```
 1  `timescale 1ns / 1ps
 2
 3  module suboptimal_decoder(reset, clk, num_fhts, symbol_in, bits_out,
    ready, first_upper_corr_sum, first_lower_corr_sum, ctr,
    first_big_corr_idx, first_big_corr_val, upper_corr, lower_corr, idx);
 4    input reset, clk;
 5    input [2:0] num_fhts;
 6    parameter WIDTH=10;
 7    input [WIDTH-1:0] symbol_in;
 8    output [5:0] bits_out;
 9    output ready;
10    output [15:0] first_upper_corr_sum, first_lower_corr_sum;
11    output [6:0] ctr;
12    output [2:0] first_big_corr_idx;
13    output [WIDTH-1:0] first_big_corr_val;
14    output [WIDTH-1:0] upper_corr, lower_corr;
15    output [5:0] idx;
16    wire [WIDTH-1:0] upper_corr, lower_corr;
17
18    reg ready;
19     reg ready_thrown;
20
21    wire [15:0] reset_fhts;
22    reg [WIDTH-1:0] symbols [63:0];
23
24    // Chagned
25    wire [15:0] fht_dones;
26    reg [WIDTH*16-1:0] upper_inputs, lower_inputs;
27    wire [WIDTH*16-1:0] upper_corrs, lower_corrs;
28    assign upper_corr = upper_corrs[9:0];
29    assign lower_corr = lower_corrs[9:0];
30
31
32      // 8 FHTs for first 3 bits, 8 FHTs for last 3 bits
33    fht_8_symbol fht[15:0](reset, clk, upper_inputs, lower_inputs,
    upper_corrs, lower_corrs, fht_dones);
34
35    wire [15:0] first_upper_corr_sum, first_lower_corr_sum,
    last_upper_corr_sum, last_lower_corr_sum;
36    assign first_upper_corr_sum = upper_corrs[1*WIDTH-1:0*WIDTH]
37                        + upper_corrs[2*WIDTH-1:1*WIDTH]
38                        + upper_corrs[3*WIDTH-1:2*WIDTH]
39                        + upper_corrs[4*WIDTH-1:3*WIDTH]
40                        + upper_corrs[5*WIDTH-1:4*WIDTH]
41                        + upper_corrs[6*WIDTH-1:5*WIDTH]
42                        + upper_corrs[7*WIDTH-1:6*WIDTH]
43                        + upper_corrs[8*WIDTH-1:7*WIDTH];
44    assign first_lower_corr_sum = lower_corrs[1*WIDTH-1:0*WIDTH]
45                        + lower_corrs[2*WIDTH-1:1*WIDTH]
46                        + lower_corrs[3*WIDTH-1:2*WIDTH]
47                        + lower_corrs[4*WIDTH-1:3*WIDTH]
48                        + lower_corrs[5*WIDTH-1:4*WIDTH]
49                        + lower_corrs[6*WIDTH-1:5*WIDTH]
50                        + lower_corrs[7*WIDTH-1:6*WIDTH]
51                        + lower_corrs[8*WIDTH-1:7*WIDTH];
52
```

```
   53   assign last_upper_corr_sum = upper_corrs[9*WIDTH-1:8*WIDTH]
   54                       + upper_corrs[10*WIDTH-1:9*WIDTH]
   55                       + upper_corrs[11*WIDTH-1:10*WIDTH]
   56                       + upper_corrs[12*WIDTH-1:11*WIDTH]
   57                       + upper_corrs[13*WIDTH-1:12*WIDTH]
   58                       + upper_corrs[14*WIDTH-1:13*WIDTH]
   59                       + upper_corrs[15*WIDTH-1:14*WIDTH]
   60                       + upper_corrs[16*WIDTH-1:15*WIDTH];
   61   assign last_lower_corr_sum = lower_corrs[9*WIDTH-1:8*WIDTH]
   62                       + lower_corrs[10*WIDTH-1:9*WIDTH]
   63                       + lower_corrs[11*WIDTH-1:10*WIDTH]
   64                       + lower_corrs[12*WIDTH-1:11*WIDTH]
   65                       + lower_corrs[13*WIDTH-1:12*WIDTH]
   66                       + lower_corrs[14*WIDTH-1:13*WIDTH]
   67                       + lower_corrs[15*WIDTH-1:14*WIDTH]
   68                       + lower_corrs[16*WIDTH-1:15*WIDTH];
   69
   70   reg [6:0] ctr;
   71   integer i;
   72   reg [2:0] first_big_corr_idx;
   73   reg [WIDTH-1:0] first_big_corr_val;
   74   reg [2:0] last_big_corr_idx;
   75   reg [WIDTH-1:0] last_big_corr_val;
   76   wire [5:0] bits_out;
   77   assign bits_out = ready ? {last_big_corr_idx[2:0],
    first_big_corr_idx[2:0]} : 6'b0;
   78
   79   always @(posedge clk) begin
   80     if (reset) begin
   81       ctr <= 7'b0;
   82       first_big_corr_idx <= 3'b0;
   83       first_big_corr_val <= 10'b0;
   84       last_big_corr_idx <= 3'b0;
   85       last_big_corr_val <= 10'b0;
   86       ready <= 0;
   87       ready_thrown <= 0;
   88     end else begin
   89       // Cycles 0-63 -- buffer symbols coming in
   90       symbols[ctr-1] <= symbol_in;
   91       // Cycles 67-70 -- get output from FHTs
   92       if (ctr <= 66) begin
   93         first_big_corr_idx <= 3'b0;
   94         first_big_corr_val <= 10'b0;
   95         last_big_corr_idx <= 3'b0;
   96         last_big_corr_val <= 10'b0;
   97       end else begin
   98         first_big_corr_idx <= first_upper_corr_sum > first_big_corr_val
    && first_upper_corr_sum > first_lower_corr_sum ? (ctr - 67) * 2 :
   99           (first_lower_corr_sum > first_big_corr_val) ? (ctr - 67) * 2
    + 1:
  100           first_big_corr_idx;
  101         first_big_corr_val <= (first_upper_corr_sum > first_big_corr_val
    && first_upper_corr_sum > first_lower_corr_sum) ? first_upper_corr_sum :
  102           (first_lower_corr_sum > first_big_corr_val) ?
    first_lower_corr_sum :
  103           first_big_corr_val;
```

```
104         last_big_corr_idx <= last_upper_corr_sum > last_big_corr_val &&
    last_upper_corr_sum > last_lower_corr_sum ? (ctr - 67) * 2 :
105             (last_lower_corr_sum > last_big_corr_val) ? (ctr - 67) * 2 +
    1 :
106             last_big_corr_idx;
107         last_big_corr_val <= (last_upper_corr_sum > last_big_corr_val &&
    last_upper_corr_sum > last_lower_corr_sum) ? last_upper_corr_sum :
108             (last_lower_corr_sum > last_big_corr_val) ?
    last_lower_corr_sum :
109             last_big_corr_val;
110        end
111       if (ctr > 69) begin
112         ctr <= 0;
113         if (!ready_thrown) begin
114           ready <= 1;
115         end
116         ready_thrown <= 1;
117       end else begin
118         ready <= 0;
119         ctr <= ctr + 1;
120       end
121     end
122   end
123   reg [5:0] idx;
124   always @(ctr) begin
125
126     // Cycles 64-67 -- feed FHTs input
127     if (ctr > 63 && ctr < 68) begin
128       // First three bits' inputs
129         idx = 0+8*(ctr-64);
130         upper_inputs[WIDTH*(0+1)-1:WIDTH*0] = symbols[0+8*(ctr-64)];
131         lower_inputs[WIDTH*(0+1)-1:WIDTH*0] = symbols[0+8*(ctr-64)+32];
132         upper_inputs[WIDTH*(1+1)-1:WIDTH*1] = symbols[1+8*(ctr-64)];
133         lower_inputs[WIDTH*(1+1)-1:WIDTH*1] = symbols[1+8*(ctr-64)+32];
134         upper_inputs[WIDTH*(2+1)-1:WIDTH*2] = symbols[2+8*(ctr-64)];
135         lower_inputs[WIDTH*(2+1)-1:WIDTH*2] = symbols[2+8*(ctr-64)+32];
136         upper_inputs[WIDTH*(3+1)-1:WIDTH*3] = symbols[3+8*(ctr-64)];
137         lower_inputs[WIDTH*(3+1)-1:WIDTH*3] = symbols[3+8*(ctr-64)+32];
138         upper_inputs[WIDTH*(4+1)-1:WIDTH*4] = symbols[4+8*(ctr-64)];
139         lower_inputs[WIDTH*(4+1)-1:WIDTH*4] = symbols[4+8*(ctr-64)+32];
140         upper_inputs[WIDTH*(5+1)-1:WIDTH*5] = symbols[5+8*(ctr-64)];
141         lower_inputs[WIDTH*(5+1)-1:WIDTH*5] = symbols[5+8*(ctr-64)+32];
142         upper_inputs[WIDTH*(6+1)-1:WIDTH*6] = symbols[6+8*(ctr-64)];
143         lower_inputs[WIDTH*(6+1)-1:WIDTH*6] = symbols[6+8*(ctr-64)+32];
144         upper_inputs[WIDTH*(7+1)-1:WIDTH*7] = symbols[7+8*(ctr-64)];
145         lower_inputs[WIDTH*(7+1)-1:WIDTH*7] = symbols[7+8*(ctr-64)+32];
146       // Second three bits
147         upper_inputs[WIDTH*(0+9)-1:WIDTH*8] = symbols[(0*8)+(ctr-64)];
148         lower_inputs[WIDTH*(0+9)-1:WIDTH*8] = symbols[(0*8)+(ctr-64)+4];
149         upper_inputs[WIDTH*(1+9)-1:WIDTH*9] = symbols[(1*8)+(ctr-64)];
150         lower_inputs[WIDTH*(1+9)-1:WIDTH*9] = symbols[(1*8)+(ctr-64)+4];
151         upper_inputs[WIDTH*(2+9)-1:WIDTH*10] = symbols[(2*8)+(ctr-64)];
152         lower_inputs[WIDTH*(2+9)-1:WIDTH*10] =
    symbols[(2*8)+(ctr-64)+4];
153         upper_inputs[WIDTH*(3+9)-1:WIDTH*11] = symbols[(3*8)+(ctr-64)];
154         lower_inputs[WIDTH*(3+9)-1:WIDTH*11] =
```

```
  154  symbols[(3*8)+(ctr-64)+4];
  155          upper_inputs[WIDTH*(4+9)-1:WIDTH*12] = symbols[(4*8)+(ctr-64)];
  156          lower_inputs[WIDTH*(4+9)-1:WIDTH*12] =
       symbols[(4*8)+(ctr-64)+4];
  157          upper_inputs[WIDTH*(5+9)-1:WIDTH*13] = symbols[(5*8)+(ctr-64)];
  158          lower_inputs[WIDTH*(5+9)-1:WIDTH*13] =
       symbols[(5*8)+(ctr-64)+4];
  159          upper_inputs[WIDTH*(6+9)-1:WIDTH*14] = symbols[(6*8)+(ctr-64)];
  160          lower_inputs[WIDTH*(6+9)-1:WIDTH*14] =
       symbols[(6*8)+(ctr-64)+4];
  161          upper_inputs[WIDTH*(7+9)-1:WIDTH*15] = symbols[(7*8)+(ctr-64)];
  162          lower_inputs[WIDTH*(7+9)-1:WIDTH*15] =
       symbols[(7*8)+(ctr-64)+4];
  163      end
  164    end
  165  endmodule
```

```
 1 `timescale 1ns / 1ps
 2 //////////////////////////////////////////////////////////////////////////////////
 3 // Company:
 4 // Engineer:
 5 //
 6 // Create Date:    16:32:33 05/03/06
 7 // Design Name:
 8 // Module Name:    VerticalLine
 9 // Project Name:
10 // Target Device:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module vertical_line(start_y, end_y, x, pixel, line, color);
22   input [9:0] start_y, end_y, x;
23   input [9:0] pixel, line;
24
25   output [23:0] color;
26
27   reg [23:0] color;
28
29   parameter WHITE = 24'hFFFFFF;
30   parameter BLACK = 24'h0;
31
32   always @(pixel or line or start_y or end_y or x) begin
33     if (line >= start_y && line <= end_y && pixel == x) begin
34       color = WHITE;
35     end else begin
36       color = BLACK;
37     end
38   end
39 endmodule
```

```
 1  `timescale 1ns / 1ps
 2  module vga (pixel_clock, reset, hsync, vsync, sync_b, blank_b,
    pixel_count, line_count);
 3    input pixel_clock; // 31.5 MHz pixel clock
 4    input reset; // system reset
 5    output hsync; // horizontal sync
 6    output vsync; // vertical sync
 7    output sync_b; // hardwired to Vdd
 8    output blank_b; // composite blank
 9    output [9:0] pixel_count; // number of the current pixel
10    output [9:0] line_count; // number of the current line
11    // 640x480 75Hz parameters
12    parameter PIXELS = 800;
13    parameter LINES = 525;
14    parameter HACTIVE_VIDEO = 640;
15    parameter HFRONT_PORCH = 16;
16    parameter HSYNC_PERIOD = 96;
17    parameter HBACK_PORCH = 48;
18    parameter VACTIVE_VIDEO = 480;
19    parameter VFRONT_PORCH = 11;
20    parameter VSYNC_PERIOD = 2;
21    parameter VBACK_PORCH = 32;
22    // current pixel count
23    reg [9:0] pixel_count = 10'b0;
24    reg [9:0] line_count = 10'b0;
25    // registered outputs
26    reg hsync = 1'b1;
27    reg vsync = 1'b1;
28    reg blank_b = 1'b1;
29    wire sync_b; // connected to Vdd
30    wire pixel_clock;
