```
//vsim -L E:/Modeltech_6.0b/xilinx/XilinxCoreLib_ver -L
E:/Modeltech_6.0b/xilinx/simprims_ver -L
E:/Modeltech_6.0b/xilinx/unisims_ver work.labkit

//////////////////////////////////////////////////////////////////////
////////
//
// 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)
//
// 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-Apr-29: Change history started
```

```
//
// 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-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-Oct-31: Adapted to new revision 004 board.
//
//////////////////////////////////////////////////////////////////////
////////

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
   //

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


      //Here is where our modules are declared
      wire fpga_clk, reset, pixel_clk;
      assign ram1_clk = ram0_clk;
      wire clock_27mhz_2, delay_in, clock_27mhz_3, ignore;
```

```
      //synthesis attribute period of "clock_27mhz" is 37.04ns;

      IBUF clk27_ibuf(.I(clock_27mhz), .O(clock_27mhz_2));
     BUFG clk27_bufg(.I(clock_27mhz_2),.O(clock_27mhz_3));
     IBUFG delay_ibufg(.I(clock_feedback_in), .O(delay_in));
     clock_generator cgen(clock_27mhz_3, fpga_clk, ram0_clk,
clock_feedback_out,
                          delay_in, reset, ignore, pixel_clk);

      // Logic Analyzer
      wire busy;
      wire rs2 = ~(switch[0] | reset);

      assign analyzer1_data[0] = fpga_clk;
      assign analyzer1_data[1] = ram1_clk;
      assign analyzer1_data[2] = pixel_clk;
      assign analyzer1_data[3] = delay_in;
      assign analyzer1_data[4] = busy;
      assign analyzer1_data[5] = ram0_we_b;
      assign analyzer1_data[15:6] = 0;
      assign analyzer1_clock = clock_27mhz_3;
      assign analyzer2_data = ram0_address[15:0];

      wire [1:0] model_choice = {switch[2],switch[1]};

      wire [1:0] state;
      wire status, render_done, next_frame, frame;
      wire [255:0] matrix;

      assign analyzer4_data[1:0] = state;
      assign analyzer4_data[5] = status;
      assign analyzer4_data[6] = render_done;
      assign analyzer4_data[7] = next_frame;

      ////////////////////////////////////////////////////
      //////////    Synchronizer  ////////////////////////
      ////////////////////////////////////////////////////

      reg res1, res2, reset_sync, up1, up2, up_sync, d1, d2, down_sync,
l1, l2,
          left_sync, r1, r2, right_sync, three_sync, two_sync, three1,
three2, two1, two2;

      always @ (posedge pixel_clk) begin
            reset_sync <= res2;
            res2 <= res1;
            res1 <= rs2;
            up_sync <= up2;
            up2 <= up1;
            up1 <= ~button_up;
            down_sync <= d2;
            d2 <= d1;
            d1 <= ~button_down;
            left_sync <= l2;
            l2 <= l1;
            l1 <= ~button_left;
            right_sync <= r2;
```

```
            r2 <= r1;
            r1 <= ~button_right;
            three_sync <= three2;
            three2 <= three1;
            three1 <= ~button3;
            two_sync <= two2;
            two2 <= two1;
            two1 <= ~button2;
            end

      top_module top(pixel_clk, reset_sync, pixel_clk, ram0_we_b,
ram0_address, ram0_data, ram1_we_b,
                        ram1_address, ram1_data, vga_out_pixel_clock,
vga_out_sync_b,
                        vga_out_blank_b, vga_out_hsync, vga_out_vsync,
vga_out_red, vga_out_green,
                        vga_out_blue, up_sync, down_sync, left_sync,
right_sync, three_sync, two_sync,
                        matrix, model_choice);

      assign ram0_cen_b = 0;
      assign ram1_cen_b = 0;
      assign ram0_oe_b = 0;
      assign ram1_oe_b = 0;
      assign ram0_bwe_b = 4'b0000;
      assign ram1_bwe_b = 4'b0000;
      assign ram0_adv_ld = 0;
      assign ram1_adv_ld = 0;
      assign ram0_ce_b = 0;
      assign ram1_ce_b = 0;

      assign led[0] = ~up_sync;
      assign led[1] = ~down_sync;
      assign led[2] = ~left_sync;
      assign led[3] = ~right_sync;
      assign led[4] = ~three_sync;
      assign led[5] = ~two_sync;


   // Audio Input and Output
   assign beep= 1'b0;
   assign audio_reset_b = 1'b0;
   assign ac97_synch = 1'b0;
   // ac97_sdata_out and ac97_sdata_out are inputs;

   // VGA Output
      //Controlled by our stuff

   // 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
      //These chips are controlled by our stuff


   // 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;
   // disp_data_out 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

  wire [35:0] control;


   icon i_icon
    (
      .control0(control)
    );

  ila i_ila
    (
      .control(control),
      .clk(fpga_clk),
      .trig0(matrix)
    );

endmodule

module icon
  (
     control0
  );
  output [35:0] control0;


endmodule

module ila
  (
    control,
    clk,
    trig0
  );
  input [35:0] control;
  input clk;
  input [255:0] trig0;

endmodule
```

```
module top_module(fpga_clk, reset, pixel_clk, ram0_we_b, ram0_address,
ram0_data, ram1_we_b,
                  ram1_address, ram1_data, vga_out_pixel_clock,
vga_out_sync_b, vga_out_blank_b,
                  vga_out_hsync, vga_out_vsync, vga_out_red,
vga_out_green, vga_out_blue,
                  up, down, left, right, three, two, matrix,
model_choice);

input fpga_clk, reset, pixel_clk;
input up, down, left, right, three, two;
input [1:0] model_choice;
output ram0_we_b, ram1_we_b;
inout [35:0] ram0_data, ram1_data;
output [18:0] ram0_address, ram1_address;
output vga_out_pixel_clock, vga_out_sync_b, vga_out_blank_b,
vga_out_hsync, vga_out_vsync;
output [7:0] vga_out_red, vga_out_green, vga_out_blue;

//////////////////////////////////////////////////
//////////    Button Register   //////////////////
//////////////////////////////////////////////////
reg render_start;

wire out_up, out_down, out_left, out_right, out_three, out_two;

button_register breg(fpga_clk, render_start, up, down, left, right,
three, two,
                     out_up, out_down, out_left, out_right, out_three,
out_two);


//////////////////////////////////////////////////
//////////    ZBT Reset   ////////////////////////
//////////////////////////////////////////////////

reg status, clear_all, reset_rams;
wire clear_rams_busy;

reset_ZBT clear_rams(fpga_clk, reset_rams, clear_rams_busy, status,
                        clear_all,
                  ram0_data,ram0_address,ram0_we_b,
                  ram1_data,ram1_address,ram1_we_b);


//////////////////////////////////////////////////
//////////   Video Controller  ///////////////////
//////////////////////////////////////////////////

reg video_reset, video_start;
wire [18:0] video_ram_addr;
wire [23:0] video_in;
wire next_frame;

video_ctrl vid_control(video_start, video_reset, pixel_clk,
video_ram_addr,
                       ram_we_b, video_in, next_frame, 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);

//////////////////////////////////////////////////
//////////   3D Unit    //////////////////////////
//////////////////////////////////////////////////

reg render_reset;
wire render_done, render_we;
wire [19:0] render_address;
wire [35:0] render_data_in, render_data_out;

output [255:0] matrix;

three_d_unit render_engine(fpga_clk, render_reset, render_start,
                           render_done, render_we, render_address,
                                render_data_in, render_data_out,
                                out_up, out_down, out_left, out_right,
out_three, out_two,
                                model_choice, matrix);

//////////////////////////////////////////////////
//////////   I/O 3D Unit <===> ZBT  //////////////
//////////////////////////////////////////////////
wire render_disable;

IO_render2ram IO_r2r(fpga_clk, render_disable, status,render_data_in,
render_data_out,
                         render_address,render_we,ram0_data,

ram0_address,ram0_we_b,ram1_data,ram1_address,ram1_we_b);


//////////////////////////////////////////////////
//////////   I/O ZBT <===> VIDEO  ////////////////
//////////////////////////////////////////////////

IO_ram2video IO_r2v(fpga_clk, status, video_in, video_ram_addr,
ram_we_b, ram0_data,
                    ram0_address, ram0_we_b, ram1_data, ram1_address,
ram1_we_b);


//////////////////////////////////////////////////
//////////   TOP FSM DECLARATIONS ////////////////
//////////////////////////////////////////////////
reg status_flip;
reg [2:0] state, next;

parameter reset_state = 0;
parameter reset_wait = 1;
parameter render_clear_start = 2;
parameter render_clear_wait = 3;
parameter render_init = 4;
parameter render_main = 5;
```

```
assign render_disable = ~((state == render_init) | (state ==
render_main));

always @ (posedge fpga_clk or negedge reset)
   begin
      if (!reset) begin
          status <= 0;
         state <= reset_state;
            end
      else
      begin
            if(status_flip) status <= ~status;
                  else status <= status;
                  state <= next;
      end
   end

always @ (state or clear_rams_busy or render_done or next_frame)
begin
   reset_rams = 1;
   video_reset = 0;
   render_reset = 0;
   render_start = 0;
   video_start = 0;
   status_flip = 0;
   clear_all = 0;

   case(state)

      reset_state:
      begin
          video_reset = 1;
          reset_rams = 0;
          render_reset = 1;
            clear_all = 1;
          if (clear_rams_busy) next = reset_wait;
          else next = reset_state;
      end

      reset_wait:
         if (!clear_rams_busy)
            begin
                  video_start = 1;
               next = render_init;
                  reset_rams = 0;
            end
         else begin
             clear_all = 1;
            video_reset = 1;
            render_reset = 1;
            next = reset_wait;
         end

        render_clear_start: begin
            reset_rams = 0;
            if (clear_rams_busy) next = render_clear_wait;
```

```
        else next = render_clear_start;
      end

      render_clear_wait: begin
          if (!clear_rams_busy) begin
                next = render_init;
          end
          else begin
                next = render_clear_wait;
          end
      end

     render_init:
        begin
           render_start = 1;
           if(render_done) next = render_init;
             else next = render_main;
       end

     render_main: begin
        if ((render_done) & (next_frame)) begin
              status_flip = 1;
              next = render_clear_start;
        end
        else begin
             next = render_main;
        end
      end



endcase
end

endmodule
```

```
module three_d_unit(clk, reset, start, done, we, address, data_in,
data_out,
                    up, down, left, right, in, out, model_choice,
t_matrix);

    input clk, reset, start;
    input up, down, left, right, in, out;
    input [1:0] model_choice;

    output done, we;
    output [19:0] address;
    input [35:0] data_in;
    output [35:0] data_out;
    output [255:0] t_matrix;

    wire [63:0] row1;
    wire [63:0] row2;
    wire [63:0] row3;
    wire [63:0] row4;

    wire [95:0] face;
    wire [11:0] face_addr;
    wire [11:0] vertex_addr2, normal_addr;
    wire [11:0] vertex_addr = (reset ? 12'h0:vertex_addr2);
    wire [47:0] vertex, normal;

    wire [255:0] p_matrix;
    reg [255:0] t_matrix;

    wire render_done;
    wire [107:0] pixels;
    wire shader_done;
    wire data_ready;

    reg done;
    reg dtr;
    wire done2 = reset | (shader_done & render_done & ~dtr);
    wire [19:0] address;
    wire [71:0] colors;
    wire [31:0] shift;

    // Model Choices
    // model       model_choice:
    // Tetrahedron    0
    // Cube           1
    // Teapot         2
    // Shuttle        3

    model_switcher mdl_sw(clk, model_choice, face_addr, vertex_addr,
                          normal_addr, face, vertex, normal);

    render_unit rnd(clk, reset, start, shader_done, shift, t_matrix,
face_addr,
                    vertex_addr2, normal_addr, face, vertex, normal,
pixels, colors,
                    data_ready, render_done);
    poly_shader sdr(clk, reset, dtr, pixels, colors,
```

```
                we, address, data_in, data_out, shader_done);


    reg [11:0] scale;

    wire [15:0] scale1 = {4'b0, scale};

    always @ (posedge clk) begin
            if  (reset) begin
                  dtr <= 0;
                  done <= 1;
                  scale <= vertex[11:0];
            end
            else begin
                  if(shader_done & dtr) dtr <= 0;
            else if(data_ready) dtr <= 1;
            else dtr <= dtr;
            done <= done2;
                  scale <= scale;
            end

            if (start) begin
                  t_matrix[255:192] <= row1;
                  t_matrix[191:128] <= row2;
                  t_matrix[127:64] <= row3;
                  t_matrix[63:0] <= row4;
            end
            else t_matrix <= t_matrix;

      end


    ////////////////////////////////////////////////////
    ///////// ROTATION MODULE /////////////////////////
    ////////////////////////////////////////////////////

    rotation rot(clk, reset, start, left, right, up, down, in, out,
scale,
                        row1, row2, row3, row4, shift);


endmodule
```

```
module reset_ZBT(clk, reset, busy, status, all,
                 ram0_data, ram0_address, ram0_we_b,
                 ram1_data, ram1_address, ram1_we_b);

   input clk, reset, status, all;
   output  [35:0] ram0_data, ram1_data;
   output [18:0] ram0_address, ram1_address;
   output ram0_we_b,ram1_we_b;
   output busy;

   reg [18:0] ram0_address, ram1_address;
   reg ram0_we_b, ram1_we_b, busy;

   wire [35:0] data_value;
   assign data_value[35:0] = 36'hfff000000;
   assign ram0_data = ((~status | all) && busy) ?
