```verilog
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Accumulates one color component and prevents overflow
//
////////////////////////////////////////////////////////////////////////////////
module color_component_acc(clk, reset, in, sum);
   input clk;
   input reset;
   input [7:0] in;
   output [7:0] sum;
   reg [7:0] b, q;

   parameter MAX = 8'hFF;

   wire [7:0] x;
   wire cout;
   assign {cout, x} = b + q;
   assign sum = cout ? MAX : x;

   always @ (posedge clk)
    begin
       b <= in;
       if (reset)
          q <= 0;
       else
          q <= sum;
    end

endmodule


/////////////////////////////////////////////////////////////////////////////////
// Accumulates one color component and prevents overflow
//
////////////////////////////////////////////////////////////////////////////////
module color_acc(clk, reset, color_red, color_green, color_blue, color);
   input clk;
   input reset;
   input [7:0] color_red, color_green, color_blue;
   output [23:0] color;

   color_component_acc red_acc (clk, reset, color_red, color[23:16]);
   color_component_acc green_acc (clk, reset, color_green, color[15:8]);
   color_component_acc blue_acc (clk, reset, color_blue, color[7:0]);

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Debounce/Synchronize module
//
//
// Use your system clock for the clock input to produce a synchronous,
// debounced output
//
////////////////////////////////////////////////////////////////////////////////
module debounce (reset, clock, noisy, clean);
   parameter DELAY = 270000;   // .01 sec with a 27Mhz clock
   input reset, clock, noisy;
   output clean;

   reg [18:0] count;
   reg new, clean;

   always @(posedge clock)
      if (reset)
       begin
          count <= 0;
          new <= noisy;
          clean <= noisy;
       end
      else if (noisy != new)
       begin
          new <= noisy;
          count <= 0;
       end
      else if (count == DELAY)
          clean <= new;
      else
          count <= count+1;

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Delays input by two clock cycles
//
////////////////////////////////////////////////////////////////////////////////
module delay3(clk, in, out);
    input clk, in;
    output out;

     reg a, b, out;

     always @ (posedge clk)
     begin
       a <= in;
       b <= a;
       out <= b;
     end

endmodule

////////////////////////////////////////////////////////////////////////////////
//
// Delays input by 5 clock cycles
//
////////////////////////////////////////////////////////////////////////////////
module delay5(clk, in, out);
    input clk, in;
    output out;

     reg a, b, c, d, out;

     always @ (posedge clk)
     begin
       a <= in;
       b <= a;
       c <= b;
       d <= c;
       out <= d;
     end

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
Divider Module
// Sam Gross and Adam Lerer
//
// Latency: 7
// Throughput: 1/7 (can be improved)
// Speed: ~104 Mhz or 85?
//
////////////////////////////////////////////////////////////////////////////////
module div18(clk, a_in, b_in, quot, start, done);
    input clk;
    input [16:0] a_in;
    input [16:0] b_in;
    reg [16:0] a;
    output reg [17:0] quot;
    input start;
    output done;

    reg [17:0] quot_next;

    parameter ONE = 18'h20000;      // 1.0
    parameter FACTOR = 18'h2E9FB;   // 1.457

    reg [4:0] shift, shift_next;
    wire [4:0] shift_out;
    wire [16:0] b_norm;
    reg signed [17:0] den, den_next;

    fp_norm normalizer (clk, 1'b1, b_in, b_norm, shift_out);

    reg [17:0] rcp, next_rcp; // reciprocal of denominator
    reg [17:0] mult_a, mult_b;
    wire [17:0] product;
    wire [35:0] mult_out;
    wire [34:0] temp;
    assign temp = mult_out << (shift + 1);

    reg [2:0] state;
    //assign quot = next_rcp;
    assign done = (state == 7);

    mult_full multiplier (clk, mult_a, mult_b, mult_out);

    assign product = mult_out[34:17];

    always @ *
     begin
      den_next = den;
      next_rcp = rcp;
      quot_next = quot;
      mult_a = 18'hz;
      mult_b = 18'hz;
      shift_next = shift;
      case (state)
         3'd0:  // --------------------- clk = 0
         begin
            den_next = -b_norm;
            shift_next = shift_out;
         end
         3'd1: // --------------------- clk = 1
```

```
          begin
            next_rcp = FACTOR + den;
            mult_a = den;
            mult_b = next_rcp;
          end
         3'd2: // -------------------- clk = 2
          begin
            mult_a = rcp;
            mult_b = ONE + product;
          end
         3'd3: // -------------------- clk = 3
          begin
            next_rcp = product << 1;
            mult_a = den;
            mult_b = next_rcp;
          end
         3'd4: // -------------------- clk = 4
          begin
            mult_a = rcp;
            mult_b = ONE + product;
          end
         3'd5: // -------------------- clk = 5, product contains rcp
          begin
            mult_a = a;
            mult_b = product << 1;
           end
         3'd6: // -------------------- clk = 6 compute division
          begin
            quot_next = temp[34:17];
          end
          // ------------------------- clk = 7 answer out
          3'd7:
          begin
          end
         endcase
    end

   always @ (posedge clk)
    begin
     if (start)
      begin
        rcp <= 0;
        state <= 0;
        a <= a_in;
      end
     else
      begin
        if (state != 3'd7)
           state <= state + 1;
        rcp <= next_rcp;
      end
     den <= den_next;
     shift <= shift_next;
     quot <= quot_next;
    end
endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Normalizes fixed point numbers to between 0.5 and 1
//
////////////////////////////////////////////////////////////////////////////////
module fp_norm(clk, ce, in, out, shift);
    input clk;
    input ce;
    input [16:0] in;
    output [16:0] out;
    output [4:0] shift;

    reg [16:0] num;
    reg [4:0] shift;

    always @ (posedge clk)
        if (ce)
            num <= in;

    always @ (num)
         casez (num[16:1])
            16'b1zzz_zzzz_zzzz_zzzz:
                shift = 5'd0;
            16'b01zz_zzzz_zzzz_zzzz:
                shift = 5'd1;
            16'b001z_zzzz_zzzz_zzzz:
                shift = 5'd2;
            16'b0001_zzzz_zzzz_zzzz:
                shift = 5'd3;
            16'b0000_1zzz_zzzz_zzzz:
                shift = 5'd4;
            16'b0000_01zz_zzzz_zzzz:
                shift = 5'd5;
            16'b0000_001z_zzzz_zzzz:
                shift = 5'd6;
            16'b0000_0001_zzzz_zzzz:
                shift = 5'd7;
            16'b0000_0000_1zzz_zzzz:
                shift = 5'd8;
            16'b0000_0000_01zz_zzzz:
                shift = 5'd9;
            16'b0000_0000_001z_zzzz:
                shift = 5'd10;
            16'b0000_0000_0001_zzzz:
                shift = 5'd11;
            16'b0000_0000_0000_1zzz:
                shift = 5'd12;
            16'b0000_0000_0000_01zz:
                shift = 5'd13;
            16'b0000_0000_0000_001z:
                shift = 5'd14;
            16'b0000_0000_0000_0001:
                shift = 5'd15;
            default:
                shift = 5'd16;
        endcase

     assign out = (num << shift) + 1;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Intersector module
//
// Problems: None?