31    wire [9:0] next_pixel_count;
32    wire [9:0] next_line_count;
33    always @ (posedge pixel_clock) begin
34      if (reset) begin
35        pixel_count <= 10'b0;
36        line_count <= 10'b0;
37        hsync <= 1'b1;
38        vsync <= 1'b1;
39        blank_b <= 1'b1;
40      end else  begin
41        pixel_count <= next_pixel_count;
42        line_count <= next_line_count;
43        hsync <=
44          (next_pixel_count < HACTIVE_VIDEO + HFRONT_PORCH) |
45          (next_pixel_count >= HACTIVE_VIDEO+HFRONT_PORCH+
46          HSYNC_PERIOD);
47        vsync <=
48          (next_line_count < VACTIVE_VIDEO+VFRONT_PORCH) |
49          (next_line_count >= VACTIVE_VIDEO+VFRONT_PORCH+
50          VSYNC_PERIOD);
51        // this is the and of hblank and vblank
52        blank_b <=
53          (next_pixel_count < HACTIVE_VIDEO) &
54          (next_line_count < VACTIVE_VIDEO);
55      end
```

```
   end

   // next state is computed with combinational logic
   assign next_pixel_count = (pixel_count == PIXELS-1) ?
     10'h000 : pixel_count + 1'b1;
   assign next_line_count = (pixel_count == PIXELS-1) ?
     (line_count == LINES-1) ? 10'h000 :
     line_count + 1'b1 : line_count;
   // since we are providing hsync and vsync to the display, we
   // can hardwire composite sync to Vdd.
   assign sync_b = 1'b1;
endmodule
```

```
 1 `timescale 1ns / 1ps
 2
 3 module walsh(reset, clk, enable, num_fhts, optimal, symbols,
   decoded_bits);
 4   input clk, enable;
 5   input [2:0] num_fhts;
 6   input optimal;
 7   input [63:0] symbols;
 8
 9   output [5:0] decoded_bits;
10   reg [5:0] decoded_bits;
11
12   parameter WIDTH = 10;
13   wire [WIDTH-1:0] confidence;
14
15   reg upper_input;
16   reg lower_input;
17   wire ready;
18   fht_64_symbol optimal_decoder(reset, clk, upper_input, lower_input,
   decoded_bits, confidence, ready);
19
20   always @(posedge pixel_clock) begin
21     if (enable) begin
22       if
23     end
24   end
25 endmodule
```

```
 1 ////////////////////////////////////////////////////////////////////////
   ///////
 2 //
 3 // 6.111 FPGA Labkit -- Alphanumeric Display Interface for Lab 3 Memory
   Tester
 4 //
 5 //
 6 // Created: November 5, 2003
 7 // Author: Nathan Ickes
 8 //  Updated by Chris Buenrostro - 5/01/2006
 9 //  Updated by Mike Scharfstein - 1/30/2006
10 //
11 //    Notes:
12 //
13 //      *** Code written for Agilent Technologies HCMS-2973 Alphanumeric
   Displays
14 //
15 //      1)  manual_reset:
16 //        a)  when low, systems runs as initialized from power up
17 //        b)  when high, system reloads control words
18 //
19 //      2)  disp_test:
20 //        a)  when low, system loads dots (module input) into dot data
   register
21 //        b)  when high, system loads 640 1's into dot data register
22 //
23 //      3)  disp_blank:
24 //        a) always configured low for zero blanking
25 //
26 //      4)  disp_clock:
27 //        a)  set at 1/54th global_clock speed ( .5Mhz for 27Mhz input
   clock)
28 //
29 //      5)  control_reg_value (parameter):
30 //        a)  controls brightness of LED banks (four banks)
31 //          1)  values for desired brightness can be found in datasheet
32 //
33 ////////////////////////////////////////////////////////////////////////
   ///////
34
35 module alpha_display
36 (
37   global_clock, manual_reset, disp_test,
38   disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b,
   disp_data_out,
39   dots
40 );
41
42    input          global_clock;
43   input         manual_reset;
44    output         disp_blank;
45    output         disp_clock;
46    output         disp_data_out;    // serial data to displays
47    output         disp_rs;          // register select
48    output         disp_ce_b;        // chip enable
49    output         disp_reset_b;     // display reset
```

```
   50    input  [639:0]   dots;
   51   input         disp_test;        // check functionality of display
   52
   53    assign         disp_blank = 1'b0;  // 0=not blanked
   54
   55    reg          disp_clock;
   56   reg         disp_data_out;
   57   reg         disp_rs;
   58   reg         disp_ce_b;
   59   reg         disp_reset_b;
   60    // Internal signals
   61    reg    [5:0]   count;
   62    reg    [7:0]   state;
   63    reg    [9:0]   dot_index;
   64    reg    [639:0]   ldots;
   65   ////////////////////////////////
   66   //
   67   //  There are four Control Words
   68   //
   69   ////////////////////////////////
   70   //  parameter   control_reg_load_length = 8;  //first four Banks
   71   //  parameter   control_reg_load_length = 16; //second four Banks
   72   //  parameter   control_reg_load_length = 24; //third four Banks
   73   parameter   control_reg_load_length = 32;   //all four Banks
   74    reg    [control_reg_load_length - 1:0] control;
   75
   76   parameter RESET           = 0;
   77   parameter END_RESET       = 1;
   78   parameter INIT_DOT_REG      = 2;
   79   parameter LATCH_INIT_DOT_DATA = 3;
   80   parameter SELECT_CONTROL_REG  = 4;
   81   parameter SET_CONTROL_REG   = 5;
   82   parameter LATCH_CONTROL_REG   = 6;
   83   parameter SELECT_DOT_REG      = 7;
   84   parameter FILL_DOT_REG      = 8;
   85   parameter LATCH_DOT_DATA      = 9;
   86   parameter LOAD_NEW_DOT_DATA   = 10;
   87
   88   parameter control_reg_value   =  32'h7F7F7F7F;  // Controls LED
      brightness
   89
   90
      ////////////////////////////////////////////////////////////////////
      ////
   91    //
   92    // Initial Reset Generation
   93   //
   94   //    *** SRL16 is a variable-width shift register
   95    //
   96
      ////////////////////////////////////////////////////////////////////
      ////
   97
   98    wire reset;
   99
  100    SRL16 reset_sr (.D(1'b0), .CLK(global_clock), .Q(reset),
```

```
101         .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
102     defparam reset_sr.INIT = 16'hFFFF;
103
104
    ////////////////////////////////////////////////////////////////////
    ////
105     //
106     // Sequential Logic Block
107     //
108
    ////////////////////////////////////////////////////////////////////
    ////
109
110     always @(posedge global_clock)
111      begin
112         if (reset || manual_reset)
113         begin
114           count       <= 0;
115           disp_clock    <= 0;
116           disp_data_out   <= 0;
117           disp_rs       <= 0;
118           disp_ce_b     <= 1;
119           disp_reset_b  <= 0;
120           dot_index     <= 0;
121           state       <= 0;
122           control       <= control_reg_value;
123         end
124        else if (count==26)
125           begin
126             count       <= count+1;
127             disp_clock    <= 1;
128           end
129         else if (count==53)
130         begin
131             count       <= 0;
132             disp_clock    <= 0;
133
134
    ////////////////////////////////////////////////////////////////////
    ////
135      //
136      // Display State Machine
137      //
138
    ////////////////////////////////////////////////////////////////////
    ////
139
140         casex (state)
141
142           RESET:
143             begin
144               disp_data_out     <= 1'b0;
145               disp_rs         <= 1'b0; // dot register
146               disp_ce_b       <= 1'b1;
147               disp_reset_b    <= 1'b0;
148               dot_index       <= 0;
```

```
149               state         <= END_RESET;
150               end
151             END_RESET:
152               begin
153               disp_reset_b    <= 1'b1;
154               state         <= INIT_DOT_REG;
155               end
156             INIT_DOT_REG:
157               begin
158               disp_ce_b       <= 1'b0;
159               disp_data_out     <= 1'b0;
160
161               if (dot_index == 0)
162                 state       <= LATCH_INIT_DOT_DATA;
163               else
164                 dot_index     <= dot_index-1;
165             end
166            LATCH_INIT_DOT_DATA:
167               begin
168               disp_ce_b       <= 1'b1;
169               dot_index       <= control_reg_load_length-1;
170               state         <= SELECT_CONTROL_REG;
171               end
172           SELECT_CONTROL_REG:
173             begin
174               disp_rs         <= 1'b1;
175               state         <= SET_CONTROL_REG;
176             end
177             SET_CONTROL_REG:
178               begin
179               disp_ce_b <= 1'b0;
180               disp_data_out     <= control[dot_index];
181
182               if (dot_index == 0)
183                 state       <= LATCH_CONTROL_REG;
184               else
185                 dot_index     <= dot_index-1;
186               end
187             LATCH_CONTROL_REG:
188               begin
189               disp_ce_b       <= 1'b1;
190               dot_index       <= 639;
191               state         <= SELECT_DOT_REG;
192               end
193           SELECT_DOT_REG:
194             begin
195               disp_rs         <= 1'b0;
196               state         <= FILL_DOT_REG;
197             end
198             FILL_DOT_REG:
199               begin
200               disp_ce_b       <= 1'b0;
201
202               if (disp_test)
203                 disp_data_out   <= 1'b1;
204               else
```

```
  205                 disp_data_out   <= ldots[dot_index];
  206
  207               if (dot_index == 0)
  208                 state       <= LATCH_DOT_DATA;
  209               else
  210                 dot_index     <= dot_index-1;
  211             end
  212           LATCH_DOT_DATA:
  213              begin
  214              disp_ce_b       <= 1'b1;
  215              dot_index       <= 639;
  216              state         <= LOAD_NEW_DOT_DATA;
  217              end
  218           LOAD_NEW_DOT_DATA:
  219              begin
  220              ldots         <= dots;
  221              state         <= FILL_DOT_REG;
  222              end
  223            default:  state       <= state;
  224         endcase
  225       end
  226     else
  227          count <= count+1;
  228   end
  229 endmodule
```

```
    1 ////////////////////////////////////////////////////////////////////////
      ////////
    2 //
    3 // Lab 3 Memory Tester: Number to bitmap decoder
    4 //
    5 // This module converts a 4-bit input to a 80-dot (2 digit) bitmap
      representing
    6 // the numbers ' 0' through '15'.
