1 and 2: NRZ, 76.8 kBaud
0x75, 0xA0, 0x00, // Freq A
0x58, 0x32, 0x8D, // Freq B
0x01, 0xAB, // FSEP 1 and 0
0x40, // PLL_RX
0x30, // PLL_TX
0x6C, // CURRENT_RX
0xF3, // CURRENT_TX
0x32, // FREQD
0xFF, // PA_POW
0x00, // MATCH
0x00, // PRESCALER
};

#endif

#ifdef FREQ915

// X-tal frequency: 14.745600 MHz
// RF frequency A: 915.027455 MHz      Rx
// RF frequency B: 915.027455 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 4 dBm
// IF/RSSI: RSSI Enabled

RF_RTXPAIR_SETTINGS code RF_SETTINGS = {
0x4B, 0x2F, 0x15, // Modem 0, 1 and 2
0xA8, 0xB0, 0x00, // Freq A
0x5C, 0xF4, 0x02, // Freq B
0x01, 0xAB, // FSEP 1 and 0
0x58, // PLL_RX
0x30, // PLL_TX
0x6C, // CURRENT_RX
0xF3, // CURRENT_TX
0x32, // FREQD
0xFF, // PA_POW
0x00, // MATCH
0x00, // PRESCALER
};

#endif

#ifdef FREQ433

// X-tal frequency: 14.745600 MHz
// RF frequency A: 433.302000 MHz      Rx
// RF frequency B: 433.302000 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 10 dBm
// IF/RSSI: RSSI Enabled

RF_RTXPAIR_SETTINGS code RF_SETTINGS = {
0x4B, 0x2F, 0x0E, // Modem 0, 1 and 2
0x58, 0x00, 0x00, // Freq A
0x41, 0xFC, 0x9C, // Freq B
0x02, 0x80, // FSEP 1 and 0
0x60, // PLL_RX
0x48, // PLL_TX
0x44, // CURRENT_RX
0x81, // CURRENT_TX
0x0A, // FREQD
0xFF, // PA_POW
};

```

```

    0xC0,          // MATCH
    0x00,          // PRESCALER
};

#endif

// Calibration data
RF_RXTXPAIR_CALDATA xdata RF_CALDATA;

// Fill up reference string
for(i = 0; i < TEST_STRING_LENGTH; i++){
    rf_rx_string[i]=i;
}

// Disable watchdog timer
WDT_ENABLE(FALSE);

// Set optimum settings for speed and low power consumption
MEM_NO_WAIT_STATES();
FLASH_SET_POWER_MODE(FLASH_STANDBY_BETWEEN_READS);

// Calibrate
halRFCalib(&RF_SETTINGS, &RF_CALDATA);

// Change Baud Rate
//halRFOVERRIDEBaudRate(RF_1200B);

// Turn on RF for RX
halRFSetRXTxOff(RF_RX, &RF_SETTINGS, &RF_CALDATA);

// Setup UART0 with polled I/O
UART0_SETUP(57600, CC1010EB_CLKFREQ, UART_NO_PARITY | UART_RX_TX | UART_POLLED);

// Enable the LEDs
RLED_ON(TRUE);
YLED_ON(TRUE);
GLED_ON(TRUE);
BLED_ON(TRUE);

RLED = LED_OFF;
YLED = LED_OFF;
GLED = LED_OFF;
BLED = LED_OFF;

// Setup RF receive
RFSetupReceive();

// Reset main loop monitor
main_monitor = MAIN_MONITOR_TIMEOUT;

// Reset RSSI monitor
rssi_monitor = RSSI_MONITOR_TIMEOUT;

// Clear screen
//////printf("%c[2J", ASCII_ESC);

// Place cursor at upper left corner
//////printf("%c[H", ASCII_ESC);

// Display RF packet data + RSSI level:
while (TRUE) {

    // Detect RF packet error and display RF packet data:
    if(rf_rx_display == TRUE){
        rf_rx_display = FALSE;
        for(i = 2; i < TEST_STRING_LENGTH+2; i++){
            if(rf_rx_buf[i] != i-2){
                packet_error = TRUE;
                RLED = LED_ON;
            }
            ////printf("RXD%d=%X ", i-2, (int)rf_rx_buf[i]);
            putchar (rf_rx_buf[i]);
        }

        if(packet_error == TRUE){
            packet_error_cnt++;
        }
        packet_error = FALSE;

        ////printf("\n");
    }else{

        // Indicate main loop is running ok:
        if (main_monitor-- < 0x0000) {
            BLED = !BLED;
            main_monitor = MAIN_MONITOR_TIMEOUT;
        }
    }
}

// Power down ADC:
// In this application example this instruction does not have
// any effect, since it is placed outside the infinite loop above.
// However, if the "real" application needs to, for instance,
// enter a power save mode, then it might be important power down
// the ADC to save as much power as possible.
ADC_POWER(FALSE);
}