//
//////////////////////////////////////////////////////////////////////////////////
module intersector(clk, reset, ray, min_dist, stop_on_intersect, start, intersected,
shape_id, int_pos, normal, done, new_frame);
    input clk;
    input reset;

    input new_frame;
    input [107:0] ray;
    wire signed [17:0] orig_x, orig_y, orig_z, vec_x, vec_y, vec_z;
    assign {orig_x, orig_y, orig_z, vec_x, vec_y, vec_z} = ray;

    input start;
    input [16:0] min_dist;
    input stop_on_intersect;

    output intersected;
    output reg [3:0] shape_id;
    output [53:0] int_pos;
    output reg [53:0] normal;
    output done;

    parameter MIN_T_VAL = 18'd131;

    parameter IDLE = 0;
    parameter REQ_SHAPE = 1;
    parameter CALC_PLANE_DPS = 2;
    parameter CALC_PLANE_DPS2 = 3;
    parameter CALC_PLANE_DPS3 = 4;
    parameter GET_PLANE_DPS = 5;
    parameter GET_PLANE_DPS2 = 6;
    parameter CALC_SPH_DPS = 7;
    parameter CALC_SPH_DPS2 = 8;
    parameter CALC_SPH_DPS3 = 9;
    parameter GET_SPH_DPS = 10;
    parameter CALC_SPH_DISC = 11;
    parameter CALC_SPH_DISC2 = 12;
    parameter CALC_MIN_P = 13;
    parameter CALC_MIN_P2 = 14;
    parameter CALC_MIN_P3 = 15;
    parameter WAIT_PIPE = 16;
    parameter CALC_SPH_NORMAL = 17;
    parameter CALC_SPH_NORMAL2 = 18;
    parameter COMPARE_SPH = 19;
    parameter COMPARE_SPH2 = 20;
    parameter COMPARE_SPH3 = 21;
    parameter COMPARE_PLANE = 22;
    parameter COMPARE_PLANE2 = 23;
    parameter COMPARE_PLANE3 = 24;

    parameter SPHERE = 0;
    parameter PLANE = 1;

    reg done;
    reg signed [17:0] int_pos_x, int_pos_y, int_pos_z;
```

```
    reg signed [17:0] int_pos_x_next, int_pos_y_next, int_pos_z_next;
    assign int_pos = {int_pos_x, int_pos_y, int_pos_z};
    reg [4:0] state, next_state;
    reg [34:0] normal_x_int, normal_y_int, normal_z_int;
    reg signed [17:0] normal_x_next, normal_y_next, normal_z_next;

    // multiplier / accumulators
    reg reset1, reset2;
    reg signed [17:0] a1, b1, a2, b2, acc1, acc2;
    wire signed [17:0] q1, q2;
    wire [16:0] prod_low1, prod_low2;
    mac mac1(clk, reset1, a1, b1, q1, prod_low1);
    mac mac2(clk, reset2, a2, b2, q2, prod_low2);

    // divider
    reg [16:0] numerator, divisor;
    wire signed [17:0] quotient;
    reg div_start;
    wire div_done;
    reg div_loaded;
    reg [3:0] div_shape;
    div18 plane_div(clk, numerator, divisor, quotient, div_start, div_done);

    // sqrt
    reg signed [17:0] sqrt_in;
    wire signed [17:0] sqrt_out;
    reg sqrt_start;
    wire sqrt_done;
    reg signed [17:0] sqrt_b, sqrt_b_buf;
    reg signed [17:0] b, b_next;
    reg sqrt_sign, sqrt_sign_buf;
    wire signed [17:0] sqrt_sum;
    reg [1:0] sqrt_loaded;
    reg [3:0] sqrt_shape, sqrt_shape_buf;
    sqrt18 sph_sqrt (.x_in(sqrt_in), .clk(clk), .x_out(sqrt_out), .nd(sqrt_start),
.rdy(sqrt_done));

    assign sqrt_sum = -sqrt_b + (sqrt_sign ? -sqrt_out : sqrt_out);

    // shapes
    wire [3:0] n;
    reg [3:0] cur_shape;
    wire shape_type;
    wire [53:0] shape_vector;
    wire signed [17:0] shape_x, shape_y, shape_z;
    assign {shape_x, shape_y, shape_z} = shape_vector;
    wire signed [17:0] shape_scalar;
    wire [16:0] sph_inv_rad;
    wire [4:0] sph_inv_rad_mag;
    shapes scene_shapes(clk, reset, n, cur_shape, shape_type, shape_vector,
shape_scalar, sph_inv_rad, sph_inv_rad_mag, new_frame);

    // comparing for shadows
    reg set_intersected;
    reg comp_sign, comp_sign_next;
    reg [16:0] comp_disc, comp_disc_next;

    //random stuff
    reg intersected;
    reg signed [17:0] Vd, V0, Vd_next, V0_next;
    reg [16:0] min_t;
    reg x_sign, x_sign_next, y_sign, y_sign_next, z_sign, z_sign_next;
    //records for closest shape
```

```
    always @ *
     begin
        a1 = 0;
        a2 = 0;
        b1 = 0;
        b2 = 0;
        V0_next = V0;
        Vd_next = Vd;
        reset1 = 0;
        reset2 = 0;
        div_start = 0;
        sqrt_start = 0;
        numerator = 0;
        divisor = 0;
        sqrt_in = 0;
        done = 0;
        next_state = state;
        b_next = b;
        set_intersected = 0;
        {normal_x_next, normal_y_next, normal_z_next} = normal;
        {int_pos_x_next, int_pos_y_next, int_pos_z_next} = {int_pos_x, int_pos_y,
int_pos_z};
        {x_sign_next, y_sign_next, z_sign_next} = {x_sign, y_sign, z_sign};
        comp_sign_next = comp_sign;
        comp_disc_next = comp_disc;
        case (state)
          IDLE:
           begin
             next_state = IDLE;
             done = 1;
           end
          REQ_SHAPE:
           begin
             reset1 = 1;
             reset2 = 1;
             if (cur_shape < n)
                next_state = (shape_type == PLANE) ? CALC_PLANE_DPS : CALC_SPH_DPS;
             else if (sqrt_loaded == 0 && div_loaded == 0)
                next_state = intersected ? CALC_MIN_P : IDLE; //should self-cycle
until the sqrt/div are finished
             else
                next_state = WAIT_PIPE;
           end
          WAIT_PIPE:
           begin
            if(sqrt_loaded == 0 && div_loaded == 0)
             next_state = intersected ? CALC_MIN_P : IDLE;
            else
             next_state = WAIT_PIPE;
           end
          CALC_PLANE_DPS:
           begin
             next_state = CALC_PLANE_DPS2;
             a1 = -vec_x;
             b1 = shape_x;
             a2 = orig_x;
             b2 = shape_x;
           end
          CALC_PLANE_DPS2:
           begin
             next_state = CALC_PLANE_DPS3;
             a1 = -vec_y;
```

```
              b1 = shape_y;
              a2 = orig_y;
              b2 = shape_y;
            end
           CALC_PLANE_DPS3:
            begin
              next_state = GET_PLANE_DPS;
              a1 = -vec_z;
              b1 = shape_z;
              a2 = orig_y;
              b2 = shape_z;
            end
           GET_PLANE_DPS:
            begin
              V0_next = q2 + shape_scalar;
              Vd_next = q1;
              next_state = stop_on_intersect ? COMPARE_PLANE : GET_PLANE_DPS2;
            end
           COMPARE_PLANE:
            begin
              reset1 = 1;
              a1 = min_t;
              b1 = Vd;
              next_state = V0 > 0 && Vd > 0 && Vd > V0 ? COMPARE_PLANE2 : REQ_SHAPE;
            end
           COMPARE_PLANE2:
              next_state = COMPARE_PLANE3;
           COMPARE_PLANE3:
            begin
              set_intersected = V0 < acc1;
              next_state = set_intersected ? IDLE : REQ_SHAPE;
            end
           GET_PLANE_DPS2:
            begin
              // WARNING: Assuming division takes less time than intersection for the
sake of speed
              // If we want to fix, we can just add condition to div_start
              next_state = REQ_SHAPE;
              numerator = V0[16:0];
              divisor = Vd[16:0];
              if (V0 > 0 && Vd > 0 && Vd > V0)
               begin
                 div_start = 1;
               end
            end
           CALC_SPH_DPS:
            begin
              next_state = CALC_SPH_DPS2;
              a1 = vec_x;
              b1 = orig_x - shape_x;
              a2 = b1;
              b2 = b1;
            end
           CALC_SPH_DPS2:
            begin
              next_state = CALC_SPH_DPS3;
              a1 = vec_y;
              b1 = orig_y - shape_y;
              a2 = b1;
              b2 = b1;
            end
           CALC_SPH_DPS3: // 09
            begin
```

```
      next_state = GET_SPH_DPS;
      a1 = vec_z;
      b1 = orig_z - shape_z;
      a2 = b1;
      b2 = b1;
   end
  GET_SPH_DPS: //0a
   begin
      next_state = CALC_SPH_DISC;
      reset1 = 1; //we want to reset mac 1 to reuse to calc b^2
      a1 = q1;
      b1 = q1;
      a2 = shape_scalar;
      b2 = -shape_scalar;
      b_next = q1;
   end
  CALC_SPH_DISC: //0b
   begin
      // register accumulations to enable higher clock speeds
      next_state = CALC_SPH_DISC2;
   end
  CALC_SPH_DISC2: //0c
   begin
      //q1 = b^2
      //q2 = c
      sqrt_in = (acc1 - acc2);
      comp_sign_next = (b < 0 && acc2 > 0);
      comp_disc_next = sqrt_in;
      if (stop_on_intersect)
       begin
         if (acc1 > acc2)
            next_state = COMPARE_SPH;
         else
            next_state = REQ_SHAPE;
       end
      else
       begin
         next_state = REQ_SHAPE;
         if (acc1 > acc2)
            sqrt_start = 1;
       end
   end
  COMPARE_SPH:
   begin
      reset1 = 1;
      a1 = min_t + b;
      b1 = a1;
      set_intersected = comp_sign && a1 > 0;
      if (set_intersected)
         next_state = IDLE;
      else if (!comp_sign && a1 < 0)
         next_state = REQ_SHAPE;
      else
         next_state = COMPARE_SPH2;
   end
  COMPARE_SPH2:
      next_state = COMPARE_SPH3;
  COMPARE_SPH3:
   begin
      if (comp_sign)
         set_intersected = comp_disc > acc1;
      else
```

```
                set_intersected = comp_disc < acc1 && comp_disc > (b << 1) +
MIN_T_VAL;
             next_state = set_intersected ? IDLE : REQ_SHAPE;
          end
         CALC_MIN_P:
          begin
             next_state = CALC_MIN_P2;
             a1 = vec_x;
             b1 = min_t;
             a2 = vec_y;
             b2 = min_t;
          end
         CALC_MIN_P2:
          begin
             next_state = CALC_MIN_P3;
             int_pos_x_next = q1 + orig_x;
             int_pos_y_next = q2 + orig_y;
             reset1 = 1;
             a1 = vec_z;
             b1 = min_t;
          end
         CALC_MIN_P3:
          begin
             //assume it's a sphere, start on normal
             reset1 = 1;
             a1 = int_pos_x - shape_x;
             x_sign_next = a1[17];
             b1 = sph_inv_rad;
             reset2 = 1;
             a2 = int_pos_y - shape_y;
             y_sign_next = a2[17];
             b2 = sph_inv_rad;
             int_pos_z_next = q1 + orig_z;
             // assume plane calculate normal
             {normal_x_next, normal_y_next, normal_z_next} = shape_vector;