    7 //
    8 // Author: Yun Wu, Nathan Ickes
    9 // Date: March 8, 2006
   10 //
   11 ////////////////////////////////////////////////////////////////////////
      ////////
   12
   13 module alpha_dots(clk, num, dots);
   14    input clk;
   15    input [3:0] num;
   16    output [79:0] dots;
   17
   18    reg [79:0] dots;
   19    always @ (posedge clk)
   20      case (num)
   21        4'd15: dots <= {40'b00000000_01000010_01111111_01000000_00000000,
      // '15'
   22                40'b00100111_01000101_01000101_01000101_00111001};
   23        4'd14: dots <= {40'b00000000_01000010_01111111_01000000_00000000,
      // '14'
   24                40'b00011000_00010100_00010010_01111111_00010000};
   25        4'd13: dots <= {40'b00000000_01000010_01111111_01000000_00000000,
      // '13'
   26                40'b00100010_01000001_01001001_01001001_00110110};
   27        4'd12: dots <= {40'b00000000_01000010_01111111_01000000_00000000,
      // '12'
   28                40'b01100010_01010001_01001001_01001001_01000110};
   29        4'd11: dots <= {40'b00000000_01000010_01111111_01000000_00000000,
      // '11'
   30                40'b00000000_01000010_01111111_01000000_00000000};
   31        4'd10: dots <= {40'b00000000_01000010_01111111_01000000_00000000,
      // '10'
   32                40'b00111110_01010001_01001001_01000101_00111110};
   33        4'd09: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
      // ' 9'
   34                40'b00000110_01001001_01001001_00101001_00011110};
   35        4'd08: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
      // ' 8'
   36                40'b00110110_01001001_01001001_01001001_00110110};
   37        4'd07: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
      // ' 7'
   38                40'b00000001_01110001_00001001_00000101_00000011};
   39        4'd06: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
      // ' 6'
   40                40'b00111100_01001010_01001001_01001001_00110000};
   41        4'd05: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
      // ' 5'
   42                40'b00100111_01000101_01000101_01000101_00111001};
```

```
43         4'd04: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
   // ' 4'
44             40'b00011000_00010100_00010010_01111111_00010000};
45         4'd03: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
   // ' 3'
46             40'b00100010_01000001_01001001_01001001_00110110};
47         4'd02: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
   // ' 2'
48             40'b01100010_01010001_01001001_01001001_01000110};
49         4'd01: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
   // ' 1'
50             40'b00000000_01000010_01111111_01000000_00000000};
51         4'd00: dots <= {40'b00000000_00000000_00000000_00000000_00000000,
   // ' 0'
52             40'b00111110_01010001_01001001_01000101_00111110};
53         // No default case, becase every case is already accounted for.
54       endcase
55
56 endmodule
```

```
 1  `timescale 1ns / 1ps
 2
 3  module ber(first_input, second_input, error_count);
 4    // Combinational, parallel detection of mismatched bits
 5    input [5:0] first_input, second_input;
 6
 7    output [2:0] error_count;
 8    wire [5:0] a, b;
 9
10    parameter WIDTH = 10;
11
12    assign a = first_input;
13    assign b = second_input;
14    assign error_count = (a[0] ^ b[0]) + (a[1] ^ b[1]) + (a[2] ^ b[2]) +
    (a[3] ^ b[3]) +
15                  (a[4] ^ b[4]) + (a[5] ^ b[5]); // + (a[6] ^ b[6]) +
    (a[7] ^ b[7]) +
16                  /*(a[8] ^ b[8]) + (a[9] ^ b[9]) + (a[10] ^ b[10]) +
    (a[11] ^ b[12]) +
17                  (a[12] ^ b[12]) + (a[13] ^ b[13]) + (a[14] ^ b[14]) +
    (a[15] ^ b[15]) +
18                  (a[16] ^ b[16]) + (a[17] ^ b[17]) + (a[18] ^ b[18]) +
    (a[19] ^ b[19]) +
19                  (a[20] ^ b[20]) + (a[21] ^ b[21]) + (a[22] ^ b[22]) +
    (a[23] ^ b[23]) +
20                  (a[24] ^ b[24]) + (a[25] ^ b[25]) + (a[26] ^ b[26]) +
    (a[27] ^ b[27]) +
21                  (a[28] ^ b[28]) + (a[29] ^ b[29]) + (a[30] ^ b[30]) +
    (a[31] ^ b[31]) +
22                  (a[32] ^ b[32]) + (a[33] ^ b[33]) + (a[34] ^ b[34]) +
    (a[35] ^ b[35]) +
23                  (a[36] ^ b[36]) + (a[37] ^ b[37]) + (a[38] ^ b[38]) +
    (a[39] ^ b[39]) +
24                  (a[40] ^ b[40]) + (a[41] ^ b[41]) + (a[42] ^ b[42]) +
    (a[43] ^ b[43]) +
25                  (a[44] ^ b[44]) + (a[45] ^ b[45]) + (a[46] ^ b[46]) +
    (a[47] ^ b[47]) +
26                  (a[48] ^ b[48]) + (a[49] ^ b[49]) + (a[50] ^ b[50]) +
    (a[51] ^ b[51]) +
27                  (a[52] ^ b[52]) + (a[53] ^ b[53]) + (a[54] ^ b[54]) +
    (a[55] ^ b[55]) +
28                  (a[56] ^ b[56]) + (a[57] ^ b[57]) + (a[58] ^ b[58]) +
    (a[59] ^ b[59]);
29                  */
30  endmodule
```

```
 1  `timescale 1ns / 1ps
 2  //////////////////////////////////////////////////////////////////////////////////
 3  // Company:
 4  // Engineer:
 5  //
 6  // Create Date:    22:41:05 05/10/06
 7  // Design Name:
 8  // Module Name:    control
 9  // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module control(clk, reset, up, down, addr);
22    input clk, reset, up, down;
23    output [5:0] addr;
24
25
26    reg [5:0] ctr;
27    wire [5:0] addr;
28    assign addr = ctr;
29
30    always @(posedge clk) begin
31      if (reset) begin
32        ctr <= 0;
33      end else begin
34        ctr <= ctr + 1;
35      end
36    end
37  endmodule
```

```
 1 //
 2 // File:   cstringdisp.v
 3 // Date:   24-Oct-05
 4 // Author: I. Chuang, C. Terman
 5 //
 6 // Display an ASCII encoded character string in a video window at some
 7 // specified x,y pixel location.
 8 //
 9 // INPUTS:
10 //
11 //   vclock       - video pixel clock
12 //   hcount       - horizontal (x) location of current pixel
13 //   vcount       - vertical (y) location of current pixel
14 //   cstring      - character string to display (8 bit ASCII for each
   char)
15 //   cx,cy        - pixel location (upper left corner) to display string
   at
16 //
17 // OUTPUT:
18 //
19 //   pixel        - video pixel value to display at current location
20 //
21 // PARAMETERS:
22 //
23 //   NCHAR        - number of characters in string to display
24 //   NCHAR_BITS   - number of bits to specify NCHAR
25 //
26 // pixel should be OR'ed (or XOR'ed) to your video data for display.
27 //
28 // Each character is 8x12, but pixels are doubled horizontally and
   vertically
29 // so fonts are magnified 2x.  On an XGA screen (1024x768) you can fit
30 // 64 x 32 such characters.
31 //
32 // Needs font_rom.v and font_rom.ngo
33 //
34 // For different fonts, you can change font_rom.  For different string
35 // display colors, change the assignment to cpixel.