// RF interrupt service routine:
void RF_ISR (void) interrupt INUM_RF {

    INT_ENABLE(INUM_RF, INT_OFF);
    INT_SETFLAG (INUM_RF, INT_CLR);

    // Get RF receive data
    rf_rx_buf[rf_rx_index] = RF_RECEIVE_BYTE();

    RLED = LED_OFF;

    // Verify packet contents:
    switch(rf_rx_index){
        case 0:
            RF_LOCK_AVERAGE_FILTER(TRUE);
            if(rf_rx_buf[rf_rx_index] != RF_SUITABLE_SYNC_BYTE){
                RLED = LED_ON;
                // YLED = LED_ON;
            }else{
            }
            rf_rx_index++;
            break;

        case 1:
            if(rf_rx_buf[rf_rx_index] != TEST_STRING_LENGTH){
                RLED = LED_ON;
            }
    }
}

```

```

    }else{
    }
    rf_rx_index++;
    break;

case TEST_STRING_LENGTH+3:
    rf_rx_index = 0;
    rf_rx_display = TRUE;
    // RLED = LED_ON;
    PDET &= ~0x80;
    PDET |= 0x80;
    break;

default:
    if((rf_rx_buf[rf_rx_index] != rf_rx_string[rf_rx_index-2]) && (rf_rx_index < TEST_STRING_LENGTH+2)){
        RLED = LED_ON;
    }else{
    }
    rf_rx_index++;
    break;
}

// Indicate packet reception
GLED = !GLED;

INT_ENABLE(INUM_RF, INT_ON);

return;
}

// Setup RF for RX
void RFSetupReceive (void) {

// Disable global interrupt
INT_GLOBAL_ENABLE (INT_OFF);

// Setup RF interrupt
INT_SETFLAG (INUM_RF, INT_CLR);
INT_PRIORITY (INUM_RF, INT_HIGH);
INT_ENABLE (INUM_RF, INT_ON);

// Enable RF interrupt based on bytemode
RF_SET_BYTEMODE();

// Setup preamble configuration
RF_SET_PREAMBLE_COUNT(16);
RF_SET_SYNC_BYTE(RF_SUITABLE_SYNC_BYTE);

// Make sure avg filter is free-running + 22 baud settling time
MODEM1=(MODEM1&0x03)|0x24;

// Reset preamble detection
PDET &= ~0x80;
PDET |= 0x80;

// Start RX
RF_START_RX();

// Enable global interrupt
INT_GLOBAL_ENABLE (INT_ON);
}

/*
extern byte xdata hal_dataformat_override,
                hal_baudrate_override,
                hal_pa_pow_override,
                hal_modem0_original,
                hal_pa_pow_original;

*/
//void halRFOverrideBaudRate(byte baudRate) {
//    if (hal_baudrate_override==baudRate)
//        MODEM0=(MODEM0&0x1F)|(baudRate&0xE0/*0x70*/ );
//    //Torgeir

//    else
//        MODEM0=(MODEM0&0x1F)|(hal_modem0_original&0xE0/*0x70*/ );
//    //Torgeir
//}

```

```

// TRANSMITTER MODULE of Wireless Musical EKG
// Included ADC on the EB (Evaluation Board)
// test_data is in a loop, incrementing by one 0x00-0xFF
// Increments at 12.5Hz (80ms period)

// RF Frequency of 915 MHz
// Included pathways: U:\Keil\C51\INC; U:\Keil\C51\LIB

#include <chipcon/reg1010.h>
#include <chipcon/cc1010eb.h>
#include <chipcon/hal.h>
#include <stdio.h>

// Protocol constants
#define PREAMBLE_BYTE_COUNT 7

// Test packet
#define TEST_STRING_LENGTH 1

// Initiate halTimer
byte halTimerUsage;
byte adc_data[TEST_STRING_LENGTH];

void main(void) {
#ifdef FREQ915
// X-tal frequency: 14.745600 MHz
// RF frequency A: 915.027455 MHz      Rx
// RF frequency B: 915.027455 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 4 dBm
// IF/RSSI: RSSI Enabled

RF_RXTXPAIR_SETTINGS code RF_SETTINGS = {
    0x4B, 0x2F, 0x15, // Modem 0, 1 and 2
    0xAA, 0x80, 0x00, // Freq A
    0x5C, 0xF4, 0x02, // Freq B
    0x01, 0xAB, // FSEP 1 and 0
    0x58, // PLL_RX
    0x30, // PLL_TX
    0x6C, // CURRENT_RX
    0xF3, // CURRENT_TX
    0x32, // FREQD
    0xFF, // PA_POW
    0x00, // MATCH
    0x00, // PRESCALER
};

#endif

#ifdef FREQ868
// X-tal frequency: 14.745600 MHz
// RF Frequency A: 868.277200 MHz      Rx
// RF Frequency B: 868.277200 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 4 dBm
// IF/RSSI: RSSI Enabled