             if (shape_type == PLANE)
                next_state = IDLE;
             else
                next_state = CALC_SPH_NORMAL;
          end
         CALC_SPH_NORMAL:
          begin
             next_state = CALC_SPH_NORMAL2;

             normal_x_int = ({q1, prod_low1} << sph_inv_rad_mag);
             if (x_sign == normal_x_int[34])
                normal_x_next = normal_x_int[34:17];
             else if (x_sign == 0)
                normal_x_next = 18'h1ffff;
             else
                normal_x_next = 18'h20000;

             normal_y_int = ({q2, prod_low2} << sph_inv_rad_mag);
             if (y_sign == normal_y_int[34])
                normal_y_next = normal_y_int[34:17];
             else if (y_sign == 0)
                normal_y_next = 18'h1ffff;
             else
                normal_y_next = 18'h20000;

             reset1 = 1;
             a1 = int_pos_z_next - shape_z;
             b1 = sph_inv_rad;
```

```
            z_sign_next = a1[17];
         end
        CALC_SPH_NORMAL2:
         begin
            next_state = IDLE;
            normal_z_int = ({q1, prod_low1} << sph_inv_rad_mag);
            if (z_sign == normal_z_int[34])
               normal_z_next = normal_z_int[34:17];
            else if (z_sign == 0)
               normal_z_next = 18'h1ffff;
            else
               normal_z_next = 18'h20000;
         end
     endcase
 end


always @ (posedge clk)
 begin
   if (reset)
      state <= IDLE;
   else if (start)
    begin
      state <= REQ_SHAPE;
      cur_shape <= 0;
      intersected <= 0;
      min_t <= min_dist;
      div_loaded <= 0;
      sqrt_loaded <= 0;
    end
   else
    begin
      if (div_start)
         div_loaded <= 1;

      if (sqrt_start)
         sqrt_loaded <= sqrt_loaded + 1;

      // increment cur_shape after REQ_SHAPE operation
      if (next_state == REQ_SHAPE)
         cur_shape <= cur_shape + 1;
      else if (next_state == CALC_MIN_P2)
         cur_shape <= shape_id;


      state <= next_state;
      normal <= {normal_x_next, normal_y_next, normal_z_next};

      if (sqrt_start)
       begin
         if (sqrt_loaded == 1) //we assume sqrt_loaded <=2 at all times
          begin
            sqrt_sign_buf <= (b < 0 && q2 > 0);
            sqrt_shape_buf <= cur_shape;
            sqrt_b_buf <= b;
          end
         else
          begin
            sqrt_sign <= (b < 0 && q2 > 0);
            sqrt_shape <= cur_shape;
            sqrt_b <= b;
          end
       end
```

```
          if (div_start)
           begin
              div_shape <= cur_shape;
           end

          //get outputs from sqrt and divider
          if (div_done && div_loaded)
           begin
              div_loaded <= 0;
              if(quotient > MIN_T_VAL && quotient < min_t)
               begin
                  intersected <= 1;
                  min_t <= quotient;
                  shape_id <= div_shape;
               end
           end

          if (sqrt_done && sqrt_loaded != 0)
           begin
              if (sqrt_sum < min_t && sqrt_sum > MIN_T_VAL)
               begin
                  min_t <= sqrt_sum;
                  shape_id <= sqrt_shape;
                  intersected <= 1;
               end
              if (sqrt_loaded > 1)
               begin
                  sqrt_sign <= sqrt_sign_buf;
                  sqrt_shape <= sqrt_shape_buf;
                  sqrt_b <= sqrt_b_buf;
               end
              sqrt_loaded <= sqrt_loaded - 1;
           end
        end
       if (set_intersected)
          intersected <= 1;
       {int_pos_x, int_pos_y, int_pos_z} <= {int_pos_x_next, int_pos_y_next,
int_pos_z_next};
       {Vd, V0} <= {Vd_next, V0_next};
       {acc1, acc2} <= {q1, q2};
       b <= b_next;
       {x_sign, y_sign, z_sign} <= {x_sign_next, y_sign_next, z_sign_next};
       comp_disc <= comp_disc_next;
       comp_sign <= comp_sign_next;
    end

endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
///////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
```

```
//
///////////////////////////////////////////////////////////////////////////////

module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
	       ac97_bit_clock,

	       vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
	       vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
	       vga_out_vsync,

	       tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
	       tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
	       tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

	       tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
	       tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
	       tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
	       tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

	       ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
	       ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

	       ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
	       ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

	       clock_feedback_out, clock_feedback_in,

	       flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
	       flash_reset_b, flash_sts, flash_byte_b,

	       rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

	       mouse_clock, mouse_data, keyboard_clock, keyboard_data,

	       clock_27mhz, clock1, clock2,

	       disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
	       disp_reset_b, disp_data_in,

	       button0, button1, button2, button3, button_enter, button_right,
	       button_left, button_down, button_up,

	       switch,

	       led,

	       user1, user2, user3, user4,

	       daughtercard,

	       systemace_data, systemace_address, systemace_ce_b,
	       systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

	       analyzer1_data, analyzer1_clock,
	       analyzer2_data, analyzer2_clock,
	       analyzer3_data, analyzer3_clock,
	       analyzer4_data, analyzer4_clock);

   output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
   input  ac97_bit_clock, ac97_sdata_in;

   output [7:0] vga_out_red, vga_out_green, vga_out_blue;
   output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
```

```
   vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
   tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
   tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
   tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
   tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
   button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
output [6:0]  systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
input  systemace_irq, systemace_mpbrdy;

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
     analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////////////////////////////////////////////
//
```

```
// I/O Assignments
//
////////////////////////////////////////////////////////////////////////////

// Audio Input and Output
assign beep= 1'b0;
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;
// ac97_sdata_in is an input

// VGA Output
//assign vga_out_red = 8'h0;
//assign vga_out_green = 8'h0;
//assign vga_out_blue = 8'h0;
//assign vga_out_sync_b = 1'b1;
//assign vga_out_blank_b = 1'b1;
//assign vga_out_pixel_clock = 1'b0;
//assign vga_out_hsync = 1'b0;
//assign vga_out_vsync = 1'b0;

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;
// tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
// tv_in_aef, tv_in_hff, and tv_in_aff are inputs

// SRAMs
//assign ram0_data = 36'hZ;
//assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
//assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b0;
assign ram0_ce_b = 1'b0;
assign ram0_oe_b = 1'b0;
//assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'h0;
//assign ram1_data = 36'hZ;
//assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
//assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b0;
assign ram1_ce_b = 1'b0;
assign ram1_oe_b = 1'b0;
//assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'h0;
```

```
//assign clock_feedback_out = 1'b0;
// clock_feedback_in is an input

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;
// flash_sts is an input

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;
// rs232_rxd and rs232_cts are inputs

// PS/2 Ports
// mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

// LED Displays
assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
assign disp_data_out = 1'b0;
// disp_data_in is an input

// Buttons, Switches, and Individual LEDs
assign led = 8'hFF;
// button0, button1, button2, button3, button_enter, button_right,
// button_left, button_down, button_up, and switches are inputs

// User I/Os
assign user1 = 32'hZ;
assign user2 = 32'hZ;
assign user3 = 32'hZ;
assign user4 = 32'hZ;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;
// systemace_irq and systemace_mpbrdy are inputs

// Logic Analyzer

assign analyzer1_data = 16'h0;
assign analyzer1_clock = 1'b1;
assign analyzer2_data = 16'h0;
assign analyzer2_clock = 1'b1;
assign analyzer3_data = 16'h0;
assign analyzer3_clock = 1'b1;
assign analyzer4_data = 16'h0;
assign analyzer4_clock = 1'b1;
```

```
//
// Generate a 31.5MHz pixel clock from clock_27mhz
//

wire pclk, pixel_clock;
DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pixel_clock));
// synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
// synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
// synthesis attribute CLK_FEEDBACK of pixel_clock_dcm is "NONE"
// synthesis attribute period of clock_27mhz is "37ns"

//
// VGA output signals
//

wire[9:0] vga_mem_x;
wire[8:0] vga_mem_y;
wire[23:0] vga_mem_color;

wire fpga_clock, locked;
ramclock ramclock (pixel_clock, fpga_clock, ram0_clk, ram1_clk,
        clock_feedback_in, clock_feedback_out, locked);

// Inverting the clock to the DAC provides half a clock period for signals
// to propagate from the FPGA to the DAC.