36
37
38 ////////////////////////////////////////////////////////////////////////
   //////
39 //
40 // video character string display
41 //
42 ////////////////////////////////////////////////////////////////////////
   //////
43
44 module char_string_display (vclock,hcount,vcount,pixel,cstring,cx,cy);
45
46    parameter NCHAR = 11;  // number of 8-bit characters in cstring
47    parameter NCHAR_BITS = 5; // number of bits in NCHAR
48   parameter SCALE = 1;
49
50    input vclock;  // 65MHz clock
51    input [10:0] hcount; // horizontal index of current pixel (0..1023)
```

```
   52    input [9:0]  vcount; // vertical index of current pixel (0..767)
   53    output [2:0] pixel;  // char display's pixel
   54    input [NCHAR*8-1:0] cstring; // character string to display
   55    input [10:0] cx;
   56    input [9:0]  cy;
   57
   58    // 1 line x 8 character display (8 x 12 pixel-sized characters)
   59
   60    wire [10:0]  hoff = (hcount-1-cx) * SCALE;
   61    wire [9:0]   voff = (vcount-cy)*SCALE;
   62    wire [NCHAR_BITS-1:0] column = NCHAR-1-hoff[NCHAR_BITS-1+4:4];  // <
      NCHAR
   63    wire [2:0]   h = hoff[3:1];            // 0 .. 7
   64    wire [3:0]   v = voff[4:1];      // 0 .. 11
   65
   66    // look up character to display (from character string)
   67    reg [7:0]  char;
   68    integer  n;
   69    always @( * )
   70      for (n=0 ; n<8 ; n = n+1 )   // 8 bits per character (ASCII)
   71        char[n] <= cstring[column*8+n];
   72
   73    // look up raster row from font rom
   74    wire reverse = char[7];
   75    wire [10:0] font_addr = char[6:0]*12 + v;    // 12 bytes per
      character
   76    wire [7:0]  font_byte;
   77    font_rom f(font_addr,vclock,font_byte);
   78
   79    // generate character pixel if we're in the right h,v area
   80    wire [2:0] cpixel = (font_byte[7 - h] ^ reverse) ? 7 : 0;
   81    wire dispflag = ((hcount > cx) & (vcount >= cy) & (hcount <=
      cx+NCHAR*((SCALE == 1) ? 16 : 8))
   82         & (vcount < cy + ((SCALE == 1) ? 24 : 12)));
   83    wire [2:0] pixel = dispflag ? cpixel : 0;
   84
   85 endmodule
```

```
 1 // Switch Debounce Module
 2 // use your system clock for the clock input
 3 // to produce a synchronous, debounced output
 4 module debounce (reset, clock, noisy, clean);
 5    parameter DELAY = 270000;   // .01 sec with a 27Mhz clock
 6    input reset, clock, noisy;
 7    output clean;
 8
 9    reg [18:0] count;
10    reg new, clean;
11
12    always @(posedge clock)
13      if (reset)
14        begin
15     count <= 0;
16     new <= noisy;
17     clean <= noisy;
18        end
19      else if (noisy != new)
20        begin
21     new <= noisy;
22     count <= 0;
23        end
24      else if (count == DELAY)
25        clean <= new;
26      else
27        count <= count+1;
28
29 endmodule
```

```
 1  `timescale 1ns / 1ps
 2
 3  module display(got_new_data, clock, reset, enable, pixel, line, color,
    current_error_count, power_usage, new_data_count, error_count);
 4    input clock, enable, reset, got_new_data;
 5    input [9:0] pixel, line;
 6    input [2:0] current_error_count;
 7    input [2:0] power_usage;
 8    output [7:0] new_data_count, error_count;
 9    output [23:0] color;
10
11
12    parameter WIDTH = 640;
13    parameter HEIGHT = 480;
14    parameter MIT_RED = 24'b0101_1111_0001_1111_0001_1111;
15    parameter WHITE = 24'hFFFFFF;
16    parameter BLACK = 24'h000000;
17
18    parameter DOTS_WIDTH = 255;
19    parameter LEFT_X = 10'd192;
20    parameter RIGHT_X = LEFT_X + DOTS_WIDTH;
21    parameter TOP_Y = 10'd138;
22    // Oversquare sized
23    parameter BOTTOM_Y = TOP_Y + 80;
24
25    parameter BAT_L_X = RIGHT_X - 50;
26    parameter BAT_R_X = BAT_L_X + 50;
27    parameter BAT_T_Y = BOTTOM_Y + 28;
28    parameter BAT_B_Y = BAT_T_Y + 80;
29    parameter BAT_MIDPT_X = BAT_L_X + 25;
30
31    integer i,j;
32
33    wire [23:0] color;
34    wire [23:0] c1;
35    wire [23:0] c2;
36    wire [23:0] c3a;
37    wire [23:0] c3b;
38    wire [23:0] c3c;
39    wire [23:0] c4;
40    wire [23:0] c5;
41    wire [23:0] c6;
42    wire [23:0] c7;
43    wire [23:0] c8;
44    wire [23:0] c9;
45    wire [23:0] c10;
46    wire [23:0] c11;
47    wire [23:0] c12;
48
49    wire [2:0] p1;
50    assign c9 = {p1, p1, p1, p1, p1, p1, p1, p1};
51    wire [2:0] p2;
52    assign c10 = {16'b0, p2[1:0], p2, p2};
53    wire [2:0] p3;
54    assign c11 = {8'b0, p3[1:0], p3, p3, 8'b0};
55    wire [2:0] p4;
```

```
56    assign c12 = {p4[1:0], p4, p4, 16'b0};
57
58
59    reg [7:0] error_count;
60
61    char_string_display title_csd(clock, pixel, line, p1, {8'd71, 8'd101,
    8'd110, 8'd101, 8'd114, 8'd97, 8'd108, 8'd105, 8'd122, 8'd101, 8'd100,
    8'd32, 8'd76, 8'd111, 8'd99, 8'd97, 8'd108, 8'd32, 8'd68, 8'd101, 8'd99,
    8'd111, 8'd100, 8'd105, 8'd110, 8'd103}, 100, 50);
62     defparam title_csd.NCHAR = 26;
63
64    char_string_display ber_csd(clock, pixel, line, p2, {8'd66, 8'd69,
    8'd82}, LEFT_X, BOTTOM_Y + 10);
65     defparam ber_csd.NCHAR = 3;
66    defparam ber_csd.SCALE = 2;
67
68    char_string_display snr_csd(clock, pixel, line, p3, {8'd83, 8'd78,
    8'd82}, LEFT_X + 100, BOTTOM_Y + 10);
69     defparam snr_csd.NCHAR = 3;
70    defparam snr_csd.SCALE = 2;
71
72    char_string_display power_csd(clock, pixel, line, p4, {8'd80, 8'd111,
    8'd119, 8'd101, 8'd114}, LEFT_X + 200, BOTTOM_Y + 10);
73     defparam power_csd.NCHAR = 5;
74    defparam power_csd.SCALE = 2;
75
76
77    horizontal_line time_axis(.start_x(LEFT_X), .end_x(RIGHT_X),
    .y(BOTTOM_Y), .pixel(pixel), .line(line), .color(c1));
78    vertical_line data_axis(.x(LEFT_X), .start_y(TOP_Y), .end_y(BOTTOM_Y),
    .pixel(pixel), .line(line), .color(c2));
79    dots ber_dots(.clock(clock), .enable(enable),
    .current_word(error_count), .left_x(LEFT_X), .bottom_y(BOTTOM_Y),
    .pixel(pixel), .line(line), .color(c3a));
80    dots snr_dots(.clock(clock), .enable(enable), .current_word(8'd50),
    .left_x(LEFT_X), .bottom_y(BOTTOM_Y), .pixel(pixel), .line(line),
    .color(c3b));
81    defparam snr_dots.DOT_COLOR = 24'h00FF00;
82    dots power_dots(.clock(clock), .enable(enable), .current_word(8'd40),
    .left_x(LEFT_X), .bottom_y(BOTTOM_Y), .pixel(pixel), .line(line),
    .color(c3c));
83    defparam power_dots.DOT_COLOR = 24'hFF0000;
84
85    horizontal_line battery_bottom(.start_x(BAT_L_X), .end_x(BAT_R_X),
    .y(BAT_B_Y), .pixel(pixel), .line(line), .color(c4));
86    vertical_line battery_left(.x(BAT_L_X), .start_y(BAT_T_Y),
    .end_y(BAT_B_Y), .pixel(pixel), .line(line), .color(c5));
87    vertical_line battery_right(.x(BAT_R_X), .start_y(BAT_T_Y),
    .end_y(BAT_B_Y), .pixel(pixel), .line(line), .color(c6));
88    horizontal_line battery_top(.start_x(BAT_L_X), .end_x(BAT_R_X),
    .y(BAT_T_Y), .pixel(pixel), .line(line), .color(c7));
89    gauge power_usage_gauge(.start_x(BAT_L_X), .end_x(BAT_R_X),
    .y(BAT_B_Y), .value(power_usage), .pixel(pixel), .line(line),
    .color(c8));
90
91    assign color = c1 | c2 | c3a | c3b | c3c | c4 | c5 | c6 | c7 | c8 | c9
```

```
 91 | c10 | c11 | c12;
 92   reg [5:0] ticker;
 93   reg [7:0] new_data_count;
 94   always @(posedge clock) begin
 95     if (reset || enable) begin
 96       error_count <= 8'b0;
 97       new_data_count <= 8'b0;
 98     end else if (got_new_data && new_data_count < 5) begin
 99       new_data_count <= new_data_count + 1;
100       error_count <= error_count + current_error_count;
101     end
102     if (reset) begin
103       ticker <= 0;
104     end else begin
105       if (enable) begin
106         ticker <= ticker + 10;
107       end
108     end
109   end
110 endmodule
111
```

```
 1  `timescale 1ns / 1ps
 2
 3  module display_test_v;
 4
 5    // Inputs
 6    reg got_new_data;
 7    reg clock;
 8    reg reset;
 9    reg enable;
10    reg [9:0] pixel;
11    reg [9:0] line;
12    reg [2:0] current_error_count;
13
14    // Outputs
15    wire [23:0] color;
16    wire [5:0] error_count;
17
18    reg up, down;
19    wire [5:0] addr;
20    control control_uut(
21      .clk(clock),
22      .reset(reset),
23      .up(up),
24      .down(down),
25      .addr(addr)
26    );
27    // Instantiate the Unit Under Test (UUT)
28    display uut (
29      .got_new_data(got_new_data),
30      .clock(clock),
31      .reset(reset),
32      .enable(enable),
33      .pixel(pixel),
34      .line(line),
35      .color(color),
36      .current_error_count(current_error_count),
37      .error_count(error_count)
38    );
39
40
41    reg [4:0] ctr;
42    always #10 clock=~clock;
43    always #10 ctr=ctr+1;
44    integer i; integer j;
45    initial begin
46      // Initialize Inputs
47      got_new_data = 0;
48      clock = 0;
49      ctr = 0;
50      reset = 0;
51      enable = 0;
52      pixel = 0;
53      line = 0;
54      current_error_count = 0;
55
56      // Wait 100 ns for global reset to finish
```

```
57      #100;
58      for (j = 0; j < 100; j = j + 1) begin