RF_RXTXPAIR_SETTINGS code RF_SETTINGS = {
    0x4B, 0x2F, 0x15, // Modem 0, 1 and 2: Manchester, 2.4 kBaud
    //0x43, 0x2F, 0x15, // Modem 0, 1 and 2: NRZ, 2.4 kBaud
    //0xA1, 0x2F, 0x29, // Modem 0, 1 and 2: NRZ, 38.4 kBaud
    //0xA0, 0x2F, 0x52, // Modem 0, 1 and 2: NRZ, 76.8 kBaud
    0x75, 0xA0, 0x00, // Freq A
    0x58, 0x32, 0x8D, // Freq B
    0x01, 0xAB, // FSEP 1 and 0
    0x40, // PLL_RX
    0x30, // PLL_TX
    0x6C, // CURRENT_RX
    0xF3, // CURRENT_TX
    0x32, // FREQD
    0xFF, // PA_POW
    0x00, // MATCH
    0x00, // PRESCALER
};

#endif

#ifdef FREQ433
// X-tal frequency: 14.745600 MHz
// RF Frequency A: 433.302000 MHz      Rx
// RF Frequency B: 433.302000 MHz      Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 10 dBm
// IF/RSSI: RSSI Enabled

RF_RXTXPAIR_SETTINGS code RF_SETTINGS = {
    0x4B, 0x2F, 0x0E, // Modem 0, 1 and 2
    0x58, 0x00, 0x00, // Freq A
    0x41, 0xFC, 0x9C, // Freq B
    0x02, 0x80, // FSEP 1 and 0
    0x60, // PLL_RX
    0x48, // PLL_TX
    0x44, // CURRENT_RX
    0x81, // CURRENT_TX
    0x0A, // FREQD
    0xFF, // PA_POW
    0xC0, // MATCH
    0x00, // PRESCALER
};

#endif

// Calibration data
RF_RXTXPAIR_CALDATA xdata RF_CALDATA;

// Disable watchdog timer
WDT_ENABLE(FALSE);

// Set optimum settings for speed and low power consumption
MEM_NO_WAIT_STATES();
FLASH_SET_POWER_MODE(FLASH_STANDBY_BETWEEN_READS);

// Calibrate
halRFCalib(&RF_SETTINGS, &RF_CALDATA);

// Change Baud Rate

```

```

//halRfOverrideBaudRate(RF_19200B);

// Turn on RF for TX, send packet
halRfSetRxtxoff(RF_TX, &RF_SETTINGS, &RF_CALDATA);

// Setup UART0 with polled I/O
UART0_SETUP(57600, CC1010EB_CLKFREQ, UART_NO_PARITY | UART_RX_TX | UART_POLLED);

WDT_ENABLE(FALSE);
RLED_OE(TRUE);
YLED_OE(TRUE);
GLED_OE(TRUE);
BLED_OE(TRUE);

RLED = LED_OFF;
YLED = LED_OFF;
GLED = LED_ON;
BLED = LED_OFF;

//FOR TRANSMITTER

//Setup ADC, turn it on
halConfigADC(ADC_MODE_SINGLE | ADC_REFERENCE_VDD, CC1010EB_CLKFREQ, 0); // 0: Threshold value (not used in this program)
ADC_POWER(TRUE); // Power up ADC from sleep mode

// test_data[0] = 0;
while (TRUE)
{
    YLED=-YLED;

    ADC_SELECT_INPUT(ADC_INPUT_AD0);
    adc_data[0] = ADC_GET_SAMPLE_8BIT();
    ADC_SAMPLE_SINGLE();
    //printf("building %03X from ADC\n",adc_data[0]);

    halRfSendPacket(PREAMBLE_BYTE_COUNT, &adc_data[0], 1);

    //test_data[0]++;
    //test_data[1] = 255;
    //test_data[2] = 128;

    // halRfSendPacket(PREAMBLE_BYTE_COUNT, &test_data[0], 1);

    // printf("data_0 %X \n",test_data[0]);
    //printf("data_1 %X \n",test_data[1]);
    //printf("data_2 %X \n",test_data[2]);
}

}

void halConfigADC(byte options, word clkFreq, byte threshold) {
    clkFreq= (clkFreq-250*8)/(250*16);
    ADCON2=(options&0x80)|((byte)clkFreq&0x3F);
    if (options&ADC_INTERRUPT_ENABLE) {
        EXIF&=-0x40;
        ADIE=1;
    }
    ADCON=0x80|(options&0x38);
    ADTRH=threshold;
}

/*
extern byte xdata hal_dataformat_override,
hal_baudrate_override,
hal_pa_pow_override,
hal_modem0_original,
hal_pa_pow_original;
*/
void halRfOverrideBaudRate(byte baudRate) {
    if (hal_baudrate_override==baudRate)
// MODEM0=(MODEM0&0x1F)|((baudRate&0xE0/*0x70*/ //);
//Torgeir
//
// else
// MODEM0=(MODEM0&0x1F)|(hal_modem0_original&0xE0/*0x70*/ //);
//Torgeir
//}

```