data_value[35:0]:36'bz;
   assign ram1_data = ((status | all) && busy) ?
data_value[35:0]:36'bz;

   ////////////////////////////////////////////////
   ////////  Upon a reset, cycle through addresses /////
   ////////////////////////////////////////////////

   always @ (posedge clk) begin
            if (!reset & ~busy) begin
                  if (~status | all) begin
                        ram0_we_b <= 0;
                  ram0_address <= 0;
                  end
                  if (status | all) begin
                        ram1_we_b <= 0;
                  ram1_address <= 0;
            end
                  busy <= 1;
            end
            else if (busy && ((ram0_address == 19'h78002) ||
                              (ram1_address == 19'h78002)))
                    begin  //1024*480+2=h78002
                  busy <= 0;
                        ram0_address <= 19'bz;
                  ram1_address <= 19'bz;
                  ram0_we_b <= 1'bz;
                  ram1_we_b <= 1'bz;
            end
            else if (busy) begin
                  busy <= 1;
                  if (~status | all) begin
                  ram0_address <= ram0_address + 1;
                  ram0_we_b <= 0;
               end
                  if (status | all) begin
                        ram1_address <= ram1_address + 1;
                        ram1_we_b <= 0;
                  end
            end
            else  begin
```

```
            ram0_we_b <= 1'bz;
            ram0_address <= 19'bz;
            ram1_we_b <= 1'bz;
            ram1_address <= 19'bz;
            busy <= 0;
            end
      end

endmodule
```

```
// version 1.4
// changed for column-wise multiplication

module rotate_x(clk, reset, up, in, out);

    input clk, reset, up;
    input [63:0] in;
    output [63:0] out;

    reg [63:0] out;
    reg [53:0] temp;
    reg [15:0] v1, v2;
    wire [25:0] q1,q2,q3,q4;

    cos_multiply cm1(clk, v1, q1);  // b*cos
    cos_multiply cm2(clk, v2, q2);  // c*cos
    sin_multiply sm1(clk, v1, q3);  // b*sin
    sin_multiply sm2(clk, v2, q4);  // c*sin

    always @ (posedge clk) begin
            if (reset) begin
                  v1 <= in[47:32];
          v2 <= in[31:16];
               out[63:48] <= in[63:48];
          out[47:32] <= in[47:32];
          out[31:16] <= in[31:16];
          out[15:0]  <= in[15:0];
            end
            else begin
                  v1 <= v1;
                  v2 <= v2;

                  if (up) begin
                  temp[53:27] <= q1 + q4; //b*cos + c*sin
                  temp[26:0] <= ~q3 + 1 + q2;  // -b*sin + c*cos
               end
                  else begin
                    temp[53:27] <= q1 + ~q4 + 1;  //b*cos - c*sin
                  temp[26:0] <= q2 + q3; //b*sin + c*cos
                  end

                  out[63:48] <= in[63:48];
                  out[47:32] <= temp[51:36];
                  out[31:16] <= temp[24:9];
                  out[15:0] <=  in[15:0];

            end
      end

      endmodule
```

```
// version 1.4
// changed for column-wise multiplication

module rotate_y(clk, reset, right, in, out);

    input clk, reset, right;
    input [63:0] in;

    output [63:0] out;

    reg [63:0] out;
    reg [53:0] temp;
    reg [15:0] v1, v2;

    wire [25:0] q1,q2,q3,q4;

    cos_multiply cm1(clk, v1, q1);  // a*cos
    cos_multiply cm2(clk, v2, q2);  // c*cos
    sin_multiply sm1(clk, v1, q3);  // a*sin
    sin_multiply sm2(clk, v2, q4);  // c*sin


    always @ (posedge clk) begin
            if (reset) begin
                    v1 <= in[63:48];
            v2 <= in[31:16];
                 out[63:48] <= in[63:48];
                 out[47:32] <= in[47:32];
                 out[31:16] <= in[31:16];
                 out[15:0]  <= in[15:0];
            end

            else begin
                  v1 <= v1;
                  v2 <= v2;

                  if (right) begin
                  temp[53:27] <= q1 + ~q4 + 1; // a*cos - c*sin
                temp[26:0] <= q2 + q3;       // a*sin + c*cos
               end
                  else begin
                  temp[53:27] <= q1 + q4;      // a*cos + c*sin
                  temp[26:0] <= ~q3 + 1 + q2;  // -a*sin + c*cos
               end

                  out[63:48] <= temp[51:36];
                  out[47:32] <= in[47:32];
                  out[31:16] <= temp[24:9];
                  out[15:0] <=  in[15:0];

            end
      end

      endmodule
```

```
module simple_shader(clk, reset, start, pixels, we, addr, data, done);
    input clk, reset, start;
    input [107:0] pixels;
    output we;
    reg we;
    output [19:0] addr;
    inout [35:0] data;
    output done;
    reg done;

    reg [107:0] pxls;

    reg data_out;
    reg [19:0] addr;
    reg [35:0] dt;

    reg [2:0] state;
    parameter IDLE=0;
    parameter DELAY=1;
    parameter ADDR_1 = 2;
    parameter ADDR_2 = 3;
    parameter COMBINED = 4;
    parameter DATA_2 = 5;
    parameter DATA_3 = 6;

    parameter UNDRIVEN = 36'hzzzzzzzzz;

    wire done2 = (state == IDLE && ~start);
    wire we2 = ~(state == ADDR_1 | state == ADDR_2 | state ==
COMBINED);
    assign data = (data_out ? dt:UNDRIVEN);

    parameter XSHIFT = 320;
    parameter YSHIFT = 240;
    parameter WHITE = 24'hfefefe;

    wire [9:0] x1 = (pxls[105:96]+XSHIFT);
    wire [9:0] y1 = (pxls[93:84]+YSHIFT);
    wire [11:0] z1 = pxls[83:72];
    wire [9:0] x2 = (pxls[69:60]+XSHIFT);
    wire [9:0] y2 = (pxls[57:48]+YSHIFT);
    wire [11:0] z2 = pxls[47:36];
    wire [9:0] x3 = (pxls[33:24]+XSHIFT);
    wire [9:0] y3 = (pxls[21:12]+YSHIFT);
    wire [11:0] z3 = pxls[11:0];

    wire [19:0] address_1;
    wire [19:0] address_2;
    wire [19:0] address_3;

    assign address_1[19:10]= y1;
    assign address_1[9:0] = x1;
    assign address_2[19:10]= y2;
    assign address_2[9:0] = x2;
    assign address_3[19:10]= y3;
    assign address_3[9:0] = x3;
```

```
wire [35:0] data_1;
wire [35:0] data_2;
wire [35:0] data_3;

assign data_1[35:24] = z1;
assign data_1[23:0] = WHITE;
assign data_2[35:24] = z2;
assign data_2[23:0] = WHITE;
assign data_3[35:24] = z3;
assign data_3[23:0] = WHITE;

always @ (posedge clk) begin
    if(reset) begin
        state <= IDLE;
        data_out <= 0;
        addr <= 0;
        dt <= 0;
        done <= 1;
    end
    else begin
        we <= we2;
        done <= done2;
        case(state)
           IDLE: begin
               pxls <= 0;
               data_out <= 0;
               addr <= 0;
               dt <= 0;
               if(start) state <= DELAY;
               else state <= IDLE;
           end
           DELAY: begin
               pxls <= pixels;
               dt <= 0;
               data_out <= 0;
               addr <= 0;
               state <= ADDR_1;
           end
           ADDR_1: begin
              pxls <= pxls;
              dt <= 0;
              data_out <= 0;
              addr <= address_1;
              state <= ADDR_2;
           end
           ADDR_2: begin
              pxls <= pxls;
              dt <= 0;
              data_out <= 0;
              addr <= address_2;
              state <= COMBINED;
           end
           COMBINED: begin
              pxls <= pxls;
              dt <= data_1;
              data_out <= 1;
              addr <= address_3;
```

```
                    state <= DATA_2;
                end
                DATA_2: begin
                    pxls <= pxls;
                    dt <= data_2;
                    data_out <= 1;
                    addr <= 0;
                    state <= DATA_3;
                end
                DATA_3: begin
                    pxls <= pxls;
                    dt <= data_3;
                    data_out <= 1;
                    addr <= 0;
                    state <= IDLE;
                end
                default: begin
                    pxls <= pxls;
                    dt <= 0;
                    data_out <= 0;
                    addr <= 0;
                    state <= IDLE;
                end
            endcase
        end
    end
endmodule
```

```
module rotation(clk, reset, start, left, right, up, down, in, out,
scale,
                row1_out, row2_out, row3_out, row4_out, shift);

    input clk, reset, start, left, right, up, down, in, out;
    input [11:0] scale;
    output [31:0] shift;
    output [63:0] row1_out, row2_out, row3_out, row4_out;

    reg [63:0] row1_out, row2_out, row3_out, row4_out,
                 temp1, temp2, temp3, temp4;

    wire [63:0] out_x1, out_x2, out_x3, out_x4,
                out_y1, out_y2, out_y3, out_y4;

    reg [4:0] count;

    wire [63:0] col1 = {row1_out[63:48], row2_out[63:48],
                        row3_out[63:48], row4_out[63:48]};

    wire [63:0] col2 = {row1_out[47:32], row2_out[47:32],
                        row3_out[47:32], row4_out[47:32]};

    wire [63:0] col3 = {row1_out[31:16], row2_out[31:16],
                        row3_out[31:16], row4_out[31:16]};

    wire [63:0] col4 = {row1_out[15:0],  row2_out[15:0],
                        row3_out[15:0],  row4_out[15:0]};

    rotate_x rot_x1(clk, start, up, col1, out_x1);
    rotate_x rot_x2(clk, start, up, col2, out_x2);
    rotate_x rot_x3(clk, start, up, col3, out_x3);
    rotate_x rot_x4(clk, start, up, col4, out_x4);

    rotate_y rot_y1(clk, start, right, col1, out_y1);
    rotate_y rot_y2(clk, start, right, col2, out_y2);
    rotate_y rot_y3(clk, start, right, col3, out_y3);
    rotate_y rot_y4(clk, start, right, col4, out_y4);


    wire [31:0] scale8 = {17'b0, scale, 3'b0};
    wire [31:0] neg_add = 32'hffffff00;
    wire [31:0] pos_add = 32'h00000100;
    reg [31:0] shift;
    reg [31:0] add;

    always @ (posedge clk) begin
       if (reset) begin
                  shift <= {scale8[30:0], 1'b0};
           //temp1 <= 64'h0800000000000000;
           //temp2 <= 64'h0000080000000000;
           //temp3 <= 64'h0000000008000000;
           temp1 <= 64'h0287009bf9d90000;
           temp2 <= 64'h0016073100880000;
           temp3 <= 64'h057cffeb02f90000;
           temp4 <= 64'h0000000000000001;
           row1_out <= temp1;
```

```
                row2_out <= temp2;
                row3_out <= temp3;
               row4_out <= temp4;
              count <= 0;
                  add <= 0;
          end
          else begin
             if (start) begin
                    shift <= (shift + add);
                    add <= 0;
                    row1_out <= temp1;
                   row2_out <= temp2;
                   row3_out <= temp3;
                  row4_out <= temp4;
                 count <= 0;
                 temp1 <= temp1;
                 temp2 <= temp2;
                 temp3 <= temp3;
                 temp4 <= temp4;
           end
              else begin
                 row1_out <= row1_out;
                 row2_out <= row2_out;
                 row3_out <= row3_out;
                 row4_out <= row4_out;
                    shift <= shift;
                    if (count == 20) begin
                       count <= count;
                          if(in || out) begin
                                   if(in) add <= (shift > scale8) ?