59          for (i = 0; i < 10; i = i + 1) begin
60          current_error_count = ctr[2:0];
61          got_new_data = 1;
62        end
63        //got_new_data = 0;
64        enable = 1;
65        #20;
66        enable = 0;
67      end
68      // Add stimulus here
69
70    end
71
72  endmodule
73
```

```
 1  `timescale 1ns / 1ps
 2
 3  module divider(clock, enable, reset);
 4     input reset, clock;
 5
 6     output enable;
 7
 8    reg enable;
 9     reg [25:0] cnt;
10
11     always @(posedge clock) begin
12      if (reset == 1) begin
13        cnt <= 26'd0;
14        enable <= 1'b0;
15      end else if (cnt == 26'd6299999) begin
16        cnt <= 26'd0;
17        enable <= 1'b1;
18      end else begin
19        cnt <= cnt + 1;
20        enable <= 1'b0;
21      end
22     end
23  endmodule
```

```
 1 `timescale 1ns / 1ps
 2
 3 module dots(clock, enable, current_word, left_x, bottom_y, pixel, line,
   color);
 4   input clock, enable;
 5   input [9:0] left_x, bottom_y;
 6   input [9:0] pixel, line;
 7   input [7:0] current_word;
 8
 9   output [23:0] color;
10
11   parameter DOT_COLOR = 24'h0000FF;
12   parameter BLACK = 24'h0;
13   parameter DOTS_WIDTH = 255;
14
15   reg [23:0] color;
16   reg [7:0] dot_data [DOTS_WIDTH:0]; // DOTS_WIDTH 8-bit words
17   integer i;
18
19   integer j;
20   reg [7:0] ctr;
21
22   always @(posedge clock) begin
23     if (enable) begin
24       // Shift dots backward
25       for (i = 0; i < DOTS_WIDTH; i = i + 1) begin
26         dot_data[i] <= dot_data[i + 1];
27       end
28
29       dot_data[DOTS_WIDTH] <= current_word;
30     end
31   end
32
33   always @(current_word or clock or enable or pixel or line or left_x or
   bottom_y) begin
34     if (pixel > left_x && line < bottom_y && pixel <= left_x + 256 &&
   line >= bottom_y - 64
35         && line == bottom_y - dot_data[pixel - left_x] - 1) begin
36       color = DOT_COLOR;
37     end else begin
38       color = BLACK;
39     end
40   end
41
42 endmodule
```

```
 1 `timescale 1ns / 1ps
 2
 3 module feedback_path(enable, reset, clk, enabled, decrement, increment,
   error_count, num_fhts, saw_button);
 4   input [2:0] error_count;
 5   input enable, reset, clk, enabled, decrement, increment;
 6   output [2:0] num_fhts;
 7   output saw_button;
 8
 9   reg [2:0] num_fhts;
10
11   parameter TARGET_BER = 8'd10;
12
13   reg saw_button;
14   always @(posedge clk) begin
15     if (reset) begin
16       num_fhts <= 3'd7;
17       saw_button <= 0;
18     end else begin
19       if (enabled) begin
20         if (enable) begin
21           if (error_count > TARGET_BER) begin
22             if (num_fhts < 7) begin
23               num_fhts <= num_fhts + 1;
24             end
25           end else begin
26             if (num_fhts > 0) begin
27               num_fhts <= num_fhts - 1;
28             end
29           end
30         end
31       end else if (decrement) begin
32         if (!saw_button) begin
33           saw_button <= 1;
34           if (num_fhts > 0) begin
35             num_fhts <= num_fhts - 1;
36           end
37         end
38       end else if (increment) begin
39         if (!saw_button) begin
40           saw_button <= 1;
41           if (num_fhts < 7) begin
42             num_fhts <= num_fhts + 1;
43           end
44         end
45       end else begin
46         saw_button <= 0;
47       end
48     end
49   end
50 endmodule
```

```
 1  function y = fht(x)
 2  % Returns fast Hadamard transform of columns of x
 3
 4  L = length(x(:,1));
 5  n = log2(L);
 6  y = x;
 7  for i=1:n
 8    tmp(1:L/2,:) = y(1:2:L,:) + y(2:2:L,:);
 9    tmp(L/2+1:L,:) = y(1:2:L,:) - y(2:2:L,:);
10    y = tmp;
11  end
```

```
 1  `timescale 1ns / 1ps
 2  module fht_8_symbol(reset, clk, upper_input, lower_input, upper_corr,
    lower_corr, ready);
 3    input reset, clk;
 4    parameter WIDTH = 10;
 5    input signed [WIDTH-1:0] upper_input, lower_input;
 6    output ready;
 7    output [WIDTH-1:0] upper_corr, lower_corr;
 8
 9    reg [4:0] ctr;
10
11    // Combinational logic
12    //
13    // NORMAL STAGES
14    wire signed [WIDTH-1:0] s5_upper_out, s5_lower_out;
15    wire signed [WIDTH-1:0] s6_upper_out, s6_lower_out, s7_upper_out,
    s7_lower_out;
16    wire signed [WIDTH-1:0] upper_fly_out, lower_fly_out;
17
18    wire twoenable, oneenable;
19
20    fht_stage s6(reset, clk, ctr[1], upper_input, lower_input,
    s6_upper_out, s6_lower_out);
21    defparam s6.SIZE = 2;
22    fht_stage s7(reset, clk, ctr[0], s6_upper_out, s6_lower_out,
    s7_upper_out, s7_lower_out);
23    defparam s7.SIZE = 1;
24
25    // FINAL STAGE
26    // Abs() takes the place of squaring the correlation values
27    assign upper_corr = (s7_upper_out + s7_lower_out > 0) ? (s7_upper_out
    + s7_lower_out)
28                                              : -(s7_upper_out +
    s7_lower_out);
29    assign lower_corr = (s7_upper_out - s7_lower_out > 0) ? (s7_upper_out
    - s7_lower_out)
30                                              : -(s7_upper_out -
    s7_lower_out);
31    // Clocked logic
32    reg [5:0] largest_component_idx;
33    reg signed [WIDTH-1:0] largest_component_val;
34    reg ready;
35
36    always @(posedge clk) begin
37      if (reset) begin
38      ctr <= 0;
39      ready <= 1'b0;
40     end else begin
41        // Ready signal indicates we're done
42        ready <= (ctr == 5'd31);
43        ctr <= ctr + 1;
44     end
45    end
46  endmodule
47
```

```
 1  `timescale 1ns / 1ps
 2  // 64-symbol FHT. From Bahl's paper. During cycles 0-31, takes inputs
    0-31 on the upper input and 32-63 on the lower input.
 3  // During cycles 31-62, outputs 64 correlation coefficients (first on
    upper, second lower, third upper second cycle, etc.)
 4  module fht_64_symbol(reset, clk, upper_input, lower_input,
    largest_component_idx, largest_component_val, ready, upper_fly_out,
    lower_fly_out, ctr);
 5    input reset, clk;
 6    parameter WIDTH = 10;
 7    input signed [WIDTH-1:0] upper_input, lower_input;
 8    output ready;
 9    output [5:0] largest_component_idx;
10    output [WIDTH-1:0] largest_component_val;
11    output [WIDTH-1:0] upper_fly_out, lower_fly_out;
12    output [5:0] ctr;
13    // 7 bit counter
14    reg [5:0] ctr;
15
16    // First few stages
17    wire signed [WIDTH-1:0] s3_upper_out, s3_lower_out;
18    wire signed [WIDTH-1:0] s4_upper_out, s4_lower_out, s5_upper_out,
    s5_lower_out;
19    wire signed [WIDTH-1:0] s6_upper_out, s6_lower_out, s7_upper_out,
    s7_lower_out;
20    wire signed [WIDTH-1:0] upper_fly_out, lower_fly_out;
21
22    fht_stage s3(reset, clk, ctr[4], upper_input, lower_input,
    s3_upper_out, s3_lower_out);
23    defparam s3.SIZE = 16;
24    fht_stage s4(reset, clk, ctr[3], s3_upper_out, s3_lower_out,
    s4_upper_out, s4_lower_out);
25    defparam s4.SIZE = 8;
26    fht_stage s5(reset, clk, ctr[2], s4_upper_out, s4_lower_out,
    s5_upper_out, s5_lower_out);
27    defparam s5.SIZE = 4;
28    fht_stage s6(reset, clk, ctr[1], s5_upper_out, s5_lower_out,
    s6_upper_out, s6_lower_out);
29    defparam s6.SIZE = 2;
30    fht_stage s7(reset, clk, ctr[0], s6_upper_out, s6_lower_out,
    s7_upper_out, s7_lower_out);
31    defparam s7.SIZE = 1;
32
33    // Last stage
34    // abs takes the place of squaring
35    assign upper_fly_out = (s7_upper_out + s7_lower_out > 0) ?
    (s7_upper_out + s7_lower_out)
36                                          : -(s7_upper_out +
    s7_lower_out);
37    assign lower_fly_out = (s7_upper_out - s7_lower_out > 0) ?
    (s7_upper_out - s7_lower_out)
38                                          : -(s7_upper_out -
    s7_lower_out);
39
40    reg [5:0] largest_component_idx;
41    reg signed [WIDTH-1:0] largest_component_val;
```

```
42   reg ready, ready_thrown;
43
44   always @(posedge clk) begin
45     if (reset) begin
46     ctr <= 0;
47     largest_component_idx <= 10'b0;
48     largest_component_val <= 10'b0;
49     ready <= 1'b0;
50     ready_thrown <= 0;
51    end else begin
52     // Comparator
53     largest_component_idx <= reset || ready ? 10'b0 :
54                          (upper_fly_out > largest_component_val &&
   upper_fly_out > lower_fly_out) ? (ctr - 31) * 2 :
55                          lower_fly_out > largest_component_val ? (ctr -
   31) * 2 + 1 :
56                        largest_component_idx;
57      largest_component_val <= reset || ready ? 10'b0 :
58                           (upper_fly_out > largest_component_val &&
   upper_fly_out > lower_fly_out) ? upper_fly_out :
59                          lower_fly_out > largest_component_val ?
   lower_fly_out :
60                        largest_component_val;
61       if (ctr == 6'd62 && !ready_thrown) begin
62       ready <= 1;
63       ready_thrown <= 1;
64     end
65       ctr <= ready ? 6'd0 : ctr + 1;
66    end
67   end
68 endmodule
69
```

```
 1  `timescale 1ns / 1ps
 2  // Single FHT stage, from Bahl's paper.