assign vga_out_pixel_clock = ~fpga_clock;

wire reset;
wire move_type;
wire left, right, up, down, forward, backward, rotate;
debounce reset_debounce (0, fpga_clock, ~button0, reset);
debounce left_debounce(reset, fpga_clock, ~button_left, left);
debounce right_debounce(reset, fpga_clock, ~button_right, right);
debounce up_debounce(reset, fpga_clock, ~button_up, up);
debounce down_debounce(reset, fpga_clock, ~button_down, down);
debounce forward_debounce(reset, fpga_clock, ~button3, forward);
debounce backward_debounce(reset, fpga_clock, ~button2, backward);
debounce type_debounce(reset, fpga_clock, switch[7], rotate);

// Instantiate the module
vga_controller vga (
     .clk(fpga_clock),
     .reset(reset),
     .vga_out_hsync(vga_out_hsync),
     .vga_out_vsync(vga_out_vsync),
     .vga_out_sync_b(vga_out_sync_b),
     .vga_out_blank_b(vga_out_blank_b),
     .vga_color({vga_out_red,vga_out_green,vga_out_blue}),
     .mem_x(vga_mem_x),
     .mem_y(vga_mem_y),
     .mem_color(vga_mem_color)
);

wire [9:0] render_x;
wire [8:0] render_y;
wire [23:0] render_color;
wire flip;

sram_controller sram (
     .clk(fpga_clock),
     .reset(reset),
```

```
        .flip(flip),
        .render_x(render_x),
        .render_y(render_y),
        .render_color(render_color),
        .vga_x(vga_mem_x),
        .vga_y(vga_mem_y),
        .vga_color(vga_mem_color),
        .ram0_data(ram0_data),
        .ram0_address(ram0_address),
        .ram0_we_b(ram0_we_b),
        .ram1_data(ram1_data),
        .ram1_address(ram1_address),
        .ram1_we_b(ram1_we_b)
    );

    wire [3:0] dirs = {left, right, up, down};
    wire [3:0] rotate4 = {rotate, rotate, rotate, rotate};

    master_fsm fsm (
        .clk(fpga_clock),
        .reset(reset),
        .sram_x(render_x),
        .sram_y(render_y),
        .sram_color(render_color),
        .sram_flip(flip),
        .pan(dirs & ~rotate4),
        .rotate(dirs & rotate4),
        .move({forward, backward})
    );

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Lights Module
//
//////////////////////////////////////////////////////////////////////////////////
module lights(n, light_id, light_pos);
    output [3:0] n;
    input [3:0] light_id;
    output reg [53:0] light_pos;

    assign n = 4'd2;

    always @ (light_id)
      case(light_id)
          0: light_pos = {18'h3c000, 18'h1999, 18'h6666};
          //1: light_pos = {18'h3c000, 18'h1999, 18'h6666};
          //0: light_pos = {18'h4000, 18'h1999, 18'h3c000};
          1: light_pos = {18'h4000, 18'h1999, 18'h3c000};
          default: light_pos = 54'h0;
      endcase

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Multiply Accumulator
//
////////////////////////////////////////////////////////////////////////////////
module mac(clk, reset, a, b, q, prod_low);
   input clk;
   input reset;
   input signed [17:0] a;
   input signed [17:0] b;
   output reg signed [17:0] q;
   output [16:0] prod_low;

   wire [35:0] product;   // product of a * b
   assign prod_low = product[16:0];
   reg signed [17:0] sum; // accumulation

   parameter MAX_VAL = 18'h1FFFF;
   parameter MIN_VAL = 18'h20000;

   mult_full multiplier(clk, a, b, product);

    always @ (posedge clk)
     begin
      sum <= reset ? 0 : q;
     end

    always @ (sum or product)
     begin
      q = sum + product[34:17];
      if (!q[17] && sum[17] && product[34])
          q = MIN_VAL;
      else if (q[17] && !sum[17] && !product[34])
          q = MAX_VAL;
     end

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Master Controller
//
////////////////////////////////////////////////////////////////////////////////
module master_fsm(clk, reset, sram_x, sram_y, sram_color, sram_flip, pan, rotate,
move);
    parameter sw = 640;
    parameter sh = 480;
    parameter NUM_RTUS = 14;

    input clk;
    input reset;
    output reg [9:0] sram_x;
    output reg [8:0] sram_y;
    output reg sram_flip;
    reg sram_flip_done;
    output reg[23:0] sram_color;
    input [3:0] pan;
    input [3:0] rotate;
    input [1:0] move;

    reg [53:0] origin, forward_vec, right_vec, up_vec;
    wire [53:0] origin_next, forward_next, right_next, up_next;

    camera camera (
       .clk(clk),
       .origin(origin),
       .forward_vec(forward_vec),
       .right_vec(right_vec),
       .up_vec(up_vec),
       .pan(pan),
       .rotate(rotate),
       .move(move),
       .origin_next(origin_next),
       .forward_next(forward_next),
       .right_next(right_next),
       .up_next(up_next)
    );

    reg [9:0] prenorm_x, norm_x;
    reg [8:0] prenorm_y, norm_y;
    reg [9:0] rtu_x [NUM_RTUS-1:0];
    reg [8:0] rtu_y [NUM_RTUS-1:0];

    reg [NUM_RTUS-1:0] rtu_start;
    wire [NUM_RTUS-1:0] req_color;
    assign req_color = rtu_start;
    wire [107:0] eye_ray;
    wire [23:0] rtu_color;
    wire [NUM_RTUS-1:0] rtu_done;
    wire [NUM_RTUS-1:0] done;
    wire rtu_rfd;

    assign done = reset ? 0 : rtu_done;

    genvar i;
    generate
       for (i = 0; i < NUM_RTUS; i = i +1)
       begin: u
```

```
        rtu_fsm rtu (
            .clk(clk),
            .reset(reset),
            .start(rtu_start[i]),
            .done(rtu_done[i]),
            .req_color(req_color[i]),
            .eye_ray(eye_ray),
            .color(rtu_color),
            .new_frame(sram_flip)
        );
      end
    endgenerate

    wire norm_nd, norm_rdy;
    wire [53:0] norm_vec;


    assign norm_nd = rtu_rfd & ~reset;
    assign rtu_rfd = (rtu_start != 0 || ~norm_rdy); //either we need a new vector or
we're waiting on the normalizer

    // projection and normalizer
    vec_projection vec_proj (
        .clk(clk),
        .reset(reset),
        .ce(norm_nd),
        .nd(norm_nd),
        .prenorm_x(prenorm_x),
        .prenorm_y(prenorm_y),
        .forward_vec(forward_vec),
        .up_vec(up_vec),
        .right_vec(right_vec),
        .norm_vec(norm_vec),
        .rdy(norm_rdy)
    );

    assign eye_ray = {origin, norm_vec};
    reg latch_color_next;
    reg [9:0] sram_x_next;
    reg [8:0] sram_y_next;
    //this should be made nicer
    always @ *
     begin
        rtu_start = 16'b0;
        if (norm_rdy)
         begin
           if (done[0]) rtu_start[0] = 1;
           else if (done[1]) rtu_start[1] = 1;
           else if (done[2]) rtu_start[2] = 1;
           else if (done[3]) rtu_start[3] = 1;
           else if (done[4]) rtu_start[4] = 1;
           else if (done[5]) rtu_start[5] = 1;
           else if (done[6]) rtu_start[6] = 1;
           else if (done[7]) rtu_start[7] = 1;
           else if (done[8]) rtu_start[8] = 1;
           else if (done[9]) rtu_start[9] = 1;
           else if (done[10]) rtu_start[10] = 1;
           else if (done[11]) rtu_start[11] = 1;
           else if (done[12]) rtu_start[12] = 1;
           else if (done[13]) rtu_start[13] = 1;
//         else if (done[14]) rtu_start[14] = 1;
//         else if (done[15]) rtu_start[15] = 1;
           //add more raytracers here
```

```
        end
    end

   always @ (posedge clk)
    begin
       if (reset)
        begin
           {origin} <= {18'h3b334, 18'h9999, 18'hcccc};
           {forward_vec} <= {18'h5555, 18'h35554, 18'h35554}; //{18'haaa9, 18'h2aaaa,
18'h2aaaa};
           {right_vec} <= {18'h1c9f4, 18'h00000, 18'he4f7};
           {up_vec} <= {18'h98a5, 18'h17da0, 18'h2ceb1};
           {prenorm_x, prenorm_y, norm_x, norm_y} <= 0;
        end
       else
        begin
           if (rtu_start[0])
               {rtu_x[0], rtu_y[0], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[0], rtu_y[0], 1'b1};
           else if (rtu_start[1])
               {rtu_x[1], rtu_y[1], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[1], rtu_y[1], 1'b1};
           else if (rtu_start[2])
               {rtu_x[2], rtu_y[2], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[2], rtu_y[2], 1'b1};
           else if (rtu_start[3])
               {rtu_x[3], rtu_y[3], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[3], rtu_y[3], 1'b1};
           else if (rtu_start[4])
               {rtu_x[4], rtu_y[4], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[4], rtu_y[4], 1'b1};
           else if (rtu_start[5])
               {rtu_x[5], rtu_y[5], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[5], rtu_y[5], 1'b1};
           else if (rtu_start[6])
               {rtu_x[6], rtu_y[6], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[6], rtu_y[6], 1'b1};
           else if (rtu_start[7])
               {rtu_x[7], rtu_y[7], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[7], rtu_y[7], 1'b1};
           else if (rtu_start[8])
               {rtu_x[8], rtu_y[8], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[8], rtu_y[8], 1'b1};
           else if (rtu_start[9])