neg_add:32'h0;
                                   else if(out) add <= (shift <
32'h7fffffff) ? pos_add:32'h0;
                                   else add <= 32'h0;
                                   temp1 <= temp1;
                                   temp2 <= temp2;
                                   temp3 <= temp3;
                                   temp4 <= temp4;
                             end
                          else if (up || down) begin
                                   add <= 0;
                                temp1 <= {out_x1[63:48], out_x2[63:48],
                                          out_x3[63:48], out_x4[63:48]};
                                temp2 <= {out_x1[47:32], out_x2[47:32],
                                          out_x3[47:32], out_x4[47:32]};
                                temp3 <= {out_x1[31:16], out_x2[31:16],
                                          out_x3[31:16], out_x4[31:16]};
                                temp4 <= {out_x1[15:0],  out_x2[15:0],
                                          out_x3[15:0],  out_x4[15:0]};
                           end
                           else if (left || right) begin
                                    add <= 0;
                                temp1 <= {out_y1[63:48], out_y2[63:48],
                                          out_y3[63:48], out_y4[63:48]};
                                temp2 <= {out_y1[47:32], out_y2[47:32],
                                          out_y3[47:32], out_y4[47:32]};
                                temp3 <= {out_y1[31:16], out_y2[31:16],
```

```
                                        out_y3[31:16], out_y4[31:16]};
                              temp4 <= {out_y1[15:0],  out_y2[15:0],
                                        out_y3[15:0],  out_y4[15:0]};
                        end
                        else begin
                          add <= 0;
                               temp1 <= temp1;
                                temp2 <= temp2;
                                temp3 <= temp3;
                                temp4 <= temp4;
                        end
                end
                else begin
                          shift <= shift;
                          add <= add;
                       count <= count + 1;
                         temp1 <= row1_out;
                          temp2 <= row2_out;
                          temp3 <= row3_out;
                          temp4 <= row4_out;
                    end
                end
          end

        end


endmodule
```

```
module area_mult(clk, reset, start, numerator, reciprocal, result,
done);
    input clk, reset, start;
    input [19:0] numerator;
    input [23:0] reciprocal;
    output done;
    reg done, done2, done3;
    output [39:0] result;

    wire [43:0] r1;
    xilinx_multiplier20x24 m1(.clk(clk), .a(numerator),
                              .b(reciprocal), .q(r1));

    assign result[38:0] = r1[43:5];
    assign result[39] = r1[43];

    always @ (posedge clk) begin
        if(reset) begin
           done <= 0;
           done2 <= 0;
           done3 <= 0;
        end
        else begin
           done <= done2;
           done2 <= done3;
           done3 <= start;
        end
    end
endmodule
```

```
module area_product(clk, reset, start, x, y, xcoeff, ycoeff, result, done);
    input clk, reset, start;
    input [9:0] x, y;
    input [39:0] xcoeff, ycoeff;
    output done;
    output [39:0] result;
    reg done, done2, done3;
    wire [49:0] temp;

    wire [49:0] r1, r2;
    xilinx_multiplier10x40 m1(.clk(clk), .a(x), .b(xcoeff), .q(r1));
    xilinx_multiplier10x40 m2(.clk(clk), .a(y), .b(ycoeff), .q(r2));

    assign temp = (r2 + r1);
    assign result = temp[39:0];

    always @ (posedge clk) begin
        if(reset) begin
           done <= 0;
           done2 <= 0;
           done3 <= 0;
        end
        else begin
           done <= done2;
           done2 <= done3;
           done3 <= start;
        end
    end
endmodule
```

```
module button_register(clk, reset, in_up, in_down, in_left, in_right, in_three,
in_two,
                              out_up, out_down, out_left, out_right, out_three,
out_two);

    input clk, reset, in_up, in_down, in_left, in_right, in_three, in_two;
    output out_up, out_down, out_left, out_right, out_three, out_two;
    reg out_up, out_down, out_left, out_right, out_three, out_two;

    always @ (posedge clk)
       if (reset)
          begin
              out_up <= 0;
              out_down <= 0;
              out_left <= 0;
              out_right <= 0;
                out_three <= 0;
                out_two <= 0;
      end
       else
          begin
          if (in_up && ~(out_down || out_left || out_right))
             out_up <= 1;
             else out_up <= out_up;
          if (in_down && ~(out_up || out_left || out_right))
             out_down <= 1;
             else out_down <= out_down;
          if (in_left && ~(out_up || out_down || out_right))
             out_left <= 1;
             else out_left <= out_left;
          if (in_right && ~(out_down || out_left || out_up))
             out_right <= 1;
             else out_right <= out_right;
            if(in_three & ~in_two) begin
                  out_three <= 1;
                  out_two <= 0;
            end
            else  if(in_two) begin
                  out_three <= 0;
                  out_two <= 1;
            end
            else begin
                  out_three <= 0;
                  out_two <= 0;
            end
         end

endmodule
```

```
module clock_generator(clk27, fpga_clk, mem_clk, delay_out, delay_in,
reset, temp_clk2,
                       pixel_clk);
      input clk27, delay_in;
      output fpga_clk, mem_clk, delay_out, reset, temp_clk2, pixel_clk;

      wire internal_reset;
      //wire pri_locked;
      wire fpga_clk2;
      wire mem_clk2;
      wire fpga_locked;
      wire mem_locked;
      wire shift_out;
      //wire temp_clk;
      wire pixel_clk2;

      DCM pixel_dcm (.CLKIN(clk27),
                  .RST(1'b0),
                  .CLKFX(pixel_clk2));
      // synthesis attribute DLL_FREQUENCY_MODE of pixel_dcm is "LOW"
      // synthesis attribute DUTY_CYCLE_CORRECTION of pixel_dcm is
"TRUE"
      // synthesis attribute STARTUP_WAIT of pixel_dcm is "TRUE"
      // synthesis attribute DFS_FREQUENCY_MODE of pixel_dcm is "LOW"
      // synthesis attribute CLKFX_DIVIDE of pixel_dcm is 30
      // synthesis attribute CLKFX_MULTIPLY of pixel_dcm is 28
      // synthesis attribute CLK_FEEDBACK of pixel_dcm  is "1X"
      // synthesis attribute CLKOUT_PHASE_SHIFT of pixel_dcm is "NONE"
      // synthesis attribute PHASE_SHIFT of pixel_dcm is 0
      // synthesis attribute clkin_period of pixel_dcm is "37.04ns"

      //DCM primary_dcm (.CLKIN(clk27),
      //          .RST(1'b0),
      //          .CLKFX(temp_clk), .LOCKED(pri_locked));
      // synthesis attribute DLL_FREQUENCY_MODE of primary_dcm is "LOW"
      // synthesis attribute DUTY_CYCLE_CORRECTION of primary_dcm is
"TRUE"
      // synthesis attribute STARTUP_WAIT of primary_dcm is "TRUE"
      // synthesis attribute DFS_FREQUENCY_MODE of primary_dcm is "LOW"
      // synthesis attribute CLKFX_DIVIDE of primary_dcm is 27
      // synthesis attribute CLKFX_MULTIPLY of primary_dcm is 30
      // synthesis attribute CLK_FEEDBACK of primary_dcm  is "1X"
      // synthesis attribute CLKOUT_PHASE_SHIFT of primary_dcm is
"NONE"
      // synthesis attribute PHASE_SHIFT of primary_dcm is 0
      // synthesis attribute clkin_period of primary_dcm is "37.04ns"

      DCM fpga_dcm (.CLKIN(temp_clk2), .CLKFB(fpga_clk),
                  .RST(internal_reset),
                  .CLK0(fpga_clk2), .LOCKED(fpga_locked));
      // synthesis attribute DLL_FREQUENCY_MODE of fpga_dcm is "HIGH"
      // synthesis attribute DUTY_CYCLE_CORRECTION of fpga_dcm is
"TRUE"
      // synthesis attribute STARTUP_WAIT of fpga_dcm is "TRUE"
      // synthesis attribute DFS_FREQUENCY_MODE of fpga_dcm is "LOW"
      // synthesis attribute CLK_FEEDBACK of fpga_dcm  is "1X"
      // synthesis attribute CLKOUT_PHASE_SHIFT of fpga_dcm is "NONE"
```

```
        // synthesis attribute clkin_period of fpga_dcm is "33.33ns"

        DCM mem_dcm (.CLKIN(temp_clk2), .CLKFB(delay_in),
                     .RST(internal_reset),
                     .CLK0(mem_clk2), .LOCKED(mem_locked));
        // synthesis attribute DLL_FREQUENCY_MODE of mem_dcm is "HIGH"
        // synthesis attribute DUTY_CYCLE_CORRECTION of mem_dcm is "TRUE"
        // synthesis attribute STARTUP_WAIT of mem_dcm is "TRUE"
        // synthesis attribute DFS_FREQUENCY_MODE of mem_dcm is "LOW"
        // synthesis attribute CLK_FEEDBACK of mem_dcm  is "1X"
        // synthesis attribute CLKOUT_PHASE_SHIFT of mem_dcm is "NONE"
        // synthesis attribute PHASE_SHIFT of mem_dcm is 0
        // synthesis attribute clkin_period of mem_dcm is "33.33ns"

        //BUFG pixel_bufg(.I(pixel_clk2), .O(pixel_clk));
        //BUFG pri_bufg(.I(temp_clk), .O(temp_clk2));
        //assign temp_clk2 = clk27;

        BUFG pixel_bufg(.I(pixel_clk2), .O(temp_clk2));
        BUFG mem_bufg(.I(mem_clk2), .O(mem_clk));
        assign delay_out = mem_clk;
        BUFG fpga_bufg(.I(fpga_clk2), .O(fpga_clk));
        assign pixel_clk = fpga_clk;

        SRL16 startup_delay(.D(1'b0), .CLK(clk27), .A3(1'b1), .A2(1'b1),
                            .A1(1'b1), .A0(1'b1), .Q(shift_out));
        // synthesis attribute INIT of startup_delay is 16'h000f

        assign internal_reset = shift_out; // | (~pri_locked);
        assign reset = internal_reset | (~fpga_locked) | (~mem_locked);

endmodule
```

```
module coeff_mult(clk, reset, start, numerator, reciprocal, result,
done);
    input clk, reset, start;
    input [10:0] numerator;
    input [23:0] reciprocal;
    output done;
    reg done, done2, done3;
    output [39:0] result;

    wire [34:0] r1;
    xilinx_multiplier11x24 m1(.clk(clk), .a(numerator),
                              .b(reciprocal), .q(r1));

    assign result[29:0] = r1[34:5];
    assign result[39:30] = {r1[34],r1[34],r1[34],r1[34],r1[34],
                            r1[34],r1[34],r1[34],r1[34],r1[34]};
    always @ (posedge clk) begin
        if(reset) begin
           done <= 0;
           done2 <= 0;
           done3 <= 0;
        end
        else begin
           done <= done2;
           done2 <= done3;
           done3 <= start;
        end
    end
endmodule
```

```
module color_computer(clk, reset, start, face_color, weight, color,
done);
    input clk, reset, start;
    output done;
    input [23:0] face_color;
    input [31:0] weight;
    output [23:0] color;
    reg done, done2, done3;

    wire [15:0] base = 16'd4096;
    wire [15:0] adj_weight = (weight[31:16] + weight[30:15] + base);
    //wire [15:0] adj_weight = (adj_weight2[15]) ? base:adj_weight2;
    wire [23:0] r1, r2, r3;

    xilinx_multiplier8x16
m1(.clk(clk),.b(adj_weight),.a(face_color[23:16]),.q(r1));
    xilinx_multiplier8x16
m2(.clk(clk),.b(adj_weight),.a(face_color[15:8]),.q(r2));
    xilinx_multiplier8x16
m3(.clk(clk),.b(adj_weight),.a(face_color[7:0]),.q(r3));

    assign color[23:16] = (r1[23:14] > 10'hff) ? 8'hff:r1[21:14];
    assign color[15:8] = (r2[23:14] > 10'hff) ? 8'hff:r2[21:14];
    assign color[7:0] = (r3[23:14] > 10'hff) ? 8'hff:r3[21:14];

    always @ (posedge clk) begin
        if(reset) begin
           done <= 0;
           done2 <= 0;
           done3 <= 0;
        end
        else begin
           if(~done) begin
              done <= done2;
              done2 <= done3;
              done3 <= start;
           end
           else begin
              done <= 0;
              done2 <= 0;
              done3 <= 0;
           end
        end
    end
endmodule
```

```
module color_interpolater (clk, w1, w2, w3, colors, colors_out);
    input clk;
    input [39:0] w1, w2, w3;
    input [71:0] colors;
    output [23:0] colors_out;

    wire [11:0] red1 = {4'b0, colors[71:64]};
    wire [11:0] green1 = {4'b0, colors[63:56]};
    wire [11:0] blue1 = {4'b0, colors[55:48]};
    wire [11:0] red2 = {4'b0, colors[47:40]};
    wire [11:0] green2 = {4'b0, colors[39:32]};
    wire [11:0] blue2 = {4'b0, colors[31:24]};
    wire [11:0] red3 = {4'b0, colors[23:16]};
    wire [11:0] green3 = {4'b0, colors[15:8]};
    wire [11:0] blue3 = {4'b0, colors[7:0]};

    wire [51:0] r1, r2, r3, g1, g2, g3, b1, b2, b3;

    xilinx_multiplier12x40 rm1(.clk(clk), .a(red1), .b(w1), .q(r1));
    xilinx_multiplier12x40 rm2(.clk(clk), .a(red2), .b(w2), .q(r2));
    xilinx_multiplier12x40 rm3(.clk(clk), .a(red3), .b(w3), .q(r3));

    wire [51:0] rtemp = r1 + r2 + r3;
    wire [33:0] rtemp2 = rtemp[51] ? 34'h0:rtemp[51:18];
    assign colors_out[23:16] = (rtemp2 < 34'hff) ? rtemp2[7:0]:8'hff;

    xilinx_multiplier12x40 gm1(.clk(clk), .a(green1), .b(w1), .q(g1));
    xilinx_multiplier12x40 gm2(.clk(clk), .a(green2), .b(w2), .q(g2));
    xilinx_multiplier12x40 gm3(.clk(clk), .a(green3), .b(w3), .q(g3));

    wire [51:0] gtemp = g1 + g2 + g3;
    wire [33:0] gtemp2 = gtemp[51] ? 34'h0:gtemp[51:18];
    assign colors_out[15:8] = (gtemp2 < 34'hff) ? gtemp2[7:0]:8'hff;

    xilinx_multiplier12x40 bm1(.clk(clk), .a(blue1), .b(w1), .q(b1));
    xilinx_multiplier12x40 bm2(.clk(clk), .a(blue2), .b(w2), .q(b2));
    xilinx_multiplier12x40 bm3(.clk(clk), .a(blue3), .b(w3), .q(b3));

    wire [51:0] btemp = b1 + b2 + b3;
    wire [33:0] btemp2 = btemp[51] ? 34'h0:btemp[51:18];
    assign colors_out[7:0] = (btemp2 < 34'hff) ? btemp2[7:0]:8'hff;
endmodule
```

```
//////////////////////////////////////////////////////
//        VIDEO CONTROL MODULE                      //////
//                                                          //////
//   This module takes in rendered data from the    //////
//   ZBT RAMs and outputs them to the video DAC     //////
//   with the appropriate sync/blank signals.       //////
//   This is based off the pset3 model (not an FSM) //////
//////////////////////////////////////////////////////

module video_ctrl (start, reset, clock, ram_addr, ram_we_b,
                       video_in, next_frame,
                       vga_out_red, vga_out_green,
vga_out_blue,vga_out_sync_b,
                       vga_out_blank_b, vga_out_clock, vga_out_hsync,
vga_out_vsync);

   //  clock is used as a pixel and system clock
   input start, reset, clock;
   input [23:0] video_in;

   wire [18:0] ram_addr_temp;
   wire ram_we_b_temp;

   reg [18:0] ram_addr;
   reg ram_we_b;

   output [18:0] ram_addr;
   output ram_we_b;
   output [7:0] vga_out_red, vga_out_green, vga_out_blue;
   // Various syncing and blanking signals follow
   output vga_out_sync_b, vga_out_blank_b;
   output vga_out_clock;
   output vga_out_hsync, vga_out_vsync;
   output next_frame;  // lets system know that its done with frame (to
switch buffers)

   reg [7:0] vga_out_red, vga_out_green, vga_out_blue;

   wire clock;
   wire vga_out_sync_b, vga_out_blank_b;

   // these signals need to be delayed by 2 clocks
   // to account for pipeline delay in video DAC.
   reg hsync1, hsync2, vga_out_hsync, vsync1, vsync2, vga_out_vsync;

   // the counter that keeps track of which pixel is being output to
the
   // DAC is delayed, as same values are used for memory addressing.