 3  module fht_stage(reset, clk, enable, upper_input, lower_input,
    upper_output, lower_output);
 4    input reset, clk, enable;
 5
 6    parameter WIDTH = 10;
 7    input signed [WIDTH-1:0] upper_input, lower_input;
 8
 9    output signed [WIDTH-1:0] upper_output, lower_output;
10    parameter SIZE = 64;
11
12    wire signed [WIDTH-1:0] upper_fly_out, lower_fly_out, first_mux_out,
    upper_sr_in, lower_sr_in;
13    wire signed [WIDTH-1:0] upper_sr_out, lower_sr_out,
    upperright_mux_out, lowerright_mux_out;
14    wire signed [WIDTH-1:0] upper_output, lower_output;
15
16    assign upper_fly_out = upper_input + lower_input;
17    assign lower_fly_out = upper_input - lower_input;
18    assign first_mux_out = enable ? lower_fly_out : upper_fly_out;
19
20    // always shift top sr forward
21    assign upper_sr_in = first_mux_out;
22    shift_register upper_sr(reset, clk, 1'b1, upper_sr_in, upper_sr_out);
23    defparam upper_sr.SIZE = SIZE;
24    defparam upper_sr.WIDTH = WIDTH;
25
26    // on !enable, shift bottom register forward
27    assign lower_sr_in = lower_fly_out;
28    shift_register lower_sr(reset, clk, !enable, lower_sr_in,
    lower_sr_out);
29    defparam lower_sr.SIZE = SIZE;
30    defparam lower_sr.WIDTH = WIDTH;
31
32    assign upperright_mux_out = !enable ? lower_sr_out : upper_sr_out;
33    assign lowerright_mux_out = !enable ? upper_sr_out : upper_fly_out;
34
35    assign upper_output = upperright_mux_out;
36    assign lower_output = lowerright_mux_out;
37 endmodule
```

```
    1 ////////////////////////////////////////////////////////////////////////
      ///////
    2 //
    3 // 6.111 FPGA Labkit -- Template Toplevel Module for Lab 4 (Spring 2006)
    4 //
    5 //
    6 // Created: March 13, 2006
    7 // Author: Nathan Ickes
    8 //
    9 ////////////////////////////////////////////////////////////////////////
      ///////
   10
   11 module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in,
      ac97_synch,
   12          ac97_bit_clock,
   13
   14          vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
   15          vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
   16          vga_out_vsync,
   17
   18          tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
   19          tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
   20          tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
   21
   22          tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
   23          tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
   24          tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
   25          tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
   26
   27          ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
   28          ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
   29
   30          ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
   31          ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
   32
   33          clock_feedback_out, clock_feedback_in,
   34
   35          flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
   36          flash_reset_b, flash_sts, flash_byte_b,
   37
   38          rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
   39
   40          mouse_clock, mouse_data, keyboard_clock, keyboard_data,
   41
   42          clock_27mhz, clock1, clock2,
   43
   44          disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
   45          disp_reset_b, disp_data_in,
   46
   47          button0, button1, button2, button3, button_enter, button_right,
   48          button_left, button_down, button_up,
   49
   50          switch,
   51
   52          led,
   53
```

```
   54          user1, user2, user3, user4,
   55
   56          daughtercard,
   57
   58          systemace_data, systemace_address, systemace_ce_b,
   59          systemace_we_b, systemace_oe_b, systemace_irq,
    systemace_mpbrdy,
   60
   61          analyzer1_data, analyzer1_clock,
   62          analyzer2_data, analyzer2_clock,
   63          analyzer3_data, analyzer3_clock,
   64          analyzer4_data, analyzer4_clock);
   65
   66    output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
   67    input  ac97_bit_clock, ac97_sdata_in;
   68
   69    output [7:0] vga_out_red, vga_out_green, vga_out_blue;
   70    output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
   71     vga_out_hsync, vga_out_vsync;
   72
   73    output [9:0] tv_out_ycrcb;
   74    output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
    tv_out_i2c_data,
   75     tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
   76     tv_out_subcar_reset;
   77
   78    input  [19:0] tv_in_ycrcb;
   79    input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
    tv_in_aef,
   80     tv_in_hff, tv_in_aff;
   81    output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
   82     tv_in_reset_b, tv_in_clock;
   83    inout  tv_in_i2c_data;
   84
   85    inout  [35:0] ram0_data;
   86    output [18:0] ram0_address;
   87    output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b,
    ram0_we_b;
   88    output [3:0] ram0_bwe_b;
   89
   90    inout  [35:0] ram1_data;
   91    output [18:0] ram1_address;
   92    output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b,
    ram1_we_b;
   93    output [3:0] ram1_bwe_b;
   94
   95    input  clock_feedback_in;
   96    output clock_feedback_out;
   97
   98    inout  [15:0] flash_data;
   99    output [24:0] flash_address;
  100    output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b,
    flash_byte_b;
  101    input  flash_sts;
  102
  103    output rs232_txd, rs232_rts;
```

```
  104    input  rs232_rxd, rs232_cts;
  105
  106    input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
  107
  108    input  clock_27mhz, clock1, clock2;
  109
  110    output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
  111    input  disp_data_in;
  112    output  disp_data_out;
  113
  114    input  button0, button1, button2, button3, button_enter,
      button_right,
  115     button_left, button_down, button_up;
  116    input  [7:0] switch;
  117    output [7:0] led;
  118
  119    inout [31:0] user1, user2, user3, user4;
  120
  121    inout [43:0] daughtercard;
  122
  123    inout  [15:0] systemace_data;
  124    output [6:0]  systemace_address;
  125    output systemace_ce_b, systemace_we_b, systemace_oe_b;
  126    input  systemace_irq, systemace_mpbrdy;
  127
  128    output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
  129      analyzer4_data;
  130    output analyzer1_clock, analyzer2_clock, analyzer3_clock,
      analyzer4_clock;
  131
  132
      ////////////////////////////////////////////////////////////////////////
      ////
  133    //
  134    // I/O Assignments
  135    //
  136
      ////////////////////////////////////////////////////////////////////////
      ////
  137
  138    // Audio Input and Output
  139    assign beep= 1'b0;
  140    assign audio_reset_b = 1'b0;
  141    assign ac97_synch = 1'b0;
  142    assign ac97_sdata_out = 1'b0;
  143
  144    // Video Output
  145    assign tv_out_ycrcb = 10'h0;
  146    assign tv_out_reset_b = 1'b0;
  147    assign tv_out_clock = 1'b0;
  148    assign tv_out_i2c_clock = 1'b0;
  149    assign tv_out_i2c_data = 1'b0;
  150    assign tv_out_pal_ntsc = 1'b0;
  151    assign tv_out_hsync_b = 1'b1;
  152    assign tv_out_vsync_b = 1'b1;
  153    assign tv_out_blank_b = 1'b1;
```

```
154     assign tv_out_subcar_reset = 1'b0;
155
156     // Video Input
157     assign tv_in_i2c_clock = 1'b0;
158     assign tv_in_fifo_read = 1'b0;
159     assign tv_in_fifo_clock = 1'b0;
160     assign tv_in_iso = 1'b0;
161     assign tv_in_reset_b = 1'b0;
162     assign tv_in_clock = 1'b0;
163     assign tv_in_i2c_data = 1'bZ;
164
165     // SRAMs
166     assign ram0_data = 36'hZ;
167     assign ram0_address = 19'h0;
168     assign ram0_adv_ld = 1'b0;
169     assign ram0_clk = 1'b0;
170     assign ram0_cen_b = 1'b1;
171     assign ram0_ce_b = 1'b1;
172     assign ram0_oe_b = 1'b1;
173     assign ram0_we_b = 1'b1;
174     assign ram0_bwe_b = 4'hF;
175     assign ram1_data = 36'hZ;
176     assign ram1_address = 19'h0;
177     assign ram1_adv_ld = 1'b0;
178     assign ram1_clk = 1'b0;
179     assign ram1_cen_b = 1'b1;
180     assign ram1_ce_b = 1'b1;
181     assign ram1_oe_b = 1'b1;
182     assign ram1_we_b = 1'b1;
183     assign ram1_bwe_b = 4'hF;
184     assign clock_feedback_out = 1'b0;
185
186     // Flash ROM
187     assign flash_data = 16'hZ;
188     assign flash_address = 24'h0;
189     assign flash_ce_b = 1'b1;
190     assign flash_oe_b = 1'b1;
191     assign flash_we_b = 1'b1;
192     assign flash_reset_b = 1'b0;
193     assign flash_byte_b = 1'b1;
194
195     // RS-232 Interface
196     assign rs232_txd = 1'b1;
197     assign rs232_rts = 1'b1;
198
199     // LED Displays
200     //assign disp_blank = 1'b1;
201     //assign disp_clock = 1'b0;
202     //assign disp_rs = 1'b0;
203     //assign disp_ce_b = 1'b1;
204     //assign disp_reset_b = 1'b0;
205     //assign disp_data_out = 1'b0;
206
207     // Buttons, Switches, and Individual LEDs
208     //assign led = 8'hFF;
209
```

```
210     // User I/Os
211     assign user1 = 32'hZ;
212     assign user2 = 32'hZ;
213     assign user3 = 32'hZ;
214     assign user4 = 32'hZ;
215
216     // Daughtercard Connectors
217     assign daughtercard = 44'hZ;
218
219     // SystemACE Microprocessor Port
220     assign systemace_data = 16'hZ;
221     assign systemace_address = 7'h0;
222     assign systemace_ce_b = 1'b1;
223     assign systemace_we_b = 1'b1;
224     assign systemace_oe_b = 1'b1;
225
226     // Logic Analyzer
227     assign analyzer1_data = 16'h0;
228     assign analyzer1_clock = 1'b1;
229     assign analyzer2_data = 16'h0;
230     assign analyzer2_clock = 1'b1;
231     assign analyzer3_data = 16'h0;
232     assign analyzer3_clock = 1'b1;
233     assign analyzer4_data = 16'h0;
234     assign analyzer4_clock = 1'b1;
235
236
    ////////////////////////////////////////////////////////////////////////
    ////
237     //
238     // Final Project Components
239     //
240
    ////////////////////////////////////////////////////////////////////////
    ////
241
242   //
243     // DCM BLOCK
244     //
245   // Generate a 31.5MHz pixel clock from clock_27mhz
246     wire pclk, pixel_clock;
247     DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
248     // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
249     // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
250     // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm is "NONE"
251     BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));
252
253
254   parameter WIDTH = 10;
255   //
256   // Heartbeat
257   //
258   reg [29:0] ctr;
259   reg alt;
260   reg [7:0] ticker;
261   wire [WIDTH-1:0] symbol;
```

```
  262   reg [3:0] hold_symbol;
  263   wire ready;
  264     wire [5:0] decoded_bits;
  265     reg [5:0] hold_decoded_bits;
  266   always @(posedge pixel_clock) begin
  267     ctr <= ctr + 1;
  268     if (ctr > 30'd6300000) begin
  269       ctr <= 30'd0;
  270       alt <= ~alt;
  271       ticker <= ticker + 1;
  272       hold_symbol <= symbol[3:0];
  273     end
  274     if (ready) begin
  275       hold_decoded_bits <= decoded_bits;
  276     end
  277   end
  278
  279   assign led[7] = alt;
  280   //assign led[6:4] = 8'b1111111;
  281   //assign led[6:3] = hold_symbol;
  282   //assign led[2:1] = 2'b11;
  283   assign led[4:1] = decoded_bits;
  284
  285   //
  286   // SYNC BLOCK
  287   //
  288   wire reset, deb_button_up, deb_button_down, deb_button_left,
      deb_button_right;
  289   debounce reset_debouncer(1'b0, pixel_clock, ~button0, reset);
  290   debounce up_debouncer(1'b0, pixel_clock, ~button_up, deb_button_up);
  291   debounce down_debouncer(1'b0, pixel_clock, ~button_down,
      deb_button_down);
  292   debounce left_debouncer(1'b0, pixel_clock, ~button_left,
      deb_button_left);
  293   debounce right_debouncer(1'b0, pixel_clock, ~button_right,
      deb_button_right);
  294   assign led[6] = deb_button_left;
  295   assign led[5] = deb_button_right;
  296   assign led[0] = reset;
  297
  298    //
  299   // CLOCK BLOCK
  300   //
  301   wire enable;
  302   divider enabler(pixel_clock, enable, reset);
  303   reg pace;
  304   reg [6:0] pace_ctr;
  305   always @(posedge pixel_clock) begin
  306     if (reset) begin
  307       pace_ctr <= 0;
  308       pace <= 0;
  309     end else begin
  310       if (pace_ctr == 129) begin
  311         pace <= 1;
  312         pace_ctr <= pace_ctr + 1;
  313       end else if (pace_ctr == 130) begin
```

```
314         pace <= 0;
315         pace_ctr <= 0;
316       end else begin
317         pace_ctr <= pace_ctr + 1;
318       end
319     end
320   end
321
322   //
323   // ROM BLOCK
324   //
325
326   wire [5:0] addr;
327
328   rom uut (
329     .addr(addr),
330     .clk(pixel_clock),
331     .dout(symbol)
332   );
333
334   //
335   // CONTROL BLOCK
336   //
337   control mycontrol(pixel_clock, reset | enable | pace, deb_button_up,
    deb_button_down, addr);
338
339   //
340   // WALSH BLOCK
341   //
342   wire [2:0] num_fhts;
343
344   wire optimal;
345   assign optimal = 1'b1;//deb_button_down;
346   //walsh mywalsh(reset, enable | pace, pixel_clock, num_fhts, optimal,
    symbol, decoded_bits, ready);
347   //wire [5:0] decoded_bits;
348   assign decoded_bits = 6'b010101;
349   assign ready = 1'b1;
350   //
351   // BER BLOCK
352   //
353   wire [2:0] error_count;
354   wire [5:0] actual_bits;
355   assign actual_bits = decoded_bits + 1;
356   ber myber(decoded_bits, actual_bits, error_count);
357
358   //
359   // FEEDBACK BLOCK
360   //
361   wire enabled;
362   assign enabled = 1'b0;
363   feedback_path myfeedback(enable, reset, pixel_clock, enabled,
    deb_button_left, deb_button_right, error_count, num_fhts);
364
365   //
366   // VGA BLOCK
```

```
  367   //
  368   wire [9:0] pixel_count;
  369   wire [9:0] line_count;
  370   wire [7:0] red;
  371   wire [7:0] green;
  372   wire [7:0] blue;
  373   wire hsync, hblank, vsync, vblank;
  374   wire hsync_delayed, vsync_delayed;
  375
  376   wire [7:0] new_data_ctr;
  377   wire [7:0] error_ctr;
  378   display mydisplay(ready, pixel_clock, reset, enable, pixel_count,
      line_count, {red, green, blue}, decoded_bits[2:0], num_fhts,
      new_data_ctr, error_ctr);
  379   vga myvga(pixel_clock, reset, hsync, vsync, sync_b, blank_b,
      pixel_count, line_count);
  380
  381    // VGA output signals
  382    //
  383    // Inverting the clock to the DAC provides half a clock period for
      signals
  384    // to propagate from the FPGA to the DAC.