               {rtu_x[9], rtu_y[9], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[9], rtu_y[9], 1'b1};
           else if (rtu_start[10])
               {rtu_x[10], rtu_y[10], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[10], rtu_y[10], 1'b1};
           else if (rtu_start[11])
               {rtu_x[11], rtu_y[11], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[11], rtu_y[11], 1'b1};
           else if (rtu_start[12])
               {rtu_x[12], rtu_y[12], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[12], rtu_y[12], 1'b1};
           else if (rtu_start[13])
               {rtu_x[13], rtu_y[13], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[13], rtu_y[13], 1'b1};
//         else if (rtu_start[14])
//             {rtu_x[14], rtu_y[14], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[14], rtu_y[14], 1'b1};
//         else if (rtu_start[15])
```

```
//           {rtu_x[15], rtu_y[15], sram_x_next, sram_y_next, latch_color_next} <=
{norm_x, norm_y, rtu_x[15], rtu_y[15], 1'b1};
//        //add more raytracers here
          else
             latch_color_next <= 1'b0;
          if (latch_color_next)
           begin
             sram_color <= rtu_color;
             sram_x <= sram_x_next;
             sram_y <= sram_y_next;
           end
          if (rtu_rfd)
           begin
             //gets the next coordinates for the prenormalized vector
             if(prenorm_x == sw - 1)
              begin
                prenorm_x <= 0;
                if(prenorm_y == sh - 1)
                   prenorm_y <= 0;
                else
                   prenorm_y <= prenorm_y + 1;
              end
             else
                prenorm_x <= prenorm_x + 1;

             if (norm_rdy)
              begin
                //gets the next value for the coordinates coming out of the
normalizer
                if(norm_x == sw - 1)
                 begin
                   norm_x <= 0;
                   if(norm_y == sh - 1)
                    begin
                      norm_y <= 0;
                    end
                   else
                      norm_y <= norm_y + 1;
                 end
                else
                   norm_x <= norm_x + 1;
              end
           end

          if (prenorm_x == 0 && prenorm_y == 0)
           begin
             if (sram_flip_done)
                sram_flip <= 0;
             else
              begin
                sram_flip <= 1;
                sram_flip_done <= 1;
              end
           end
          else
           begin
             sram_flip_done <= 0;
             sram_flip <= 0;
           end

          if (sram_flip)
           begin
             origin <= origin_next;
```

```
            forward_vec <= forward_next;
            right_vec <= right_next;
            up_vec <= up_next;
          end
      end
    end

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Shape Materials
//
////////////////////////////////////////////////////////////////////////////////
module materials(shape_id, material_color, material_reflect, material_spec,
material_checker);
    input [3:0] shape_id;
    output [23:0] material_color;
    output [17:0] material_reflect;
    output [4:0] material_spec;
    output [4:0] material_checker;

    reg [51:0] material_data;
    assign {material_color, material_reflect, material_spec, material_checker} =
material_data;

    always @ (shape_id)
      case (shape_id)
         0: material_data = {24'h880000, 18'h10000, 5'h1, 5'h00};
         1: material_data = {24'h008800, 18'h10000, 5'h1, 5'h00};
         2: material_data = {24'h000088, 18'h10000, 5'h1, 5'h00};
         3: material_data = {24'h555555, 18'h10000, 5'h1, 5'b11101};
         //4: material_data = {24'h555555, 18'h10000, 5'h1, 5'h00};
         default: material_data = 52'h0;
      endcase

endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
// Modified to work with higher clock speed
//
///////////////////////////////////////////////////////////////////////////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
///////////////////////////////////////////////////////////////////////////////

module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
            clock_feedback_in, clock_feedback_out, locked);

   input ref_clock;                 // Reference clock input
   output fpga_clock;               // Output clock to drive FPGA logic
   output ram0_clock, ram1_clock;   // Output clocks for each RAM chip
   input  clock_feedback_in;        // Output to feedback trace
   output clock_feedback_out;       // Input from feedback trace
   output locked;                   // Indicates that clock outputs are stable

   wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;

   ////////////////////////////////////////////////////////////////////////////

   BUFG ref_buf (.O(ref_clk), .I(ref_clock));

   BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

   DCM int_dcm (.CLKFB(fpga_clock),
       .CLKIN(ref_clk),
       .RST(dcm_reset),
       .CLK0(fpga_clk),
       .LOCKED(lock1));
   // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of int_dcm is 0
```

```
   BUFG ext_buf (.O(ram_clock), .I(ram_clk));

   IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

   DCM ext_dcm (.CLKFB(fb_clk),
           .CLKIN(ref_clk),
           .RST(dcm_reset),
           .CLK0(ram_clk),
           .LOCKED(lock2));
   // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of ext_dcm is 0

   SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
            .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   // synthesis attribute init of dcm_rst_sr is "000F";


   OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
             .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
             .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock), .C1(~ram_clock),
             .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

   assign locked = lock1 && lock2;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Raytracing Unit (RTU) controller
//
//////////////////////////////////////////////////////////////////////////////////
module rtu_fsm(clk, reset, start, done, req_color, eye_ray, color, new_frame);
    input clk;
    input reset;
    input start;
    output reg done;
    input req_color;
    input [107:0] eye_ray;
    output reg [23:0] color;
    input new_frame;

    reg [23:0] last_color;

    parameter IDLE = 0;
    parameter GET_INT = 1;
    parameter EYE_REFLECT = 2;
    parameter EYE_REFLECT2 = 3;
    parameter EYE_REFLECT3 = 4;
    parameter EYE_SCALE = 5;
    parameter EYE_SCALE2 = 6;
    parameter EYE_SCALE3 = 7;
    parameter EYE_SCALE4 = 8;
    parameter EYE_SCALE5 = 9;
    parameter REQ_LIGHT = 10;
    parameter NORM_LIGHT = 11;
    parameter CHECK_SHADOW = 12;
    parameter SHADE_LIGHT = 13;

    parameter DEPTH = 5;

    reg [3:0] cur_depth;
    reg [3:0] state, next;
    wire [23:0] cur_color;
    reg [107:0] cur_ray, next_ray;
    wire intersected;
    wire [3:0] shape_id;
    wire signed [17:0] int_x, int_y, int_z;
    wire [53:0] int_normal;
    wire signed [17:0] int_norm_x, int_norm_y, int_norm_z;
    assign {int_norm_x, int_norm_y, int_norm_z} = int_normal;

    wire signed [17:0] eye_vect_x, eye_vect_y, eye_vect_z;
    assign {eye_vect_x, eye_vect_y, eye_vect_z} = cur_ray[53:0];

    reg mac_reset;
    reg signed [17:0] a, b;
    wire signed [17:0] q;
    reg signed [17:0] prod;
    wire [16:0] prod_low;
    reg depth_inc;
    mac mac(clk, mac_reset, a, b, q, prod_low);

    reg int_start;
    reg [16:0] min_dist;
    reg stop_on_intersect;
    wire int_done;
```

```
    intersector rtu_intersector (
        .clk(clk),
        .reset(reset),
        .ray(cur_ray),
        .min_dist(min_dist),
        .stop_on_intersect(stop_on_intersect),
        .start(int_start),
        .intersected(intersected),
        .shape_id(shape_id),
        .int_pos({int_x, int_y, int_z}),
        .normal(int_normal),
        .done(int_done),
        .new_frame(new_frame)
    );

    reg signed [17:0] eye_dot_norm, eye_dot_norm_next;
    reg [53:0] eye_reflect;
    reg signed [17:0] eye_reflect_x, eye_reflect_y, eye_reflect_z;
    reg signed [17:0] eye_vect_component;
    wire signed [17:0] eye_reflect_int, eye_reflect_test;
    assign eye_reflect_int = eye_vect_component + prod;
    assign eye_reflect_test = eye_reflect_int + prod;

    wire [53:0] light_vect;
    reg shader_start;
    wire shader_done;
    wire [23:0] shader_color;
    shader rtu_shader (
        .clk(clk),
        .reset(reset),
        .shape_id(shape_id),
        .shape_pos({int_x, int_y, int_z}),
        .shadowed(intersected),
        .light_vect(light_vect),
        .eye_reflect(eye_reflect),
        .normal_vect(int_normal),
        .start(shader_start),