   reg [9:0] pixel_count, pixel_count1, pixel_count2, pixel_count3,
pixel_count4;
   reg [9:0] line_count;

   reg next_frame;

     // 640 X 480 60Hz with a 25MHz pixel clock
```

```
   parameter H_ACTIVE = 640;
   parameter H_FRONT_PORCH = 16;
   parameter H_SYNC           =  96;
   parameter H_BACK_PORCH = 48;
   parameter H_TOTAL    =     800;

   parameter V_ACTIVE   = 480;
   parameter V_FRONT_PORCH     =  11;
   parameter V_SYNC           =   2;
   parameter V_BACK_PORCH     =  31;
   parameter V_TOTAL    =      524;

   assign vga_out_clock = clock;

   always @(posedge clock)
      begin
         pixel_count <= pixel_count4;  //  delayed pixel count due to
memory delay
         pixel_count4 <= pixel_count3;
         pixel_count3 <= pixel_count2;
         pixel_count2 <= pixel_count1;
        if (start)
           begin
                 pixel_count1 <= 0;
                 line_count <= 0;
                 next_frame <= 0;
           end
        else if (pixel_count1 == (H_TOTAL-1))
                 begin
                    pixel_count1 <= 0;
                    if (line_count == (V_TOTAL-1))
                       line_count <= 0;
                    else line_count <= line_count + 1;
                 end
             else pixel_count1 <= pixel_count1 +1;
           if ((line_count == V_ACTIVE) & (pixel_count1 == H_ACTIVE))
next_frame <= 1;
         else next_frame <= 0;
       end

      always @ (posedge clock) begin
            ram_addr <= ram_addr_temp;
            ram_we_b <= ram_we_b_temp;
            vga_out_red <= video_in[23:16];
            vga_out_green <= video_in[15:8];
            vga_out_blue <= video_in[7:0];
            end

      assign ram_addr_temp[18:10] = (reset) ? 9'bzzzzzzzzz :
line_count;  // Memory addressing
      assign ram_addr_temp[9:0] = (reset) ? 10'bzzzzzzzzzz :
pixel_count1;
      assign ram_we_b_temp = reset ? 1'bz : 1;

     assign vga_out_sync_b = hsync1 ^ vsync1;
     assign vga_out_blank_b = ((pixel_count<H_ACTIVE) &
(line_count<V_ACTIVE));
```

```
      always @ (posedge clock) begin
          if (start)
            begin
               hsync1 <= 1;
               hsync2 <= 1;
               vga_out_hsync <= 1;
               vsync1 <= 1;
               vsync2 <= 1;
               vga_out_vsync <= 1;
            end
          else
           begin
            if (pixel_count == (H_ACTIVE+H_FRONT_PORCH))
              hsync1 <= 0;
            else if (pixel_count == (H_ACTIVE+H_FRONT_PORCH+H_SYNC))
              hsync1 <= 1;
            if (pixel_count == (H_TOTAL-1))
              begin
               if (line_count == (V_ACTIVE+V_FRONT_PORCH))
                 vsync1 <= 0;
               else if (line_count == (V_ACTIVE+V_FRONT_PORCH+V_SYNC))
                 vsync1 <= 1;
              end

         end

       hsync2 <= hsync1;
       vga_out_hsync <= hsync2;
       vsync2 <= vsync1;
       vga_out_vsync <= vsync2;

      end



endmodule
```

```
module cross_product(clk, reset, start, x1, y1, x2, y2, result, done);
    input clk, reset, start;
    input [10:0] x1, y1, x2, y2;
    output done;
    reg done, done2, done3;
    output [21:0] result;

    wire [21:0] r1, r2;
    xilinx_multiplier11x11 m1(.clk(clk), .a(x1), .b(y2), .q(r1));
    xilinx_multiplier11x11 m2(.clk(clk), .a(x2), .b(y1), .q(r2));

    assign result = (r1 +1 + ~r2);

    always @ (posedge clk) begin
        if(reset) begin
           done <= 0;
           done2 <= 0;
           done3 <= 0;
        end
        else begin
           done <= done2;
           done2 <= done3;
           done3 <= start;
        end
    end
endmodule
```

```
module cross_product_unsigned(clk, reset, start, x1, y1, x2, y2,
result, done);
    input clk, reset, start;
    input [9:0] x1, y1, x2, y2;
    output done;
    reg done, done2, done3;
    output [19:0] result;

    wire [19:0] r1, r2;
    xilinx_multiplier10x10unsigned m1(.clk(clk), .a(x1), .b(y2),
.q(r1));
    xilinx_multiplier10x10unsigned m2(.clk(clk), .a(x2), .b(y1),
.q(r2));

    assign result = (r1 +1 + ~r2);

    always @ (posedge clk) begin
        if(reset) begin
           done <= 0;
           done2 <= 0;
           done3 <= 0;
        end
        else begin
           done <= done2;
           done2 <= done3;
           done3 <= start;
        end
    end
endmodule
```

```
module depth_computer(clk, w1, w2, w3, z1, z2, z3, zmax, zmin, z);
    input clk;
    input [39:0] w1, w2, w3;
    input [11:0] z1, z2, z3, zmax, zmin;
    output [11:0] z;
    wire [51:0] r1, r2, r3;

    xilinx_multiplier12x40 wz1(.clk(clk), .a(z1), .b(w1), .q(r1));
    xilinx_multiplier12x40 wz2(.clk(clk), .a(z2), .b(w2), .q(r2));
    xilinx_multiplier12x40 wz3(.clk(clk), .a(z3), .b(w3), .q(r3));

    wire [51:0] temp = r1 + r2 + r3;
    wire [11:0] ztemp = temp[17] ? temp[29:18]:(temp[29:18]+1);
    wire [11:0] ztemp2 = (ztemp > zmax) ? zmax:ztemp;
    assign z[11:0] = (ztemp < zmin) ? zmin:ztemp2;

endmodule
```

```
module dot_product(clk, start, v1, v2, shift, result, done);
    input clk, start;
    input [47:0] v1;
    input [47:0] v2;
    input [15:0] shift;
    output [31:0] result;
    reg [31:0] result;
    output done;

    reg [47:0] vin_1;
    reg [47:0] vin_2;

    wire [31:0] mul1;
    wire [31:0] mul2;
    wire [31:0] mul3;

    reg [31:0] s1;
    reg done;

    parameter LATENCY = 2;
    reg [3:0] counter;

    xilinx_multiplier xm3(clk, vin_1[15:0], vin_2[15:0], mul3);
    xilinx_multiplier xm2(clk, vin_1[31:16], vin_2[31:16], mul2);
    xilinx_multiplier xm1(clk, vin_1[47:32], vin_2[47:32], mul1);

    always @ (posedge clk) begin
        if(start) begin
            result <= 0;
            done <= 0;
            counter <= 0;
            vin_1 <= v1;
            vin_2 <= v2;
            s1[15:0] <= shift[15:0];
                  s1[31:16] <=
{shift[15],shift[15],shift[15],shift[15],shift[15],shift[15],shift[15],
shift[15],

      shift[15],shift[15],shift[15],shift[15],shift[15],shift[15],shift
[15],shift[15]};
        end
        else if(counter > LATENCY) begin
            result <= result;
            done <= 1;
            counter <= counter;
            vin_1 <= vin_1;
            vin_2 <= vin_2;
            s1 <= s1;
        end
        else if(counter < LATENCY) begin
            result <= result;
            done <= 0;
            counter <= counter + 1;
            vin_1 <= vin_1;
            vin_2 <= vin_2;
            s1 <= s1;
        end
```

```
        else begin
            result <= (mul1+mul2+mul3+s1);
            done <= 0;
            counter <= counter + 1;
            vin_1 <= vin_1;
            vin_2 <= vin_2;
            s1 <= s1;
        end
    end
endmodule
```

```
module face_renderer(clk, reset, start, shift, t_matrix, vertices,
normals,
                     face_color, vertex_address, normal_address,
                     vertex, normal, pixels, colors, done);
    input clk, start, reset;
    input [35:0] vertices, normals;
    input [47:0] vertex, normal;
    input [23:0] face_color;
    input [255:0] t_matrix;
    input [31:0] shift;
    output done;

    output [107:0] pixels;
    reg [107:0] pixels;
    output [71:0] colors;
    reg [71:0] colors;

    output [11:0] vertex_address, normal_address;
    reg [11:0] vertex_address, normal_address;

    reg [3:0] state;
    parameter IDLE = 0;
    parameter WAIT_0 = 6;
    parameter COMPUTE_1 = 1;
    parameter WAIT_1 = 2;
    parameter COMPUTE_2 = 3;
    parameter WAIT_2 = 4;
    parameter COMPUTE_3 = 5;

    assign done = (state == IDLE);

    reg rnd_start;
    reg rnd_start2, rnd_start3;
    wire r_done;
    wire [35:0] rnd_out;

    renderer rnd(clk, reset, rnd_start3, vertex, shift,
t_matrix,rnd_out, r_done);

    wire colors_done;
    wire [23:0] colors_out;
    lighting light_unit(clk, reset, rnd_start3, t_matrix, vertex,
normal,
                        face_color, colors_out, colors_done);

    wire rnd_done = r_done & colors_done;

    always @ (posedge clk) begin
        if(reset) begin
            state <= IDLE;
            rnd_start <= 0;
            pixels <= 0;
            colors <= 0;
            vertex_address <= 0;
            normal_address <= 0;
            rnd_start3 <= 0;
            rnd_start2 <= 0;
```

```
end
else begin
case (state)
    IDLE: begin
        if(start) begin
            vertex_address <= vertices[35:24];
            normal_address <= normals[35:24];
            pixels <= 0;
            colors <= 0;
            state <= WAIT_0;
            rnd_start <= 1;
            rnd_start3 <= 1;
            rnd_start2 <= 1;
        end
        else begin
            vertex_address <= 0;
            normal_address <= 0;
            pixels <= pixels;
            colors <= colors;
            state <= IDLE;
            rnd_start <= 0;
            rnd_start3 <= 0;
            rnd_start2 <= 0;
        end
    end
    WAIT_0: begin
        vertex_address <= vertices[35:24];
        normal_address <= normals[35:24];
        pixels <= 0;
        colors <= 0;
        rnd_start <= 1;
        rnd_start3 <= 1;
        rnd_start2 <= 1;
        if(rnd_done) state <= WAIT_0;
        else state <= COMPUTE_1;
    end
    COMPUTE_1: begin
        if(rnd_done) begin
            vertex_address <= vertices[23:12];
            normal_address <= normals[23:12];
            rnd_start <= 1;
            pixels[107:72] <= rnd_out;
            pixels[71:0] <= 0;
            colors[71:48] <= colors_out;
            colors[47:0] <= 0;
            state <= WAIT_1;
            rnd_start <= 1;
            rnd_start3 <= 1;
            rnd_start2 <= 1;
        end
        else begin
            vertex_address <= vertex_address;
            normal_address <= normal_address;
            colors <= colors;
            rnd_start <= 0;
            pixels <= pixels;
            state <= state;
```

```
            rnd_start3 <= rnd_start2;
            rnd_start2 <= rnd_start;
            rnd_start <= 0;
        end
    end
    WAIT_1: begin
        vertex_address <= vertex_address;
        normal_address <= normal_address;
        colors <= colors;
        rnd_start3 <= rnd_start2;
        rnd_start2 <= rnd_start;
        rnd_start <= 0;
        pixels <= pixels;
        if(rnd_done) state <= state;
        else state <= COMPUTE_2;
    end
    COMPUTE_2: begin
        if(rnd_done) begin
            vertex_address <= vertices[11:0];
            normal_address <= normals[11:0];
            rnd_start <= 1;
            pixels[107:72] <= pixels[107:72];
            pixels[71:36] <= rnd_out;
            pixels[35:0] <= 0;
            colors <= {colors[71:48], colors_out, 24'h0};
            state <= WAIT_2;
            rnd_start <= 1;
            rnd_start3 <= 1;
            rnd_start2 <= 1;
        end
        else begin
            vertex_address <= vertex_address;
            normal_address <= normal_address;
            colors <= colors;
            rnd_start <= 0;
            pixels <= pixels;
            state <= state;
            rnd_start3 <= rnd_start2;
            rnd_start2 <= rnd_start;
            rnd_start <= 0;
        end
    end
    WAIT_2: begin
        vertex_address <= vertex_address;
        normal_address <= normal_address;
        colors <= colors;
        rnd_start3 <= rnd_start2;
        rnd_start2 <= rnd_start;
        rnd_start <= 0;
        pixels <= pixels;
        if(rnd_done) state <= state;
        else state <= COMPUTE_3;
    end
    COMPUTE_3: begin
        vertex_address <= vertex_address;
        normal_address <= normal_address;
        if(rnd_done) begin
```

```
                    rnd_start <= 0;
                    pixels[107:36] <= pixels[107:36];
                    pixels[35:0] <= rnd_out;
                    colors <= {colors[71:24], colors_out};
                    state <= IDLE;
                    rnd_start3 <= 0;
                    rnd_start2 <= 0;
                end
                else begin
                    colors <= colors;
                    pixels <= pixels;
                    state <= state;
                    rnd_start3 <= rnd_start2;
                    rnd_start2 <= rnd_start;
                    rnd_start <= 0;
                end
            end
            default: begin
                colors <= colors;
                pixels <= pixels;
                state <= IDLE;