  385    assign vga_out_pixel_clock = ~pixel_clock;
  386   assign vga_out_red = red;
  387   assign vga_out_green = green;
  388   assign vga_out_blue = blue;
  389   assign vga_out_hsync = hsync;
  390   assign vga_out_vsync = vsync;
  391   assign vga_out_blank_b = blank_b;
  392   assign vga_out_sync_b = sync_b;
  393
  394   wire [639:0] final_dots;
  395   wire [79:0] symbol_dots;
  396   wire [79:0] ticker_dots;
  397   wire [79:0] error_dots;
  398   wire [79:0] decoded_dots;
  399   wire [79:0] data_ctr_dots;
  400   wire [79:0] error_ctr_dots;
  401   wire [79:0] fhts_dots;
  402   alpha_dots symbol_dots_guy(pixel_clock, hold_symbol[3:0],
      symbol_dots);
  403   alpha_dots ticker_dots_guy(pixel_clock, ticker[3:0], ticker_dots);
  404   alpha_dots error_dots_guy(pixel_clock, {1'b0, error_count[2:0]},
      error_dots);
  405   alpha_dots decoded_dots_guy(pixel_clock, {1'b0, decoded_bits[2:0]},
      decoded_dots);
  406   alpha_dots data_ctr_dots_guy(pixel_clock, new_data_ctr[3:0],
      data_ctr_dots);
  407   alpha_dots error_ctr_dots_guy(pixel_clock, error_ctr[3:0],
      error_ctr_dots);
  408   alpha_dots fhts_dots_guy(pixel_clock, {1'b0, num_fhts}, fhts_dots);
  409
  410    assign final_dots = {symbol_dots, error_dots, decoded_dots,
      data_ctr_dots, error_ctr_dots, fhts_dots, 80'b0, ticker_dots};
  411   alpha_display disp(pixel_clock, reset, deb_button_up, disp_blank,
      disp_clock, disp_rs, disp_ce_b, disp_reset_b, disp_data_out,
```

```
411  final_dots);
412  endmodule
```

```
/*****************************************************************
********
*     This file is owned and controlled by Xilinx and must be used
*
*     solely for design, simulation, implementation and creation of
*
*     design files limited to Xilinx devices or technologies. Use
*
*     with non-Xilinx devices or technologies is expressly prohibited
*
*     and immediately terminates your license.
*
*
*
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
*
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR
*
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION
*
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION
*
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS
*
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
*
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
*
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY
*
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
*
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
*
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
*
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*
*     FOR A PARTICULAR PURPOSE.
*
*
*
*     Xilinx products are not intended for use in life support
*
*     appliances, devices, or systems. Use in such applications are
*
*     expressly prohibited.
*
*
*
*     (c) Copyright 1995-2004 Xilinx, Inc.
*
*
*
*****************************************************************
*******/
```

```
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 font_rom.v when simulating
34 // the core, font_rom. 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 Help".
37
38 `timescale 1ns/1ps
39
40 module font_rom(
41   addr,
42   clk,
43   dout);
44
45
46 input [10 : 0] addr;
47 input clk;
48 output [7 : 0] dout;
49
50 // synopsys translate_off
51
52       BLKMEMSP_V6_1 #(
53     11, // c_addr_width
54     "0",  // c_default_data
55     1536, // c_depth
56     0,  // c_enable_rlocs
57     0,  // c_has_default_data
58     0,  // c_has_din
59     0,  // c_has_en
60     0,  // c_has_limit_data_pitch
61     0,  // c_has_nd
62     0,  // c_has_rdy
63     0,  // c_has_rfd
64     0,  // c_has_sinit
65     0,  // c_has_we
66     18, // c_limit_data_pitch
67     "font_rom.mif", // c_mem_init_file
68     0,  // c_pipe_stages
69     0,  // c_reg_inputs
70     "0",  // c_sinit_value
71     8,  // c_width
72     0,  // c_write_mode
73     "0",  // c_ybottom_addr
74     1,  // c_yclk_is_rising
75     1,  // c_yen_is_high
76     "hierarchy1", // c_yhierarchy
77     0,  // c_ymake_bmm
78     "16kx1",  // c_yprimitive_type
79     1,  // c_ysinit_is_high
80     "1024", // c_ytop_addr
81     0,  // c_yuse_single_primitive
82     1,  // c_ywe_is_high
```

```
   83     1)  // c_yydisable_warnings
   84   inst (
   85     .ADDR(addr),
   86     .CLK(clk),
   87     .DOUT(dout),
   88     .DIN(),
   89     .EN(),
   90     .ND(),
   91     .RFD(),
   92     .RDY(),
   93     .SINIT(),
   94     .WE());
   95
   96
   97 // synopsys translate_on
   98
   99 // FPGA Express black box declaration
  100 // synopsys attribute fpga_dont_touch "true"
  101 // synthesis attribute fpga_dont_touch of font_rom is "true"
  102
  103 // XST black box declaration
  104 // box_type "black_box"
  105 // synthesis attribute box_type of font_rom is "black_box"
  106
  107 endmodule
  108
```

```
 1 `timescale 1ns / 1ps
 2
 3 module gauge(start_x, end_x, y, pixel, value, line, color);
 4   input [9:0] start_x, end_x, y;
 5   input [9:0] pixel, line;
 6   input [2:0] value;
 7
 8   output [23:0] color;
 9
10   reg [23:0] color;
11
12   parameter WHITE = 24'hFFFFFF;
13   parameter BLACK = 24'h0;
14   parameter GREEN = 24'h00FF00;
15   parameter YELLOW_GREEN = 24'h9acd32;
16   parameter GREEN_YELLOW = 24'hadff2f;
17   parameter ORANGE_RED = 24'hff4500;
18   parameter ORANGE = 24'hffa500;
19   parameter YELLOW = 24'hFFFF00;
20
21   parameter RED = 24'hFF0000;
22
23   always @(pixel or line or start_x or end_x or y) begin
24     if (pixel > start_x + 1 && pixel < end_x - 1) begin
25       if (line > y - (value + 1) * 10 && line < y) begin // value + 1 *
   5 is STUBBED
26         if (line > y - 10 && line < y - 1) begin
27           color = GREEN;
28         end else if (line > y - 20 && line < y - 10) begin
29           color = YELLOW_GREEN;
30         end else if (line > y - 30 && line < y - 20) begin
31           color = GREEN_YELLOW;
32         end else if (line > y - 40 && line < y - 30) begin
33           color = YELLOW;
34         end else if (line > y - 50 && line < y - 40) begin
35           color = YELLOW;
36         end else if (line > y - 60 && line < y - 50) begin
37           color = ORANGE;
38         end else if (line > y - 70 && line < y - 60) begin
39           color = ORANGE_RED;
40         end else if (line > y - 80 && line < y - 70) begin
41           color = RED;
42         end else begin
43           color = BLACK;
44         end
45       end else begin
46         color = BLACK;
47       end
48     end else begin
49       color = BLACK;
50     end
51   end
52
53
54 endmodule
```

```
 1 `timescale 1ns / 1ps
 2 //////////////////////////////////////////////////////////////////////////////////
 3 // Company:
 4 // Engineer:
 5 //
 6 // Create Date:    16:32:17 05/03/06
 7 // Design Name:
 8 // Module Name:    HorizontalLine
 9 // Project Name:
10 // Target Device:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module horizontal_line(start_x, end_x, y, pixel, line, color);
22   input [9:0] start_x, end_x, y;
23   input [9:0] pixel, line;
24
25   output [23:0] color;
26
27   reg [23:0] color;
28
29   parameter WHITE = 24'hFFFFFF;
30   parameter BLACK = 24'h0;
31
32   always @(pixel or line or start_x or end_x or y) begin
33     if (pixel >= start_x && pixel <= end_x && line == y) begin
34       color = WHITE;
35     end else begin
36       color = BLACK;
37     end
38   end
39 endmodule
```

```
 1 `timescale 1ns / 1ps
 2
 3 module optimal_decoder(reset, clk, symbol_in, six_bits, ready, ctr,
   upper_input, lower_input, upper_output, lower_output, fht_ctr);
 4   input reset, clk;
 5   parameter WIDTH=10;
 6   input [WIDTH-1:0] symbol_in;
 7   output [5:0] six_bits;
 8   output ready;
 9   output [6:0] ctr;
10   output [WIDTH-1:0] upper_input, lower_input;
11   output [WIDTH-1:0] upper_output, lower_output;
12   output [5:0] fht_ctr;
13   wire ready;
14   reg [WIDTH-1:0] symbols [63:0];
15
16   wire enable_fht;
17   reg [WIDTH-1:0] upper_input, lower_input;
18   wire [WIDTH-1:0] confidence;
19   wire fht_ready;
20   wire [5:0] fht_ctr;
21   wire [WIDTH-1:0] upper_output, lower_output;
22   fht_64_symbol fht(enable_fht | reset, clk, upper_input, lower_input,
   six_bits, confidence, fht_ready, upper_output, lower_output, fht_ctr);
23
24   reg [6:0] ctr;
25   reg [3:0] saw_ready_ctr;