        .done(shader_done),
        .color(shader_color)
    );
   wire [7:0] shader_dep_r = shader_color[23:16] >> cur_depth;
   wire [7:0] shader_dep_g = shader_color[15:8] >> cur_depth;
   wire [7:0] shader_dep_b = shader_color[7:0] >> cur_depth;
   wire [23:0] shader_dep = {shader_dep_r, shader_dep_g, shader_dep_b};

   wire [3:0] lights_n;
   reg [3:0] cur_light;
   wire signed [17:0] light_x, light_y, light_z;
   lights scene_lights (
        .n(lights_n),
        .light_id(cur_light),
        .light_pos({light_x, light_y, light_z})
    );

    wire signed [17:0] prenorm_x,prenorm_y, prenorm_z;
    assign {prenorm_x, prenorm_y, prenorm_z} = {light_x - int_x, light_y - int_y,
light_z - int_z};
    wire [16:0] light_mag;
    reg norm_start;
    wire norm_done;
    vec_norm rtu_vec_norm (
        .clk(clk),
        .vector({prenorm_x, prenorm_y, prenorm_z}),
```

```
        .vect_norm(light_vect),
        .magnitude(light_mag),
        .start(norm_start),
        .done(norm_done)
    );

    reg color_reset;
    reg [23:0] color_add;
    color_acc color_acc(clk, color_reset, color_add[23:16], color_add[15:8],
color_add[7:0], cur_color);


    always @ *
     begin
       next = state;
       a = 0;
       b = 0;
       mac_reset = 0;
       {eye_reflect_x, eye_reflect_y, eye_reflect_z} = eye_reflect;
       eye_dot_norm_next = eye_dot_norm;
       eye_vect_component = eye_vect_x;
       color_reset = 0;
       color_add = 24'h0;
       norm_start = 0;
       shader_start = 0;
       done = 0;
       int_start = 0;
       depth_inc = 0;
       stop_on_intersect = 1'b0;
       min_dist = 17'h1ffff;
       next_ray = cur_ray;
       case(state)
          IDLE:
           begin
             done = 1;
             if (start)
              begin
                int_start = 1;
                next = GET_INT;
                next_ray = eye_ray;
                color_reset = 1;
              end
           end
          GET_INT:
           begin
             if (int_done)
                next = intersected ? EYE_REFLECT : IDLE;
           end
          EYE_REFLECT:
           begin
             mac_reset = 1;
             a = -eye_vect_x;
             b = int_norm_x;
             next = EYE_REFLECT2;
           end
          EYE_REFLECT2:
           begin
             a = -eye_vect_y;
             b = int_norm_y;
             next = EYE_REFLECT3;
           end
          EYE_REFLECT3:
           begin
```

```
      a = -eye_vect_z;
      b = int_norm_z;
      next = EYE_SCALE;
   end
  EYE_SCALE:
   begin
      mac_reset = 1;
      eye_dot_norm_next = q;
      a = eye_dot_norm_next;
      b = int_norm_x;
      next = EYE_SCALE2;
   end
  EYE_SCALE2:
   begin
      mac_reset = 1;
      a = eye_dot_norm;
      b = int_norm_y;
      next = EYE_SCALE3;
   end
  EYE_SCALE3:
   begin
      eye_vect_component = eye_vect_x;
      if (!eye_reflect_test[17] && eye_reflect_int[17] && prod[17])
         eye_reflect_x = 18'h20000;
      else if (eye_reflect_test[17] && !eye_reflect_int[17] && !prod[17])
         eye_reflect_x = 18'h1ffff;
      else
         eye_reflect_x = eye_reflect_test;
      mac_reset = 1;
      a = eye_dot_norm;
      b = int_norm_z;
      next = EYE_SCALE4;
   end
  EYE_SCALE4:
   begin
      eye_vect_component = eye_vect_y;
      if (!eye_reflect_test[17] && eye_reflect_int[17] && prod[17])
         eye_reflect_y = 18'h20000;
      else if (eye_reflect_test[17] && !eye_reflect_int[17] && !prod[17])
         eye_reflect_y = 18'h1ffff;
      else
         eye_reflect_y = eye_reflect_test;
      next = EYE_SCALE5;
   end
  EYE_SCALE5:
   begin
      eye_vect_component = eye_vect_z;
      if (!eye_reflect_test[17] && eye_reflect_int[17] && prod[17])
         eye_reflect_z = 18'h20000;
      else if (eye_reflect_test[17] && !eye_reflect_int[17] && !prod[17])
         eye_reflect_z = 18'h1ffff;
      else
         eye_reflect_z = eye_reflect_test;
      next = REQ_LIGHT;
   end
  REQ_LIGHT:
   begin
      if (cur_light < lights_n)
       begin
         next = NORM_LIGHT;
         norm_start = 1;
       end
      else
```

```
            next = IDLE;
       end
      NORM_LIGHT:
       begin
          stop_on_intersect = 1;
          min_dist = light_mag;
          next_ray = {int_x, int_y, int_z, light_vect};
          if (norm_done)
           begin
              int_start = 1;
              next = CHECK_SHADOW;
           end
       end
      CHECK_SHADOW:
       begin
          stop_on_intersect = 1;
          if (int_done)
           begin
              next = SHADE_LIGHT;
              shader_start = 1;
           end
       end
      SHADE_LIGHT:
          if (shader_done)
           begin
              color_add = shader_dep;
              if (cur_light < lights_n - 1)
                  next = REQ_LIGHT;
              else
               begin
                  if (cur_depth < DEPTH)
                   begin
                      next = GET_INT;
                      depth_inc = 1;
                      next_ray = {int_x, int_y, int_z, eye_reflect};
                      int_start = 1;
                   end
                  else
                      next = IDLE;
               end
           end
    endcase
 end

always @ (posedge clk)
 begin
    if (reset)
     begin
        state <= IDLE;
     end
    else
     begin
        state <= next;
        cur_ray <= next_ray;
        if (next == IDLE)
            last_color <= cur_color;
        if (start)
         begin
            cur_light <= 0;
            cur_depth <= 0;
         end
        else
         begin
```

```
             if (depth_inc)
              begin
                 cur_depth <= cur_depth + 1;
                 cur_light <= 0;
              end
             else if (next == REQ_LIGHT && state == SHADE_LIGHT)
                 cur_light <= cur_light + 1;
          end
       end
      eye_dot_norm <= eye_dot_norm_next;
      eye_reflect <= {eye_reflect_x, eye_reflect_y, eye_reflect_z};
      if (req_color)   //the color is returned one cycle AFTER it is requested
         color <= last_color; //prevents latency issues
      else
         color <= 24'hZZZZZZ;
      prod <= q;
    end

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Shader
//
////////////////////////////////////////////////////////////////////////////////
module shader(clk, reset, shape_id, shape_pos, shadowed, light_vect, eye_reflect,
normal_vect, start, done, color);
    input clk, reset;
    input [3:0] shape_id;
    input [53:0] shape_pos;
    input shadowed;
    input [53:0] light_vect;
    input [53:0] eye_reflect;
    input [53:0] normal_vect;
    input start;
    output done;
    output [23:0] color;

    reg [3:0] state, next;
    parameter NUM_STATES = 14;//7;

    assign done = (state == NUM_STATES) && !start;

    wire signed [17:0] normal_x, normal_y, normal_z;
    assign {normal_x, normal_y, normal_z} = normal_vect;
    wire signed [17:0] light_x, light_y, light_z;
    assign {light_x, light_y, light_z} = light_vect;
    wire signed [17:0] eye_x, eye_y, eye_z;
    assign {eye_x, eye_y, eye_z} = eye_reflect;

    reg [16:0] dot, dot_next;

    reg signed [17:0] a, b;
    wire signed [17:0] q;
    wire [16:0] prod_low;
    reg mac_reset;
    mac mac1(clk, mac_reset, a, b, q, prod_low);

    wire [23:0] material_color;
    wire [7:0] material_red, material_green, material_blue;
    assign {material_red, material_green, material_blue} = material_color;
    wire [17:0] material_reflect;
    wire [4:0] material_spec;
    wire [4:0] material_checker;
    materials materials (shape_id, material_color, material_reflect, material_spec,
material_checker);

    wire x_checker, z_checker, checked;
    assign x_checker = shape_pos[53:36] >> material_checker[3:0];
    assign z_checker = shape_pos[17:0] >> material_checker[3:0];
    assign checked = material_checker[4] && (x_checker ^ z_checker);

    reg color_reset;
    reg [7:0] color_red_add, color_green_add, color_blue_add;
    color_acc color_acc (clk, color_reset, color_red_add, color_green_add,
color_blue_add, color);

    reg [7:0] specular;
    reg signed [17:0] specular_next;
```

```
always @ *
 begin
   a = 0;
   b = 0;
   mac_reset = 0;
   dot_next = dot;
   color_reset = start;
   {color_red_add, color_green_add, color_blue_add} = 24'h0;
   next = state + 1;
   case (state)
      0: begin
         mac_reset = 1;
         a = normal_x;
         b = light_x;
       end
      1: begin
         a = normal_y;
         b = light_y;
       end
      2: begin
         a = normal_z;
         b = light_z;
       end
      3: begin
         mac_reset = 1;
         dot_next = q < 0 || checked ? 17'h0 : q[16:0];
         a = dot_next;
         b = material_red;
       end
      4: begin
         mac_reset = 1;