                rnd_start3 <= rnd_start2;
                rnd_start2 <= rnd_start;
                rnd_start <= 0;
                vertex_address <= vertex_address;
                normal_address <= normal_address;
            end
        endcase
        end
    end
endmodule
```

```
module gouraud_computer(clk, reset, start, xloc, yloc, x1, y1, x2, y2,
x3, y3,
                        v1x, v1y, v2x, v2y, v3x, v3y, w1, w1x, w1y,
                        w2, w2x, w2y, w3, w3x, w3y, done);
    input clk, reset, start;
    input [9:0] x1, y1, x2, y2, x3, y3, xloc, yloc;
    input [10:0] v1x, v1y, v2x, v2y, v3x, v3y;
    output [39:0] w1, w1x, w1y, w2, w2x, w2y, w3, w3x, w3y;
    reg [39:0] w1, w1x, w1y, w2, w2x, w2y, w3, w3x, w3y;
    output done;
    reg done;

    reg [2:0] state;
    parameter IDLE = 0;
    parameter COMPUTE_AREA = 1;
    parameter START_DIVIDE = 2;
    parameter STORE_DIVIDE = 3;
    parameter COMPUTE_COEFFS = 4;
    parameter COMPUTE_INITIALS = 5;

    wire done2 = (state == IDLE);

    wire area_start = (state == COMPUTE_AREA);
    wire a_done, o1_done, o2_done, o3_done;
    wire [21:0] area_temp;
    reg [21:0] area;
    wire area_done = a_done & o1_done & o2_done & o3_done;
    wire [19:0] offset1, offset2, offset3;

    cross_product area_computation(clk, reset, area_start,
                                   v1x, v1y, v2x, v2y,
                                   area_temp, a_done);

    cross_product_unsigned o1(clk, reset, area_start, x2, y2, x3, y3,
offset1, o1_done);
    cross_product_unsigned o2(clk, reset, area_start, x3, y3, x1, y1,
offset2, o2_done);
    cross_product_unsigned o3(clk, reset, area_start, x1, y1, x2, y2,
offset3, o3_done);

    wire [31:0] quotient = 32'h00800000;
    wire [31:0] divisor;
    assign divisor[31:22] =
{area[21],area[21],area[21],area[21],area[21],area[21],

area[21],area[21],area[21],area[21],area[21],area[21]};
    assign divisor[21:0] = area;

    wire divide_start = (state == START_DIVIDE);
    wire divide_done;

    wire [31:0] divide_result;
    reg [23:0] reciprocal;

    wrapped_divider dvdr(clk, divide_start, quotient, divisor,
                         divide_result, divide_done);
```

```
    wire cstart = (state == COMPUTE_COEFFS);
    wire [39:0] w1x_temp, w1y_temp, w2x_temp, w2y_temp, w3x_temp,
w3y_temp;
    wire w1x_done, w1y_done, w2x_done, w2y_done, w3x_done, w3y_done;
    wire off1_done, off2_done, off3_done;
    wire [39:0] off1, off2, off3;
    wire cdone = w1x_done & w1y_done & w2x_done & w2y_done &
                 w3x_done & w3y_done & off1_done & off2_done &
off3_done;

    coeff_mult m1x(clk, reset, cstart, v2y, reciprocal, w1x_temp,
w1x_done);
    coeff_mult m1y(clk, reset, cstart, v2x, reciprocal, w1y_temp,
w1y_done);
    coeff_mult m2x(clk, reset, cstart, v3y, reciprocal, w2x_temp,
w2x_done);
    coeff_mult m2y(clk, reset, cstart, v3x, reciprocal, w2y_temp,
w2y_done);
    coeff_mult m3x(clk, reset, cstart, v1y, reciprocal, w3x_temp,
w3x_done);
    coeff_mult m3y(clk, reset, cstart, v1x, reciprocal, w3y_temp,
w3y_done);

    area_mult a1(clk, reset, cstart, offset1, reciprocal, off1,
off1_done);
    area_mult a2(clk, reset, cstart, offset2, reciprocal, off2,
off2_done);
    area_mult a3(clk, reset, cstart, offset3, reciprocal, off3,
off3_done);

    wire vstart = (state == COMPUTE_INITIALS);
    wire [39:0] w1_temp, w2_temp, w3_temp;

    wire w1_done, w2_done, w3_done;
    wire vdone = w1_done & w2_done & w3_done;

    area_product c1(clk, reset, vstart, xloc, yloc, w1x, w1y, w1_temp,
w1_done);
    area_product c2(clk, reset, vstart, xloc, yloc, w2x, w2y, w2_temp,
w2_done);
    area_product c3(clk, reset, vstart, xloc, yloc, w3x, w3y, w3_temp,
w3_done);


    always @ (posedge clk) begin
       if(reset) begin
           w1 <= 0; w1x <= 0; w1y <= 0;
           w2 <= 0; w2x <= 0; w2y <= 0;
           w3 <= 0; w3x <= 0; w3y <= 0;
           area <= 0;
           reciprocal <= 0;
           done <= 0;
           state <= IDLE;
       end
       else begin
          done <= done2;
          case (state)
```

```
IDLE: begin
   w1 <= w1; w1x <= w1x; w1y <= w1y;
   w2 <= w2; w2x <= w2x; w2y <= w2y;
   w3 <= w3; w3x <= w3x; w3y <= w3y;
   area <= area;
   reciprocal <= reciprocal;
   if(start) state <= COMPUTE_AREA;
   else state <= IDLE;
end
COMPUTE_AREA: begin
   w1 <= w1; w1x <= w1x; w1y <= w1y;
   w2 <= w2; w2x <= w2x; w2y <= w2y;
   w3 <= w3; w3x <= w3x; w3y <= w3y;
   reciprocal <= reciprocal;
   if(area_done) begin
      area <= area_temp;
      state <= START_DIVIDE;
   end
   else begin
      area <= area;
      state <= COMPUTE_AREA;
   end
end
START_DIVIDE: begin
   w1 <= w1; w1x <= w1x; w1y <= w1y;
   w2 <= w2; w2x <= w2x; w2y <= w2y;
   w3 <= w3; w3x <= w3x; w3y <= w3y;
   area <= area;
   reciprocal <= 0;
   if(divide_done) begin
       state <= START_DIVIDE;
   end
   else begin
       state <= STORE_DIVIDE;
   end
end
STORE_DIVIDE: begin
   w1 <= w1; w1x <= w1x; w1y <= w1y;
   w2 <= w2; w2x <= w2x; w2y <= w2y;
   w3 <= w3; w3x <= w3x; w3y <= w3y;
   area <= area;
   if(divide_done) begin
       reciprocal <= divide_result[23:0];
       state <= COMPUTE_COEFFS;
   end
   else begin
       reciprocal <= reciprocal;
       state <= STORE_DIVIDE;
   end
end
COMPUTE_COEFFS: begin
    w1 <= w1; w2 <= w2; w3 <= w3;
    area <= area;
    reciprocal <= reciprocal;
    if(cdone) begin
       w1x <= ~w1x_temp+1; w1y <= w1y_temp;
       w2x <= ~w2x_temp+1; w2y <= w2y_temp;
```

```
//vsim -L E:/Modeltech_6.0b/xilinx/XilinxCoreLib_ver -L
E:/Modeltech_6.0b/xilinx/simprims_ver -L
E:/Modeltech_6.0b/xilinx/unisims_ver work.labkit

//////////////////////////////////////////////////////////////////////
////////
//
// 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)
//
// 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-Apr-29: Change history started
```

```
//
// 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-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-Oct-31: Adapted to new revision 004 board.
//
//////////////////////////////////////////////////////////////////////
////////

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
   //

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


      //Here is where our modules are declared
      wire fpga_clk, reset, pixel_clk;
      assign ram1_clk = ram0_clk;
      wire clock_27mhz_2, delay_in, clock_27mhz_3, ignore;
```

```
      //synthesis attribute period of "clock_27mhz" is 37.04ns;

      IBUF clk27_ibuf(.I(clock_27mhz), .O(clock_27mhz_2));
     BUFG clk27_bufg(.I(clock_27mhz_2),.O(clock_27mhz_3));
     IBUFG delay_ibufg(.I(clock_feedback_in), .O(delay_in));
     clock_generator cgen(clock_27mhz_3, fpga_clk, ram0_clk,
clock_feedback_out,
                          delay_in, reset, ignore, pixel_clk);

      // Logic Analyzer
      wire busy;
      wire rs2 = ~(switch[0] | reset);

      assign analyzer1_data[0] = fpga_clk;
      assign analyzer1_data[1] = ram1_clk;
      assign analyzer1_data[2] = pixel_clk;
      assign analyzer1_data[3] = delay_in;
      assign analyzer1_data[4] = busy;
      assign analyzer1_data[5] = ram0_we_b;
      assign analyzer1_data[15:6] = 0;
      assign analyzer1_clock = clock_27mhz_3;
      assign analyzer2_data = ram0_address[15:0];

      wire [1:0] model_choice = {switch[2],switch[1]};

      wire [1:0] state;
      wire status, render_done, next_frame, frame;
      wire [255:0] matrix;

      assign analyzer4_data[1:0] = state;
      assign analyzer4_data[5] = status;
      assign analyzer4_data[6] = render_done;
      assign analyzer4_data[7] = next_frame;

      ////////////////////////////////////////////////////
      //////////    Synchronizer  ////////////////////////
      ////////////////////////////////////////////////////

      reg res1, res2, reset_sync, up1, up2, up_sync, d1, d2, down_sync,
l1, l2,
          left_sync, r1, r2, right_sync, three_sync, two_sync, three1,
three2, two1, two2;

      always @ (posedge pixel_clk) begin
            reset_sync <= res2;
            res2 <= res1;
            res1 <= rs2;
            up_sync <= up2;
            up2 <= up1;
            up1 <= ~button_up;
            down_sync <= d2;
            d2 <= d1;
            d1 <= ~button_down;
            left_sync <= l2;
            l2 <= l1;
            l1 <= ~button_left;
            right_sync <= r2;
```

```
            r2 <= r1;
            r1 <= ~button_right;
            three_sync <= three2;
            three2 <= three1;
            three1 <= ~button3;
            two_sync <= two2;
            two2 <= two1;
            two1 <= ~button2;
            end

      top_module top(pixel_clk, reset_sync, pixel_clk, ram0_we_b,
ram0_address, ram0_data, ram1_we_b,
                        ram1_address, ram1_data, vga_out_pixel_clock,
vga_out_sync_b,
                        vga_out_blank_b, vga_out_hsync, vga_out_vsync,
vga_out_red, vga_out_green,
                        vga_out_blue, up_sync, down_sync, left_sync,
right_sync, three_sync, two_sync,
                        matrix, model_choice);

      assign ram0_cen_b = 0;
      assign ram1_cen_b = 0;
      assign ram0_oe_b = 0;
      assign ram1_oe_b = 0;
      assign ram0_bwe_b = 4'b0000;
      assign ram1_bwe_b = 4'b0000;
      assign ram0_adv_ld = 0;
      assign ram1_adv_ld = 0;
      assign ram0_ce_b = 0;
      assign ram1_ce_b = 0;

      assign led[0] = ~up_sync;
      assign led[1] = ~down_sync;
      assign led[2] = ~left_sync;
      assign led[3] = ~right_sync;
      assign led[4] = ~three_sync;
      assign led[5] = ~two_sync;


   // Audio Input and Output
   assign beep= 1'b0;
   assign audio_reset_b = 1'b0;
   assign ac97_synch = 1'b0;
   // ac97_sdata_out and ac97_sdata_out are inputs;

   // VGA Output
      //Controlled by our stuff

   // 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
      //These chips are controlled by our stuff


   // 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;
   // disp_data_out 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

  wire [35:0] control;


   icon i_icon
    (
      .control0(control)
    );

  ila i_ila
    (
      .control(control),
      .clk(fpga_clk),
      .trig0(matrix)
    );

endmodule

module icon
  (
     control0
  );
  output [35:0] control0;


endmodule

module ila
  (
    control,
    clk,
    trig0
  );
  input [35:0] control;
  input clk;
  input [255:0] trig0;

endmodule
```

```
                    w3x <= ~w3x_temp+1; w3y <= w3y_temp;
                    state <= COMPUTE_INITIALS;
                end
                else begin
                    w1x <= w1x; w1y <= w1y;
                    w2x <= w2x; w2y <= w2y;
                    w3x <= w3x; w3y <= w3y;
                    state <= COMPUTE_COEFFS;
                end
            end
            COMPUTE_INITIALS: begin
                area <= area;
                reciprocal <= reciprocal;
                w1x <= w1x; w1y <= w1y;
                w2x <= w2x; w2y <= w2y;
                w3x <= w3x; w3y <= w3y;
                if(vdone) begin
                    w1 <= w1_temp+off1;
                    w2 <= w2_temp+off2;
                    w3 <= w3_temp+off3;
                    state <= IDLE;
                end
                else begin
                    w1 <= w1;
                    w2 <= w2;
                    w3 <= w3;
                    state <= COMPUTE_INITIALS;
                end
            end
            default: begin
                w1 <= w1; w1x <= w1x; w1y <= w1y;
                w2 <= w2; w2x <= w2x; w2y <= w2y;
                w3 <= w3; w3x <= w3x; w3y <= w3y;
                area <= area;
                reciprocal <= reciprocal;
                state <= IDLE;
            end
        endcase
    end
  end
endmodule
```

```
module IO_render2ram(clk, reset, state,render_data_in,
render_data_out,render_addr,we_in,ram0_data,

ram0_address,ram0_we,ram1_data,ram1_address,ram1_we);

inout [35:0] ram0_data, ram1_data;
input [19:0] render_addr;
output [35:0] render_data_in;
input [35:0] render_data_out;
input clk, state, we_in, reset;

output [18:0] ram0_address, ram1_address;
output ram0_we, ram1_we;

reg we, we2;

wire [35:0] render_data_in0, render_data_in1;

// state = 0 selects ram0
assign ram0_data = ((~reset) & ((!state) & (!we))) ?