26   assign enable_fht = (ctr < 65);
27   assign ready = fht_ready && !(saw_ready_ctr == 4'b11);
28   always @(posedge clk) begin
29     if (reset) begin
30       ctr <= 0;
31       saw_ready_ctr <= 0;
32     end else begin
33       symbols[ctr-1] <= symbol_in;
34       if (ctr > 63) begin
35         upper_input <= symbols[ctr - 64];
36         lower_input <= symbols[ctr - 32];
37       end
38       ctr <= ctr + 1;
39       if (fht_ready) begin
40         if (saw_ready_ctr < 10) begin
41           saw_ready_ctr <= saw_ready_ctr + 1;
42         end
43       end
44     end
45   end
46 endmodule
```

```
 1 % Walsh Decoding Implemented by Projected 8-Point Fast Hadamard
   Transforms
 2
 3 num_codewords = 6;    % Number of codewords
 4 num_FHT = 8;      % Number of FHTs used to determine each triplet of
   bits (up to 8)
 5 SNR = -8;     % dB
 6
 7 %%%%%%%%%%%%%%%%%%%%%%% Generate Random Data
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 8 bits = round(rand(6*num_codewords,1));
 9
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%% Walsh Encoder %%%%%%%%%%%%%%%%%%%%%%%%%%%%
11 H = de2bi(0:63);
   % Walsh codeword generator matrix
12 Walsh_chips =
   mod(reshape(H*reshape(bits,6,num_codewords),64*num_codewords,1),2);
   % Compute Walsh codewords
13 Walsh_index = bi2de(reshape(bits,6,num_codewords)')+1;
   % Compute Walsh indices
14
15 %%%%%%%%%%%%%%% AWGN Channel %%%%%%%%%%%%%%%%
16 r = (1-2*Walsh_chips) + sqrt(10^(-SNR/10))*randn(64*num_codewords,1);
17
18 %%%%%%%%%%% Assume Perfect Synchronization %%%%%%%%%%%%%%
19 y = reshape(r,64,num_codewords);
20
21 %%%%%%%%%%% Decode Three Bits at a Time (When k=1, decode first 3 bits
   of codeword; when k=4, decoder last 3 bits of codeword) %%%%%%%%%%%%
22 for i=0:num_codewords-1
23   for k=1:3:4
24
25     %%%%%%%%%% Randomly Choose A Number of Distinct 3-Bit Patterns Equal
   to Number of FHTs %%%%%%%%%%
26     [s,I] = sort(rand(8,1));
27     tmp = fliplr(de2bi(I(1:num_FHT)-1,3));
28
29     %%%%%%%%%%%%%%% Compute 6-Bit Patterns Corresponding to Indices of
   FHT Inputs %%%%%%%%%%%%%%%
30     bit_index_000 = [tmp(:,1:k-1) zeros(num_FHT,3) tmp(:,k:3)];
31     bit_index_100 = [tmp(:,1:k-1) ones(num_FHT,1) zeros(num_FHT,2)
   tmp(:,k:3)];
32     bit_index_010 = [tmp(:,1:k-1) zeros(num_FHT,1) ones(num_FHT,1)
   zeros(num_FHT,1) tmp(:,k:3)];
33     bit_index_110 = [tmp(:,1:k-1) ones(num_FHT,2) zeros(num_FHT,1)
   tmp(:,k:3)];
34     bit_index_001 = [tmp(:,1:k-1) zeros(num_FHT,2) ones(num_FHT,1)
   tmp(:,k:3)];
35     bit_index_101 = [tmp(:,1:k-1) ones(num_FHT,1) zeros(num_FHT,1)
   ones(num_FHT,1) tmp(:,k:3)];
36     bit_index_011 = [tmp(:,1:k-1) zeros(num_FHT,1) ones(num_FHT,2)
   tmp(:,k:3)];
37     bit_index_111 = [tmp(:,1:k-1) ones(num_FHT,3) tmp(:,k:3)];
38
39     %%%%%%%%%%%%%%% Compute FHTs from Received Signal %%%%%%%%%%%%%%%
40     Z = sum(fht(reshape(y([bi2de(bit_index_000) bi2de(bit_index_100)
```

```
40 bi2de(bit_index_010) bi2de(bit_index_110) ...
41                              bi2de(bit_index_001) bi2de(bit_index_101)
   bi2de(bit_index_011) bi2de(bit_index_111)]'+1,i+1),8,num_FHT)).^2,2);
42
43     %%%%%%%%%%%% Hard Decison %%%%%%%%%%%%%
44     [max_corr,decoded_Walsh_index] = max(Z);
45     decoded_Walsh_bits(6*i+k:6*i+k+2,1) =
   de2bi(decoded_Walsh_index-1,3)';
46
47   end
48 end
49
50 %%%%%%%%% Compute Walsh Indices for Decoded Walsh Codewords %%%%%%%%%%
51 decoded_Walsh_index =
   bi2de(reshape(decoded_Walsh_bits,6,num_codewords)')+1;
52
53 bit_errors = sum(decoded_Walsh_bits~=bits)
54 codeword_errors = sum(decoded_Walsh_index~=Walsh_index)
55
```

```
/*****************************************************************
********
*     This file is owned and controlled by Xilinx and must be used
*
*     solely for design, simulation, implementation and creation of
*
*     design files limited to Xilinx devices or technologies. Use
*
*     with non-Xilinx devices or technologies is expressly prohibited
*
*     and immediately terminates your license.
*
*
*
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
*
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR
*
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION
*
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION
*
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS
*
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
*
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
*
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY
*
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
*
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
*
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
*
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*
*     FOR A PARTICULAR PURPOSE.
*
*
*
*     Xilinx products are not intended for use in life support
*
*     appliances, devices, or systems. Use in such applications are
*
*     expressly prohibited.
*
*
*
*     (c) Copyright 1995-2004 Xilinx, Inc.
*
*
*
*****************************************************************
*******/
```

```
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 rom.v when simulating
34 // the core, rom. 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 Help".
37
38 `timescale 1ns/1ps
39
40 module rom(
41   addr,
42   clk,
43   dout);
44
45
46 input [5 : 0] addr;
47 input clk;
48 output [9 : 0] dout;
49
50 // synopsys translate_off
51
52       BLKMEMSP_V6_1 #(
53     6,  // c_addr_width
54     "0",  // c_default_data
55     64, // c_depth
56     0,  // c_enable_rlocs
57     0,  // c_has_default_data
58     0,  // c_has_din
59     0,  // c_has_en
60     0,  // c_has_limit_data_pitch
61     0,  // c_has_nd
62     0,  // c_has_rdy
63     0,  // c_has_rfd
64     0,  // c_has_sinit
65     0,  // c_has_we
66     18, // c_limit_data_pitch
67     "rom.mif",  // c_mem_init_file
68     0,  // c_pipe_stages
69     0,  // c_reg_inputs
70     "0",  // c_sinit_value
71     10, // c_width
72     0,  // c_write_mode
73     "0",  // c_ybottom_addr
74     1,  // c_yclk_is_rising
75     1,  // c_yen_is_high
76     "hierarchy1", // c_yhierarchy
77     0,  // c_ymake_bmm
78     "16kx1",  // c_yprimitive_type
79     1,  // c_ysinit_is_high
80     "1024", // c_ytop_addr
81     0,  // c_yuse_single_primitive
82     1,  // c_ywe_is_high
```

```
   83     1)  // c_yydisable_warnings
   84   inst (
   85     .ADDR(addr),
   86     .CLK(clk),
   87     .DOUT(dout),
   88     .DIN(),
   89     .EN(),
   90     .ND(),
   91     .RFD(),
   92     .RDY(),
   93     .SINIT(),
   94     .WE());
   95
   96
   97 // synopsys translate_on
   98
   99 // FPGA Express black box declaration
  100 // synopsys attribute fpga_dont_touch "true"
  101 // synthesis attribute fpga_dont_touch of rom is "true"
  102
  103 // XST black box declaration
  104 // box_type "black_box"
  105 // synthesis attribute box_type of rom is "black_box"
  106
  107 endmodule
  108
```

```
 1  `timescale 1ns / 1ps
 2
 3  module shift_register(reset, clk, shift, sr_in, sr_out);
 4     input reset, clk, shift;
 5    parameter WIDTH = 10;
 6    input [WIDTH-1:0] sr_in;
 7    output [WIDTH-1:0] sr_out;
 8
 9     parameter SIZE = 64;
10
11    wire [WIDTH-1:0] sr_out;
12
13    reg [WIDTH-1:0] sr [SIZE-1:0];
14    integer i;
15    always @(posedge clk) begin
16      if (reset) begin
17        for (i = 0; i < SIZE; i = i + 1) begin
18          sr[i] <= 100'b0;
19        end
20      end
21      if (shift && !reset) begin
22        for (i = 1; i < SIZE; i = i + 1) begin
23          sr[SIZE - i] <= sr[SIZE - i - 1];
24        end
25        //sr[SIZE - 1:1] <= sr[SIZE - 2:0];
26        sr[0] <= sr_in;
27      end
28    end
29
30    assign sr_out = sr[SIZE - 1];
31  endmodule
```

```
 1 import math, string, sys
 2 def de2bi(n):
 3     '''convert denary integer n to binary string bStr'''
 4     bStr = ''
 5     if n < 0:  raise ValueError, "must be a positive integer"
 6     if n == 0: return '0'
 7     while n > 0:
 8         bStr = str(n % 2) + bStr
 9         n = n >> 1
10     return bStr
11
12 f = open(sys.argv[1], 'r')
13 lines = f.readlines()
14
15 for num in lines:
16   num = float(num.strip())
17   num = num * 511 / 50
18   num = math.ceil(num)
19   if num >= 0:
20     num = "0" + string.zfill(str(de2bi(int(num))), 9)
21   else:
22     num = 511 + int(num)
23     num = "1" + string.zfill(str(de2bi(num)), 9)
24   print num + ","
25
```