         color_red_add = q[7:0];
         a = dot;
         b = material_green;
       end
      5: begin
         mac_reset = 1;
         color_green_add = q[7:0];
         a = dot;
         b = material_blue;
       end
      6: begin
         color_blue_add = q[7:0];
         mac_reset = 1;
         a = eye_x;
         b = light_x;
       end
      7: begin
         a = eye_y;
         b = light_y;
       end
      8: begin
         a = eye_z;
         b = light_z;
       end
      9: begin
         mac_reset = 1;
         a = q < 0 ? 18'h0 : q;
         b = a;
       end
      10: begin
         mac_reset = 1;
         a = q;
```

```
            b = a;
          end
         11: begin
            mac_reset = 1;
            a = q;
            b = a;
          end
          12: begin
            mac_reset = 1;
            a = q;
            b = a;
          end
         13: begin
            color_red_add = q[17:10];
            color_green_add = q[17:10];
            color_blue_add = q[17:10];
          end
      endcase
     end

   always @ (posedge clk)
    begin
      if (reset)
         state <= NUM_STATES;
      else
       begin
         if (start)
          begin
            state <= shadowed ? NUM_STATES : 0;
          end
         else
          begin
            if (state < NUM_STATES)
                state <= next;
          end
       end
      dot <= dot_next;
    end

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Shapes
//
////////////////////////////////////////////////////////////////////////////////
module shapes(clk, reset, n, shape_id, shape_type, shape_vector, shape_scalar,
sph_inv_rad, sph_inv_rad_mag, new_frame);

    //plane = 1
    //sphere = 0
    input clk, reset, new_frame;
    output [3:0] n;
    input [3:0] shape_id;
    output shape_type;
    output [53:0] shape_vector;
    output [17:0] shape_scalar;
    output [16:0] sph_inv_rad;
    output [4:0] sph_inv_rad_mag;

    reg [94:0] data;

    assign {shape_type, shape_vector, shape_scalar, sph_inv_rad, sph_inv_rad_mag} =
data;

    assign n = 4'd4;

    reg [94:0] data_v [3:0];

    reg[4:0] count;
    reg dir;

    always @ (shape_id)
      case(shape_id)
          0: data = data_v[0];
          1: data = data_v[1];
          2: data = data_v[2];
          3: data = data_v[3];
          //4: data = data_v[4];
          default: data = 95'h0;
      endcase

    always @ (posedge clk)
    begin
      if (reset)
      begin
          data_v[0] <= {1'b0, 18'h3C000, 18'h01999, 18'h00000, 18'h01999, 17'h14000,
5'h5 };
          data_v[1] <= {1'b0, 18'h00000, 18'h01999, 18'h00000, 18'h01999, 17'h14000,
5'h5 };
          data_v[2] <= {1'b0, 18'h04000, 18'h01999, 18'h00000, 18'h01999, 17'h14000,
5'h5 };
          data_v[3] <= {1'b1, 18'h00000, 18'h1ffff, 18'h00000, 18'h00000, 17'h00000,
5'h0 };
          //data_v[4] <= {1'b1, 18'h00000, 18'h00000, 18'h1ffff, 18'h28000, 17'h00000,
5'h0 };
          dir <= 0;
          count <= 5'b0;
      end
      else
      begin
```

```
          if (new_frame)
          begin
             if (count == 0)
                dir <= ~dir;
             count <= count + 1;
             data_v[0][93:76] <= dir ? data_v[0][93:76] - 18'h00200 : data_v[0][93:76]
+ 18'h00200;
             data_v[2][93:76] <= dir ? data_v[2][93:76] + 18'h00200 : data_v[2][93:76]
- 18'h00200;
             data_v[1][75:58] <= dir ? ((count < 16) ? data_v[1][75:58] + 18'h00200 :
data_v[1][75:58] - 18'h00200) : data_v[1][75:58];
          end
       end
    end


endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Raytracer - SRAM Controller
// Adam Lerer / Sam Gross
////////////////////////////////////////////////////////////////////////////////
module sram_controller(clk, reset, flip, render_x, render_y, render_color, vga_x,
vga_y, vga_color,
              ram0_data, ram0_address, ram0_we_b,
              ram1_data, ram1_address, ram1_we_b); //RAM IO

   parameter x_bits = 10;
   parameter y_bits = 9;
   parameter color_bits = 24;
   parameter width = 640;
   parameter height = 480;

   input reset, clk, flip;
   input [x_bits-1:0] render_x;
   input [y_bits-1:0] render_y;
   input [color_bits-1:0] render_color;
   input [x_bits-1:0] vga_x;
   input [y_bits-1:0] vga_y;
   output [color_bits-1:0] vga_color;
   reg [color_bits-1:0] vga_color;

   inout [35:0] ram0_data, ram1_data;
   output [18:0] ram0_address, ram1_address;
   output ram0_we_b, ram1_we_b;

   wire [18:0] read_address;
   reg [18:0] write_address;

   reg [23:0] write_data_int, write_data_int2, write_data;

   reg read_buffer;

   assign ram0_data = read_buffer ? {12'h0, write_data} : 36'hz;
   assign ram1_data = ~read_buffer ? {12'h0, write_data} : 36'hz;

   assign ram0_address = read_buffer ? write_address : read_address;
   assign ram1_address = ~read_buffer ? write_address : read_address;

   assign ram0_we_b = ~read_buffer ? 1'b1 : 1'b0;
   assign ram1_we_b = read_buffer ? 1'b1 : 1'b0;

   assign read_address = (vga_y << x_bits) + vga_x;

   always @ (posedge clk)
    begin
      if (reset)
         read_buffer <= 1'b0;

      write_data_int <= render_color;
      write_data_int2 <= write_data_int;
      write_data <= write_data_int2;

      if (flip)
         read_buffer <= ~read_buffer;

      write_address <= (render_y << x_bits) + render_x;
      vga_color <= read_buffer ? ram1_data[23:0] : ram0_data[23:0];
```

```
    end
endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Unpipelined module to normalize vectors
//
// BUGS: should check for multiplication overflow - FIXED
//
////////////////////////////////////////////////////////////////////////////////
module vec_norm(clk, vector, vect_norm, magnitude, start, done);
   input clk;
   input [53:0] vector;
   output [53:0] vect_norm;
   output reg [16:0] magnitude;
   input start;
   output reg done;

   reg [16:0] magnitude_next;

   reg sign, sign_next;
   reg signed [17:0] x, y, z;
   reg signed [17:0] x_norm, y_norm, z_norm, x_next, y_next, z_next;
   assign vect_norm = {x_norm, y_norm, z_norm};

   reg signed [17:0] a, b;
   wire signed [35:0] q;
   wire signed [17:0] product;
   assign product = q[34:17];

   mult_full mult (clk, a, b, q);

   reg [16:0] sum, sum_next;
   reg [16:0] normalized;
   wire [16:0] normalized_next;
   wire [4:0] shift;

   fp_norm normalizer (
      .clk(clk),
      .ce(1'b1),
      .in(sum_next),
      .out(normalized_next),
      .shift(shift)
   );

   reg signed [17:0] estimate, estimate_next;

   parameter SQRT_HALF = 18'h16a0a;
   parameter FACTOR = 18'h26b85;

   reg [3:0] state;

   always @ *
    begin
      sum_next = sum;
      estimate_next = estimate;
      a = z;
      b = estimate;
      {x_next, y_next, z_next} = {x_norm, y_norm, z_norm};
      sign_next = sign;
      magnitude_next = magnitude;
      case (state)
         0: begin
```

```
   a = x;
   b = x;
 end
1: begin
   sum_next = product;
   a = y;
   b = y;
 end
2: begin
   sum_next = sum + product;
   a = z;
   b = z;
 end
3: begin
   sum_next = sum + product;
   magnitude_next = sum_next;
   // can do better... ?
 end
4: begin
   estimate_next = FACTOR - (normalized_next >> 1);
   a = estimate_next;
   b = estimate_next;
 end
5: begin
   a = product;
   b = estimate;
 end
6: begin
   a = product;
   b = normalized;
 end
7: begin
   estimate_next = estimate + (estimate >> 1) - product;
   a = estimate_next;
   b = estimate_next;
 end
8: begin
   a = product;
   b = estimate;
 end
9: begin
   a = product;
   b = normalized;
 end
10: begin
   estimate_next = estimate + (estimate >> 1) - product;
   a = SQRT_HALF;
   b = estimate_next;
 end
11: begin
   estimate_next = (shift % 2 == 0 ? product : estimate);
   a = magnitude;
   b = estimate_next;
 end
12: begin
   magnitude_next = q >> (17 - (1 + shift / 2));
   a = x;
   b = estimate;
   sign_next = a[17];
 end
13: begin
   x_next = q >> (17 - (1 + shift / 2));
   if (x_next[17] != sign)
```

```
                if (sign == 0)
                    x_next = 18'h1ffff;
                else
                    x_next = 18'h20000;
            a = y;
            b = estimate;
            sign_next = a[17];
         end
        14: begin
            y_next = q >> (17 - (1 + shift / 2));
            if (y_next[17] != sign)
                if (sign == 0)
                    y_next = 18'h1ffff;
                else
                    y_next = 18'h20000;
            a = z;
            b = estimate;
            sign_next = a[17];
         end
        15: begin
            z_next = q >> (17 - (1 + shift / 2));
            if (z_next[17] != sign)
                if (sign == 0)
                    z_next = 18'h1ffff;
                else
                    z_next = 18'h20000;
         end
      endcase
   end

  always @ (posedge clk)
   begin
      if (start)