render_data_out:36'hz;
assign render_data_in0 = ((~reset) & (we)) ? ram0_data:render_data_out;
assign ram0_address = ((~reset) & (!state)) ? render_addr[18:0] :
19'hz;
assign ram0_we = ((~reset) & (!state)) ? we_in : 1'hz;

// state = 1 selects ram1
assign ram1_data = ((~reset) & ((state) & (!we))) ?
render_data_out:36'hz;
assign render_data_in1 = ((~reset) & (we)) ? ram1_data:render_data_out;
assign ram1_address = ((~reset) & (state)) ? render_addr[18:0] : 19'hz;
assign ram1_we = ((~reset) & (state)) ? we_in : 1'hz;

assign render_data_in = (state) ? render_data_in1:render_data_in0;

always @ (posedge clk) begin
      if(reset) begin
            we <= 1'bz;
            we2 <= 1'bz;
      end
      else begin
            we <= we2;
            we2 <= we_in;
      end
end

endmodule
```

```
module lighting(clk, reset, start, t_matrix, vertex, normal,
                face_color, color_out, done);
    input clk, reset, start;
    input [255:0] t_matrix;
    input [47:0] vertex;
    input [47:0] normal;
    input [23:0] face_color;
    output [23:0] color_out;
    output done;
    reg done;

    reg [2:0] state;
    parameter IDLE = 0;
    parameter WAIT_MATRIX = 1;
    parameter START_DOT = 2;
    parameter WAIT_DOT = 3;
    parameter START_COMP = 4;
    parameter WAIT_COMP = 5;

    reg [47:0] light_vector = {16'd0,16'd0,-16'd16384};

    wire tdone;
    wire [127:0] tres;

    matrix_mult mmt(clk, start, normal, t_matrix[255:192],
                    t_matrix[191:128], t_matrix[127:64],
t_matrix[63:0],
                    tres, tdone);

    wire [15:0] nx = tres[122:107];
    wire [15:0] ny = tres[90:75];
    wire [15:0] nz = tres[58:43];

    wire [47:0] n2 = {nx, ny, nz};

    wire dstart = (state == START_DOT);
    wire [31:0] weight;
    wire [15:0] shift = 0;
    wire ddone;

    dot_product dpd(clk, dstart, n2, light_vector, shift, weight,
ddone);

    wire [31:0] weight2 = (weight[31] ? (~weight[31:0] +
1):weight[31:0]);

    wire cstart = (state == START_COMP);
    wire [23:0] color_out;
    wire cdone;

    color_computer ccomp(clk, reset, cstart, face_color, weight2,
                         color_out, cdone);

    always @ (posedge clk) begin
       if(reset) begin
           state <= IDLE;
           done <= 0;
```

```
        end
        else begin
            case (state)
            IDLE: begin
               if(start) begin
                  state <= WAIT_MATRIX;
                  done <= 0;
               end
               else begin
                  state <= state;
                  done <= done;
               end
            end
            WAIT_MATRIX: begin
               done <= 0;
               if(tdone) state <= START_DOT;
               else state <= state;
            end
            START_DOT: begin
               done <= 0;
               state <= WAIT_DOT;
            end
            WAIT_DOT: begin
               done <= 0;
               if(ddone) state <= START_COMP;
               else state <= state;
            end
            START_COMP: begin
               done <= 0;
               state <= WAIT_COMP;
            end
            WAIT_COMP: begin
               if(cdone) begin
                  done <= 1;
                  state <= IDLE;
               end
               else begin
                  done <= 0;
                  state <= state;
               end
            end
            endcase
        end
    end
endmodule
```

```
module matrix_mult(clk, start, vector, row1, row2, row3, row4, result,
                   done);
    input clk, start;
    input [47:0] vector;
    input [63:0] row1;
    input [63:0] row2;
    input [63:0] row3;
    input [63:0] row4;

    output [127:0] result;
    reg [127:0] result;
    output done;
    reg done;

    wire [31:0] p1;
    wire [31:0] p2;
    wire [31:0] p3;
    wire [31:0] p4;

    wire d1, d2, d3, d4;

    dot_product r1(clk, start, vector, row1[63:16], row1[15:0], p1,
d1);
    dot_product r2(clk, start, vector, row2[63:16], row2[15:0], p2,
d2);
    dot_product r3(clk, start, vector, row3[63:16], row3[15:0], p3,
d3);
    dot_product r4(clk, start, vector, row4[63:16], row4[15:0], p4,
d4);

    always @ (posedge clk) begin
        if(start) begin
            result <= 0;
            done <= 0;
        end
        else if(d1 & d2 & d3 & d4) begin
            result[127:96] <= p1;
            result[95:64] <= p2;
            result[63:32] <= p3;
            result[31:0] <= p4;
            done <= 1;
        end
        else begin
            result <= result;
            done <= done;
        end
    end

endmodule
```

```
module model_rom(clk, face_addr, vertex_addr, normal_addr, face,
vertex, normal);
    input clk;
    input [11:0] face_addr, vertex_addr, normal_addr;
    output [95:0] face;
    reg [95:0] face;
    output [47:0] vertex, normal;
    reg [47:0] vertex, normal;

    always @ (posedge clk) begin
       case (face_addr)
           //number of faces + 1
                 12'd0: face <= 5;
           //faces: color (24 bits), then n1, n2, n3, v1, v2, v3 (12
bits each)
           12'd1: face <= 96'hfe00fe001001001002004003;
                 12'd2: face <= 96'h00fe00002002002001004002;
           12'd3: face <= 96'h0000fe003003003001002003;
                 12'd4: face <= 96'hfe000004004004001003004;
           default: face <= 96'hz;
       endcase
       case (vertex_addr)
           //first value must be scale-- IT MUST EQUAL 2048
           12'd0: vertex <= 48'd2048;
           //vertices: x, y, z (16 bits each, max value 2047)
           12'd1: vertex <= {16'd1428,-16'd1224,16'd816};
           12'd2: vertex <= {16'd0,-16'd1224,-16'd1632};
           12'd3: vertex <= {16'd0, 16'd2040,16'd0};
           12'd4: vertex <= {-16'd1428, -16'd1224, 16'd816};
           default: vertex <= 48'hz;
       endcase
       case (normal_addr)
           //unused
           12'd0: normal <= 0;
           //normals: <x,y,z>, must be magnitude ~16384
           12'd1: normal <= {-16'd6412, 16'd7480, -16'd13091};
           12'd2: normal <= {16'd0, -16'd16384, 16'd0};
           12'd3: normal <= {16'd6412, 16'd7480, -16'd13091};
           12'd4: normal <= {16'd0, 16'd4501, 16'd15754};
           default: normal <= 48'hz;
       endcase
    end
endmodule
```

```
module model_rom_tetrahedron(clk, face_addr, vertex_addr, normal_addr,
face, vertex, normal);
      input clk;
      input [11:0] face_addr, vertex_addr, normal_addr;
      output [95:0] face;
      reg [95:0] face;
      output [47:0] vertex, normal;
      reg [47:0] vertex, normal;

      always @ (posedge clk) begin
            case (vertex_addr)
                  12'd0: vertex <= 2048;
                  12'd1: vertex <= {-16'd1182, -16'd1182, -16'd1182};
                  12'd2: vertex <= {16'd1182, -16'd1182, -16'd1182};
                  12'd3: vertex <= {-16'd1182, 16'd1182, -16'd1182};
                  12'd4: vertex <= {-16'd1182, -16'd1182, 16'd1182};

                  default: vertex <= 48'hz;
            endcase
            case (face_addr)
                  12'd0: face <= 6;
                  12'd1: face <= {24'hFFFF00, 12'd1, 12'd3, 12'd2,
12'd1, 12'd3, 12'd2};
                  12'd2: face <= {24'hFFFF00, 12'd1, 12'd4, 12'd3,
12'd1, 12'd4, 12'd3};
                  12'd3: face <= {24'hFFFF00, 12'd1, 12'd2, 12'd4,
12'd1, 12'd2, 12'd4};
                  12'd4: face <= {24'hFFFF00, 12'd2, 12'd3, 12'd4,
12'd2, 12'd3, 12'd4};
            default: face <= 96'hz;
            endcase
            case (normal_addr)
                  12'd0: normal <= 0;
                  12'd1: normal <= {-16'd9460, -16'd9460, -16'd9460};
                  12'd2: normal <= {16'd16384, 16'd0, 16'd0};
                  12'd3: normal <= {16'd0, 16'd16384, 16'd0};
                  12'd4: normal <= {16'd0, 16'd0, 16'd16384};
                  default: normal <= 48'hz;
            endcase
      end
endmodule
```

```
module model_switcher(clk, model_choice, face_addr, vertex_addr,
                      normal_addr, face, vertex, normal);

      input clk;
      input [1:0] model_choice;
      input [11:0] face_addr, vertex_addr, normal_addr;
      output [95:0] face;
      output [47:0] vertex, normal;

//    wire [11:0] face_addr1, vertex_addr1, normal_addr1,
//                face_addr2, vertex_addr2, normal_addr2,
//                face_addr3, vertex_addr3, normal_addr3;

      wire [95:0] face0, face1, face2, face3, face4, face5;

      wire [47:0]  vertex0, normal0, vertex1, normal1,
                        vertex2, normal2,
                        vertex3, normal3,
                        vertex4, normal4,
                        vertex5, normal5;

      model_rom_tetrahedron tetrahedron(clk, face_addr, vertex_addr,
                                    normal_addr, face0, vertex0,
normal0);

      model_rom_cube cube(clk, face_addr, vertex_addr,
                                    normal_addr, face1, vertex1,
normal1);

      model_rom_shuttle shuttle(clk, face_addr, vertex_addr,
                                    normal_addr, face2, vertex2,
normal2);

  //  model_rom_teddybear teddy(clk, face_addr, vertex_addr,
//                                    normal_addr, face3, vertex3,
normal3);

      assign face4 = (model_choice[0] == 1) ? face1 : face0;
      assign face5 = (model_choice[0] == 1) ? 96'hz : face2;
      assign face = (model_choice[1] == 0) ? face4 : face5;

     assign vertex4 = (model_choice[0] == 1) ? vertex1 : vertex0;
      assign vertex5 = (model_choice[0] == 1) ? 48'hz : vertex2;
      assign vertex = (model_choice[1] == 0) ? vertex4 : vertex5;

      assign normal4 = (model_choice[0] == 1) ? normal1 : normal0;
      assign normal5 = (model_choice[0] == 1) ? 48'hz : normal2;
      assign normal = (model_choice[1] == 0) ? normal4 : normal5;

      endmodule
```

```
//start MUST go high every other clock (during operation)
module pixel_pipe(clk, reset, start, xloc, yloc, colors, z1, z2, z3,
                  weight1, weight2, weight3, we, address, data_in,
data_out);
    input clk, reset, start;
    input [9:0] xloc, yloc;
    input [71:0] colors;
    input [39:0] weight1, weight2, weight3;
    input [11:0] z1, z2, z3;
    input [35:0] data_in;
    output [35:0] data_out;
    output we;
    output [19:0] address;

    reg [9:0] curr_x, curr_y, curr_x2, curr_y2;
    reg we;
    reg last_start;
    reg last_start2;

    assign address[19:10] = start ? yloc:curr_y;
    assign address[9:0] = start ? xloc:curr_x;

    wire loc2 = ~(weight1[39] | weight2[39] | weight3[39]);
    reg loc1;
    reg location_test;

    reg dv3;
    reg dv2;
    reg dv;

    reg [11:0] zout2, zout;
    wire [11:0] zout3;
    wire [11:0] zmax, zmax1, zmin, zmin1;
    assign zmax1 = (z1 > z2) ? z1:z2;
    assign zmin1 = (z1 < z2) ? z1:z2;
    assign zmax = (zmax1 > z3) ? zmax1:z3;
    assign zmin = (zmin1 < z3) ? zmin1:z3;

    depth_computer dc(clk, weight1, weight2, weight3, z1, z2, z3, zmax,
zmin, zout3);

    wire [23:0] color_out3;
    reg [23:0] color_out2, color_out;
    color_interpolater ci(clk, weight1, weight2, weight3, colors,
color_out3);

    wire [11:0] z_value = data_in[35:24];
    wire z_test =  (zout3 < z_value);
    wire data_valid = start & last_start2 & location_test & z_test;
    wire we2 = ~start | ~data_valid;

    wire [35:0] mem_value;
    assign mem_value[35:24] = zout;
    assign mem_value[23:0] = color_out;
    //assign mem_value[23:0] = colors[23:0];
    //assign mem_value[23:16] = zout[11:4];
    //assign mem_value[15:0] = 16'h007f;
```

```
    assign data_out = (dv) ? mem_value:36'hz;

    always @ (posedge clk) begin
        if(reset) begin
            dv <= 0;
            dv2 <= 0;
            dv3 <= 0;
            curr_x <= 0;
            curr_y <= 0;
            curr_x2 <= 0;
            curr_y2 <= 0;
            we <= 1;
            last_start <= 0;
            last_start2 <= 0;
            loc1 <= 0;
            location_test <= 0;
            zout <= 0;
            zout2 <= 0;
              color_out <= 0;
              color_out2 <= 0;
        end
        else begin
             color_out <= color_out2;
             color_out2 <= color_out3;
           zout <= zout2;
           zout2 <= zout3;
           dv <= dv2;
           dv2 <= dv3;
           dv3 <= data_valid;
           we <= we2;
           location_test <= loc1;
           loc1 <= loc2;
           last_start2 <= last_start;
           last_start <= start;
           curr_x <= curr_x2;
           curr_y <= curr_y2;
           if(start) begin
               curr_x2 <= xloc;
               curr_y2 <= yloc;
           end
           else begin
               curr_x2 <= curr_x2;
               curr_y2 <= curr_y2;
           end
        end
    end
endmodule
```

```
module poly_shader(clk, reset, start, pixels, colors, we, address,
data_in,
                   data_out, done);
    input clk, reset, start;
    input [107:0] pixels;
    input [71:0] colors;
    output we;
    output [19:0] address;
    input [35:0] data_in;
    output [35:0] data_out;
    output done;
    reg done;

    reg [107:0] pxls;
    reg [71:0] clrs;

    reg [2:0] state;
    parameter IDLE = 0;
    parameter LOAD_PXLS = 1;
    parameter LOAD_COUNTERS = 2;
    parameter START_GOURAUD = 3;
    parameter WAIT_GOURAUD = 4;
    parameter COMPUTE_PIXEL = 5;
    parameter WAIT = 6;

    parameter XSHIFT = 320;
    parameter YSHIFT = 240;

    wire [10:0] x1 = (pxls[106:96]+XSHIFT);
    wire [10:0] y1 = (pxls[94:84]+YSHIFT);
    wire [11:0] z1 = pxls[83:72];
    wire [10:0] x2 = (pxls[70:60]+XSHIFT);
    wire [10:0] y2 = (pxls[58:48]+YSHIFT);
    wire [11:0] z2 = pxls[47:36];
    wire [10:0] x3 = (pxls[34:24]+XSHIFT);
    wire [10:0] y3 = (pxls[22:12]+YSHIFT);
    wire [11:0] z3 = pxls[11:0];

    wire [9:0] xmin1 = (x1 < x2) ? x1[9:0]:x2[9:0];
    wire [9:0] xmax1 = (x1 > x2) ? x1[9:0]:x2[9:0];
    wire [9:0] xmin = (xmin1 < x3) ? xmin1:x3[9:0];
    wire [9:0] xmax = 1+((xmax1 > x3) ? xmax1:x3[9:0]);

    wire [9:0] ymin1 = (y1 < y2) ? y1[9:0]:y2[9:0];
    wire [9:0] ymax1 = (y1 > y2) ? y1[9:0]:y2[9:0];
    wire [9:0] ymin = (ymin1 < y3) ? ymin1:y3[9:0];
    wire [9:0] ymax = 1+((ymax1 > y3) ? ymax1:y3[9:0]);

    reg [9:0] xloc;
    reg [9:0] yloc;

    reg [39:0] weight1;
    reg [39:0] weight2;
    reg [39:0] weight3;
    reg [39:0] weight1_s;
    reg [39:0] weight2_s;
```

```
    reg [39:0] weight3_s;

    wire [10:0] v1x = (x2 + 1 + ~x1);
    wire [10:0] v1y = (y2 + 1 + ~y1);
    wire [10:0] v2x = (x3 + 1 + ~x2);
    wire [10:0] v2y = (y3 + 1 + ~y2);
    wire [10:0] v3x = (x1 + 1 + ~x3);
    wire [10:0] v3y = (y1 + 1 + ~y3);

    wire done2 = (state == IDLE);
    wire pixel_start = (state == COMPUTE_PIXEL);

    pixel_pipe ppipe(clk, reset, pixel_start, xloc, yloc, clrs,
                     z1, z2, z3, weight1, weight2, weight3,
                     we, address, data_in, data_out);

    wire [39:0] w1, w1x, w1y, w2, w2x, w2y, w3, w3x, w3y;
    wire gouraud_start = (state == START_GOURAUD);
    wire gouraud_done;

    gouraud_computer gcomp(clk, reset, gouraud_start, xloc, yloc,
                           x1[9:0], y1[9:0], x2[9:0], y2[9:0], x3[9:0],
y3[9:0],
                           v1x, v1y, v2x, v2y, v3x, v3y, w1, w1x, w1y,
                           w2, w2x, w2y, w3, w3x, w3y, gouraud_done);

    always @ (posedge clk) begin
        if(reset) begin
            pxls <= 0;
            clrs <= 0;
            xloc <= 0;
            yloc <= 0;
            state <= IDLE;
            done <= 1;
            weight1_s <= 0; weight2_s <= 0; weight3_s <= 0;
            weight1 <= 0; weight2 <= 0; weight3 <= 0;
        end
        else begin
            done <= done2;
            case (state)
                IDLE: begin
                     weight1 <= 0; weight2 <= 0; weight3 <= 0;
                     weight1_s <= 0; weight2_s <= 0; weight3_s <= 0;
                     pxls <= 0;
                     xloc <= 0;
                     clrs <= 0;
                     yloc <= 0;
                     if(start) state <= LOAD_PXLS;
                     else state <= IDLE;
                end
                LOAD_PXLS: begin
                     weight1 <= 0; weight2 <= 0; weight3 <= 0;
                     weight1_s <= 0; weight2_s <= 0; weight3_s <= 0;
                     pxls <= pixels;
                     clrs <= colors;
                     xloc <= 0;
                     yloc <= 0;
```

```
     state <= LOAD_COUNTERS;
end
LOAD_COUNTERS: begin
     pxls <= pxls;
     clrs <= clrs;
     weight1 <= 0; weight2 <= 0; weight3 <= 0;
     weight1_s <= 0; weight2_s <= 0; weight3_s <= 0;
     xloc <= xmin;
     yloc <= ymin;
     state <= START_GOURAUD;
end
START_GOURAUD: begin
     pxls <= pxls;
     clrs <= clrs;
     weight1 <= 0; weight2 <= 0; weight3 <= 0;
     weight1_s <= 0; weight2_s <= 0; weight3_s <= 0;
     xloc <= xloc;
     yloc <= yloc;
     if(gouraud_done) state <= START_GOURAUD;
     else state <= WAIT_GOURAUD;
end
WAIT_GOURAUD: begin
     pxls <= pxls;
     clrs <= clrs;
     xloc <= xloc;
     yloc <= yloc;
     if(gouraud_done) begin
        state <= COMPUTE_PIXEL;
        weight1 <= w1;
        weight2 <= w2;
        weight3 <= w3;
        weight1_s <= w1;
        weight2_s <= w2;
        weight3_s <= w3;
     end
     else begin
        state <= WAIT_GOURAUD;
        weight1 <= 0; weight2 <= 0; weight3 <= 0;
        weight1_s <= 0; weight2_s <= 0; weight3_s <= 0;
     end
end
COMPUTE_PIXEL: begin
     weight1 <= weight1;
     weight1_s <= weight1_s;
     weight2 <= weight2;
     weight2_s <= weight2_s;
     weight3 <= weight3;
     weight3_s <= weight3_s;
     pxls <= pxls;
     clrs <= clrs;
     xloc <= xloc;
     yloc <= yloc;
     state <= WAIT;
end
WAIT: begin
     pxls <= pxls;
     clrs <= clrs;
```

```
                    if(xloc == xmax) begin
                        xloc <= xmin;
                        yloc <= yloc+1;
                        weight1 <= weight1_s + w1y;
                        weight1_s <= weight1_s + w1y;
                        weight2 <= weight2_s + w2y;
                        weight2_s <= weight2_s + w2y;
                        weight3 <= weight3_s + w3y;
                        weight3_s <= weight3_s + w3y;
                    end
                    else begin
                        xloc <= xloc+1;
                        yloc <= yloc;
                        weight1 <= weight1+w1x;
                        weight1_s <= weight1_s;
                        weight2 <= weight2+w2x;
                        weight2_s <= weight2_s;
                        weight3 <= weight3+w3x;
                        weight3_s <= weight3_s;
                    end
                    if(yloc == ymax) state <= IDLE;
                    else state <= COMPUTE_PIXEL;
                end
            endcase
        end
    end
endmodule
```

```
// vsim -L E:/Modeltech_6.0b/xilinx/XilinxCoreLib_ver
work.poly_shader_test
`timescale 1ns/10ps
module poly_shader_test();
    reg clk;
    reg reset;
    reg start;

    reg [35:0] pix1, pix2, pix3;
    wire [107:0] pixels;

    assign pixels[107:72] = pix1;
    assign pixels[71:36] = pix2;
    assign pixels[35:0] = pix3;

    wire we;
    wire [19:0] address;
    wire [35:0] data_in, data_out;
    wire done;
    wire frame = 0;

    reg we2, we3;

    poly_shader ps(clk, reset, start, pixels, we, address, data_in,
data_out,
                   done, frame);

    initial begin
        clk = 1;
        reset = 1;
        start = 1;
        pix1 = 36'h06b0a000f;
        pix2 = 36'h0c0030007;
        pix3 = 36'h1400a000f;
        we2 = 1;
        we3 = 1;
    end

    assign data_in = we3 ? 36'hfff000000:data_out;

    always #50 clk = ~clk;
    always #400 reset =0;
    always #600 start = 0;

    always @ (posedge clk) begin
        we3 <= we2;
        we2 <= we;
    end


endmodule
```

```
module ptransform(clk, reset, start, x, y, z, shift, px, py, pz, done);
      input clk, reset, start;
      input [23:0] x, y, z;
      input [31:0] shift;
      output [11:0] px, py, pz;
   output done;

      //maximum input coordinate: 2048
      //all input coordinates should be multiplied by 2048
      wire [31:0] zext = {z[23], z[23], z[23], z[23], z[23],
                          z[23], z[23], z[23], z};
      wire [31:0] w = zext + {shift[20:0],11'b0};
      wire [31:0] wshift = {w[31],w[31],w[31],w[31],w[31],w[31],
                            w[31],w[31],w[31],w[31],w[31], w[31:11]};

      wire [19:0] xfactor = 20'h00708;
      wire [19:0] yfactor = 20'hffa60;

      wire [43:0] xtemp;
      wire [43:0] ytemp;

      wire [31:0] zshift = 32'h400000 + zext;
      wire [31:0] wshift2 =
{wshift[31],wshift[31],wshift[31],wshift[31],wshift[31:4]};

      xilinx_multiplier20x24 xmul(.clk(clk), .a(xfactor), .b(x),
.q(xtemp));
      xilinx_multiplier20x24 ymul(.clk(clk), .a(yfactor), .b(y),
.q(ytemp));

      wire [31:0] xres, yres, zres;
      wire xdone, ydone, zdone;

      reg dstart, dstart2, dstart3;

      wrapped_divider xdiv(clk, dstart, xtemp[42:11], wshift, xres,
                           xdone);
      wrapped_divider ydiv(clk, dstart, ytemp[42:11], wshift, yres,
                           ydone);
      wrapped_divider zdiv(clk, dstart, zshift, wshift2, zres,
                           zdone);

      assign done = (xdone & ydone & zdone);

      assign px = xres[11:0];
      assign py = yres[11:00];
      assign pz = (zres > 32'h00000fff) ? 12'hfff:zres[11:0];

      always @ (posedge clk) begin
            if(reset) begin
                  dstart <= 0;
                  dstart2 <= 0;
                  dstart3 <= 0;
            end
            else begin
                  dstart <= dstart2;
                  dstart2 <= dstart3;
```

```
                dstart3 <= start;
            end
        end
    endmodule
```

```
module render_unit(clk, reset, start, next, shift, t_matrix,
face_counter, vertex_addr,
                   normal_addr, face, vertex, normal, pixels, colors,
dtr, done);
    input clk, reset, start, next;
    input [95:0] face;
    input [47:0] vertex, normal;
    input [255:0] t_matrix;
    input [31:0] shift;
    output [11:0] face_counter, vertex_addr, normal_addr;
    output [107:0] pixels;
    output [71:0] colors;
    output dtr;
    output done;
    reg done;

    reg [2:0] state;
    parameter IDLE = 0;
    parameter LOAD_END = 2;
    parameter LOAD_FACE = 3;
    parameter RENDER = 4;
    parameter WAIT = 5;

    reg [11:0] face_counter;
    reg [11:0] face_end;
    reg [2:0] delay_counter;

    reg [35:0] vertices;
    reg [23:0] face_color;
    reg [35:0] normals;

    parameter DELAY = 2;

    wire face_start = (state == RENDER & delay_counter < DELAY);