       begin
          state <= 0;
          {x, y, z} <= vector;
          sum <= 0;
          estimate <= 0;
          done <= 0;
       end
      else
       begin
          if (state < 15)
              state <= state + 1;
          if (state >= 15)
              done <= 1;
          estimate <= estimate_next;
       end
      sum <= sum_next;
      magnitude <= magnitude_next;
      normalized <= normalized_next;
      {x_norm, y_norm, z_norm} <= {x_next, y_next, z_next};
      sign <= sign_next;
   end
endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Fully pipelined vector normalization
//
// Potential BUG: should check for multiplication overflow - FIXED
//
////////////////////////////////////////////////////////////////////////////////
module vec_norm_pipe(clk, reset, ce, nd, rdy, vector, vect_norm);
   input clk;
    input reset;
    input ce;
    input nd;
    output rdy;
   input [53:0] vector;
   output reg [53:0] vect_norm;

    reg [8:0] valid_data;
    assign rdy = valid_data[8];

    parameter SQRT_HALF = 18'h16a0a;
    parameter FACTOR = 18'h26b85;

    reg signed [17:0] x, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
    reg signed [17:0] y, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9;
    reg signed [17:0] z, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9;

    reg signed [17:0]  mult1_a, mult2_a, mult3_a, mult4_a, mult5_a, mult6_a, mult7_a,
mult8_a, mult9_a, mult10_a, mult11_a, mult12_a, mult13_a;
    reg signed [17:0]  mult1_b, mult2_b, mult3_b, mult4_b, mult5_b, mult6_b, mult7_b,
mult8_b, mult9_b, mult10_b, mult11_b, mult12_b, mult13_b;
    wire signed [35:0] mult1_q, mult2_q, mult3_q, mult4_q, mult5_q, mult6_q, mult7_q,
mult8_q, mult9_q, mult10_q, mult11_q, mult12_q, mult13_q;

    mult_full_ce mult1 (ce, clk, mult1_a, mult1_b, mult1_q);
    mult_full_ce mult2 (ce, clk, mult2_a, mult2_b, mult2_q);
    mult_full_ce mult3 (ce, clk, mult3_a, mult3_b, mult3_q);
    mult_full_ce mult4 (ce, clk, mult4_a, mult4_b, mult4_q);
    mult_full_ce mult5 (ce, clk, mult5_a, mult5_b, mult5_q);
    mult_full_ce mult6 (ce, clk, mult6_a, mult6_b, mult6_q);
    mult_full_ce mult7 (ce, clk, mult7_a, mult7_b, mult7_q);
    mult_full_ce mult8 (ce, clk, mult8_a, mult8_b, mult8_q);
    mult_full_ce mult9 (ce, clk, mult9_a, mult9_b, mult9_q);
    mult_full_ce mult10 (ce, clk, mult10_a, mult10_b, mult10_q);
    mult_full_ce mult11 (ce, clk, mult11_a, mult11_b, mult11_q);
    mult_full_ce mult12 (ce, clk, mult12_a, mult12_b, mult12_q);
    mult_full_ce mult13 (ce, clk, mult13_a, mult13_b, mult13_q);

    reg [16:0] sum_of_squares;
    wire [16:0] normalized_3;
    reg [16:0] normalized_4, normalized_5;

    reg signed [17:0] estimate1, estimate1_4, estimate1_5;
    reg signed [17:0] estimate2, estimate2_6, estimate2_7;
    reg signed [17:0] estimate3, estimate3_8;
    reg signed [17:0] estimate4;

    wire [4:0] shift_3;
    reg [4:0] shift_4, shift_5, shift_6, shift_7, shift_8, shift_9;

    reg signed [35:0] x_shift, y_shift, z_shift;
```

```
reg signed [17:0] x_norm, y_norm, z_norm, x_norm_int, y_norm_int, z_norm_int;

fp_norm normalizer (
    .clk(clk),
    .ce(ce),
    .in(sum_of_squares),
    .out(normalized_3),
    .shift(shift_3)
);

always @ *
 begin
    // --- clk 1 ---
    mult1_a = x;
    mult1_b = x;
    mult2_a = y;
    mult2_b = y;
    mult3_a = z;
    mult3_b = z;
    // --- clk 2 ---
    sum_of_squares = mult1_q[33:17] + mult2_q[33:17] + mult3_q[33:17];
    // --- clk 3 ---
    estimate1 = FACTOR - (normalized_3 >> 1);
    mult4_a = estimate1;
    mult4_b = estimate1;
    mult5_a = estimate1;
    mult5_b = normalized_3;
    // --- clk 4 ---
    mult6_a = mult4_q[34:17];
    mult6_b = mult5_q[34:17];
    // --- clk 5 ---
    estimate2 = estimate1_5 + (estimate1_5 >> 1) - mult6_q[34:17];
    mult7_a = estimate2;
    mult7_b = estimate2;
    mult8_a = estimate2;
    mult8_b = normalized_5;
    // --- clk 6 ---
    mult9_a = mult7_q[34:17];
    mult9_b = mult8_q[34:17];
    // --- clk 7 ---
    estimate3 = estimate2_7 + (estimate2_7 >> 1) - mult9_q[34:17];
    mult10_a = SQRT_HALF;
    mult10_b = estimate3;
    // --- clk 8 ---
    estimate4 = (shift_8 % 2 == 0 ? mult10_q[34:17] : estimate3_8);
    mult11_a = x_8;
    mult11_b = estimate4;
    mult12_a = y_8;
    mult12_b = estimate4;
    mult13_a = z_8;
    mult13_b = estimate4;
    // --- clk 9 ---
    x_shift = mult11_q << (1 + shift_9 / 2);
    y_shift = mult12_q << (1 + shift_9 / 2);
    z_shift = mult13_q << (1 + shift_9 / 2);

    x_norm_int = x_shift[34:17];
    if (x_norm_int[17] != x_9[17])
     begin
       if (x_9[17] == 0)
          x_norm = 18'h1ffff;
       else
          x_norm = 18'h20000;
```

```
         end
        else
           x_norm = x_norm_int;

        y_norm_int = y_shift[34:17];
        if (y_norm_int[17] != y_9[17])
         begin
           if (y_9[17] == 0)
               y_norm = 18'h1ffff;
           else
               y_norm = 18'h20000;
         end
        else
           y_norm = y_norm_int;

        z_norm_int = z_shift[34:17];
        if (z_norm_int[17] != z_9[17])
         begin
           if (z_9[17] == 0)
               z_norm = 18'h1ffff;
           else
               z_norm = 18'h20000;
         end
        else
           z_norm = z_norm_int;

        vect_norm = {x_norm, y_norm, z_norm};
    end

   always @ (posedge clk)
    if (reset)
       valid_data <= 9'h0;
    else if (ce)
     begin
       valid_data[8:0] <= {valid_data[7:0], nd};

       {estimate1_5, estimate1_4} <= {estimate1_4, estimate1};
       {estimate2_7, estimate2_6} <= {estimate2_6, estimate2};
       estimate3_8 <= estimate3;

       {x, y, z} <= vector;
       {x_2, y_2, z_2} <= {x, y, z};
       {x_3, y_3, z_3} <= {x_2, y_2, z_2};
       {x_4, y_4, z_4} <= {x_3, y_3, z_3};
       {x_5, y_5, z_5} <= {x_4, y_4, z_4};
       {x_6, y_6, z_6} <= {x_5, y_5, z_5};
       {x_7, y_7, z_7} <= {x_6, y_6, z_6};
       {x_8, y_8, z_8} <= {x_7, y_7, z_7};
       {x_9, y_9, z_9} <= {x_8, y_8, z_8};

       {normalized_5, normalized_4} <= {normalized_4, normalized_3};
       {shift_9, shift_8, shift_7, shift_6, shift_5, shift_4} <= {shift_8, shift_7,
shift_6, shift_5, shift_4, shift_3};
     end

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Fully pipeline calculation of normalized vector from eye through point on screen
//
////////////////////////////////////////////////////////////////////////////////
module vec_projection(clk, reset, ce, nd, prenorm_x, prenorm_y, forward_vec, up_vec,
right_vec, norm_vec, rdy);
   input clk;
   input reset;
   input ce;
   input nd;
    input [9:0] prenorm_x;
    input [8:0] prenorm_y;
   input [53:0] forward_vec;
   input [53:0] up_vec;
   input [53:0] right_vec;
   output [53:0] norm_vec;
   output rdy;

    wire signed [17:0] forward_vec_x, forward_vec_y, forward_vec_z;
    wire signed [17:0] up_vec_x, up_vec_y, up_vec_z;
    wire signed [17:0] right_vec_x, right_vec_y, right_vec_z;
    assign {forward_vec_x, forward_vec_y, forward_vec_z} = forward_vec;
    assign {up_vec_x, up_vec_y, up_vec_z} = up_vec;
    assign {right_vec_x, right_vec_y, right_vec_z} = right_vec;

    wire signed [18:0] up_x_scaled, up_y_scaled, up_z_scaled;
    wire signed [18:0] rt_x_scaled, rt_y_scaled, rt_z_scaled;

    wire signed [17:0] sam_is_dumb = (18'h6e98 - 18'h76 * prenorm_y);
    wire signed [17:0] adam_is_dumb_too = {18'h76 * prenorm_x - 18'h9375};

    // I'm slow!
    mult18_ce up_scale_x (ce, clk, up_vec_x, sam_is_dumb, up_x_scaled);
    mult18_ce up_scale_y (ce, clk, up_vec_y, sam_is_dumb, up_y_scaled);
    mult18_ce up_scale_z (ce, clk, up_vec_z, sam_is_dumb, up_z_scaled);
    mult18_ce rt_scale_x (ce, clk, right_vec_x, adam_is_dumb_too, rt_x_scaled);
    mult18_ce rt_scale_y (ce, clk, right_vec_y, adam_is_dumb_too, rt_y_scaled);
    mult18_ce rt_scale_z (ce, clk, right_vec_z, adam_is_dumb_too, rt_z_scaled);

    wire [53:0] vect_prenorm;
    reg signed [17:0] prenorm_vec_x, prenorm_vec_y, prenorm_vec_z;
    assign vect_prenorm = {prenorm_vec_x, prenorm_vec_y, prenorm_vec_z};
    wire [53:0] norm_vec;
    reg norm_nd;
    vec_norm_pipe vec_norm (
       .clk(clk),
       .reset(reset),
       .ce(ce),
       .nd(norm_nd),
       .rdy(rdy),
       .vector(vect_prenorm),
       .vect_norm(norm_vec)
     );

    always @ *
     begin
       prenorm_vec_x = forward_vec_x + up_x_scaled[17:0] + rt_x_scaled[17:0];
       prenorm_vec_y = forward_vec_y + up_y_scaled[17:0] + rt_y_scaled[17:0];
       prenorm_vec_z = forward_vec_z + up_z_scaled[17:0] + rt_z_scaled[17:0];
```

```
     end


    always @ (posedge clk)
     begin
        if (reset)
            norm_nd <= 0;
        else
         begin
            if (ce)
                norm_nd <= nd;
         end
     end
endmodule
```