    wire data_ready;
    assign dtr = (data_ready & (state == WAIT));
    wire done2 = (state == IDLE);

    face_renderer frend(clk, reset, face_start, shift, t_matrix,
vertices,
                        normals, face_color,
                        vertex_addr, normal_addr, vertex, normal,
                        pixels, colors, data_ready);

    always @ (posedge clk) begin
        if(reset) begin
            face_end <= 0;
            face_counter <= 0;
            delay_counter <= 0;
            state <= IDLE;
            vertices <= 0;
            face_color <= 0;
            normals <= 0;
            done <= 1;
        end
        else if(delay_counter < DELAY) begin
            done <= done2;
```

```
    face_end <= face_end;
    delay_counter <= delay_counter + 1;
    face_counter <= face_counter;
    state <= state;
    vertices <= vertices;
    face_color <= face_color;
    normals <= normals;
end
else begin
    done <= done2;
    case (state)
        IDLE: begin
            vertices <= 0;
            face_color <= 0;
            normals <= 0;
            face_end <= 0;
            face_counter <= 0;
            if(start) begin
                state <= LOAD_END;
                delay_counter <= 0;
            end
            else begin
                state <= state;
                delay_counter <= delay_counter;
            end
        end
        LOAD_END: begin
            face_end <= face[11:0];
            face_counter <= 1;
            state <= LOAD_FACE;
            delay_counter <= 0;
            vertices <= 0;
            face_color <= 0;
            normals <= 0;
        end
        LOAD_FACE: begin
            face_end <= face_end;
            vertices <= face[35:0];
            normals <= face[71:36];
            face_color <= face[95:72];
            face_counter <= face_counter+1;
            delay_counter <= 0;
            state <= RENDER;
        end
        RENDER: begin
            face_end <= face_end;
            vertices <= vertices;
            normals <= normals;
            face_color <= face_color;
            face_counter <= face_counter;
            delay_counter <= delay_counter;
                        if(data_ready) state <= WAIT;
            else state <= RENDER;
        end
        WAIT: begin
            face_end <= face_end;
            face_counter <= face_counter;
```

```
                    if(face_end == face_counter) begin
                       state <= IDLE;
                       delay_counter <= delay_counter;
                    end
                    else if(next) begin
                       state <= LOAD_FACE;
                       delay_counter <= 0;
                    end
                    else begin
                       state <= WAIT;
                       delay_counter <= delay_counter;
                    end
                end
                default: begin
                    vertices <= vertices;
                    normals <= normals;
                    face_color <= face_color;
                    face_end <= face_end;
                    face_counter <= face_counter;
                    state <= IDLE;
                    delay_counter <= delay_counter;
                end
            endcase
        end
    end
endmodule
```

```
module renderer(clk, reset, start, rom_value, shift, t_matrix,
                pixel_loc, done);

    input clk, reset, start;
    input [47:0] rom_value;
    input [255:0] t_matrix;
    input [31:0] shift;

    output done;
    reg pstart;
    output [35:0] pixel_loc;
    reg [35:0] pixel_loc;

    wire tdone;
    wire [127:0] tres;

    matrix_mult mmt(clk, start, rom_value[47:0], t_matrix[255:192],
                    t_matrix[191:128], t_matrix[127:64],
t_matrix[63:0],
                    tres, tdone);

    wire [23:0] xv = tres[119:96];
    wire [23:0] yv = tres[87:64];
    wire [23:0] zv = tres[55:32];
    wire pdone;
    wire [11:0] xout,yout,zout;

    ptransform pt(clk, reset, pstart, xv, yv, zv, shift, xout, yout,
zout, pdone);

    reg [2:0] state;
    parameter IDLE = 0;
    parameter ROT = 1;
    parameter PERS = 2;
    parameter HOLD = 3;

    reg done;

    always @ (posedge clk) begin
        if(reset) begin
            state <= IDLE;
            pixel_loc <= 36'hzzzzzzzzz;
            pstart <= 0;
            done <= 0;
        end
        else begin
            case (state)
                IDLE: begin
                    done <= done;
                    pixel_loc <= pixel_loc;
                    pstart <= 0;
                    if(start) state <= ROT;
                    else state <= IDLE;
                end
                ROT: begin
                    done <= 0;
                    pstart <= 1;
```

```
                    pixel_loc <= pixel_loc;
                    if(tdone) begin
                       state <= PERS;
                    end
                    else begin
                       state <= state;
                    end
                end
                PERS: begin
                    done <= 0;
                    pstart <= 0;
                    pixel_loc <= pixel_loc;
                    if(pdone) begin
                       state <= HOLD;
                    end
                    else begin
                       state <= state;
                    end
                end
                HOLD: begin
                                pstart <= 0;
                    done <= 1;
                    pixel_loc[35:24] <= xout;
                    pixel_loc[23:12] <= yout;
                    pixel_loc[11:0] <= zout;
                    state <= IDLE;
                end
            endcase
        end
    end
endmodule
```

```
module wrapped_divider(clk, start, quot, divisor, res, done);
    input clk, start;
    input [31:0] quot;
    input [31:0] divisor;
    output [31:0] res;
    output done;

    reg [31:0] x;
    reg [31:0] y;
    reg [31:0] res;
    reg done;

    parameter LATENCY = 40;
    reg [7:0] counter;

    wire [31:0] xres;
    wire [31:0] xrem;

    wire [31:0] pos_rem = xrem[31] ? (~xrem + 1):xrem;
    wire [31:0] pos_div = divisor[31] ? (~divisor+1):divisor;
    wire [31:0] half_div = {1'b0, pos_div[31:1]};

    wire [31:0] neg_add = (pos_rem > half_div) ? 32'hffffffff:32'h0;
    wire [31:0] pos_add = (pos_rem > half_div) ? 32'h1:32'h0;
    wire [31:0] add = (xrem[31] ^ divisor[31]) ? neg_add:pos_add;

    xilinx_divider dvdr(x, y, xres, xrem, clk);

    always @ (posedge clk) begin
        if(start) begin
            x <= quot;
            y <= divisor;
            done <= 0;
            res <= 0;
            counter <= 1;
        end
        else if (counter > LATENCY) begin
            res <= res;
            counter <= counter;
            done <= 1;
            x <= x;
            y <= y;
        end
        else if(counter < LATENCY) begin
            res <= res;
            counter <= counter+1;
            done <= 0;
            x <= x;
            y <= y;
        end
        else begin
            res <= (xres+add);
            counter <= counter+1;
            done <= 0;
            x <= x;
            y <= y;
        end
```

```
    end
endmodule
```

```
#!/usr/athena/bin/perl
use POSIX qw(ceil floor);
use List::Util qw(first max maxstr min minstr reduce shuffle sum);


#-----------------------
# Add vectors
#-----------------------
sub addvec {
    my @v1 = @{$_[0]};
    my @v2 = @{$_[1]};
    my @returnvec;
    my $i;

    for ($i = 0 ; $i <= 2 ; $i++) {
      @returnvec[$i] = @v1[$i] + @v2[$i];
    }

    return @returnvec;
}

#-----------------------
# Subtract vectors
#-----------------------
sub subvec {
    my @v1 = @{$_[0]};
    my @v2 = @{$_[1]};
    my @returnvec;
    my $i;

    for ($i = 0 ; $i <= 2 ; $i++) {
      @returnvec[$i] = @v1[$i] - @v2[$i];
    }

    return @returnvec;
}

#-----------------------
# Cross Product
#-----------------------
sub crossp {
    my @v1 = @{$_[0]};
    my @v2 = @{$_[1]};
    my @returnvec;

    @returnvec[0] = @v1[1]*@v2[2] - @v1[2]*@v2[1];
    @returnvec[1] = -(@v1[0]*@v2[2] - @v1[2]*@v2[0]);
    @returnvec[2] = @v1[0]*@v2[1] - @v1[1]*@v2[0];

    return @returnvec;
}

#---------------------------
#  Decimal to Signed Hex
#---------------------------
sub dec2hex {
    my $str = unpack("H8", pack("N", shift));
```

```
    $str = substr($str,4);
    return $str;
}

#----------------------------
#  Actual program
#----------------------------

#@vec1 = (1,7,9);
#@vec2 = (1,4,-4);
#print crossp(\@vec1,\@vec2);

open(PRE,"OBJ/shuttle.obj");
open(POST,">temp.obj");


while (<PRE>) {
    if (/^f/) {
      @temp = split(/\s+/,$_);
      $l = @temp;
      for ($i = 1; $i < ($l-2) ; $i++) {
          print POST "f @temp[1] @temp[1+$i] @temp[2+$i] \n";
      }
    }
    else { print POST $_ }
}

close(PRE);
close(POST);

open(MODEL,"temp.obj");

$i = 0;
$j = 0;

my %colorhash = ();
$colorhash{'blue'} = '0000FF';
$colorhash{'black'} = '000000';
$colorhash{'LimeGreen'} = '32CD32';
$colorhash{'grey'} = 'BEBEBE';
$colorhash{'red'} =  'FF0000';
$colorhash{'yellow'} = 'FFFF00';
$colorhash{'drkgrey'} = '8F8F8F';
$colorhash{''} = 'FFC0CB';
$colorhash{'brown'} = 'A52A2A';
$colorhash{'orange'} = 'FFA500';

$normal_method = 1;

   while (<MODEL>) {
   if (/^g /) { @line = split(/\s+/,$_); }
   if (/^v/)  { $vertices[$i] = $_;
            @temp = split(/\s+/,$_);
            $ver[$i][0] = @temp[1];
            $ver[$i][1] = @temp[2];
            $ver[$i][2] = @temp[3];
            $i = $i + 1;}
```

```
   if (/usemtl/) { @crap = split(/\s+/,$_);
              $crap2 = @crap[1];
             }
   if (/^f/)  { @temp = split(/\s+/,$_);
          @temp = map {$_ - 1} @temp;
          $face_ver[$j][0] = @temp[1];
          $face_ver[$j][1] = @temp[2];
          $face_ver[$j][2] = @temp[3];
      #     print "$face_ver[$j][0] $face_ver[$j][1] $face_ver[$j][2]
\n";
          @v1 = @{@ver[$temp[1]]};
          @v2 = @{@ver[$temp[2]]};
          @v3 = @{@ver[$temp[3]]};
          @vtemp1 = subvec(\@v1,\@v2);
          @vtemp2 = subvec(\@v2,\@v3);
          @vtemp3 = crossp(\@vtemp1,\@vtemp2);
          $vnormal[$j][0] = @vtemp3[0];
                $vnormal[$j][1] = @vtemp3[1];
          $vnormal[$j][2] = @vtemp3[2];
      #     print "@{@vnormal[$j]} \n";
          @facecolor[$j] = $colorhash{$crap2};
          $j = $j + 1;
        }
}
$ver_length = @vertices;
$face_length = $j;

for $h (0 .. $ver_length) {
    for $i ( 0 .. $#face_ver) {
      for $j ( 0 .. $#{$face_ver[$i]}) {
          if ($h == $face_ver[$i][$j]) {
            push @{ $v2f[$h] },$i ;
          }
      }
    }
#    print "@{@v2f[$h]} \n";
}

# print "$#v2f \n";

if ($normal_method) {
   for ($i=0; $i<= $#v2f;$i++) {
       @temp = ();
       for ($j=0; $j <= $#{$v2f[$i]}; $j++) {
         @crap1 = @{@vnormal[$v2f[$i][$j]]};
         @crap2 = addvec(\@crap1,\@temp);
         @temp = @crap2;
       }
       @temp = map(1/$j*$_, @temp);
       push @vnormal_avg,[@temp];
    #  print "@{@vnormal_avg[$i]} \n \n";
   }}
else {
    for ($i=0; $i< $face_length; $i++) {
      $x = $vnormal[$i][0];
      $y = $vnormal[$i][1];
      $z = $vnormal[$i][2];
```

```
      $temp = sqrt(($x*$x)+($y*$y)+($z*$z));
      $vnormal[$i][0] = floor(16384*$x/$temp);
      $vnormal[$i][1] = floor(16384*$y/$temp);
      $vnormal[$i][2] = floor(16384*$z/$temp);
    }
}

for ($j=0; $j < 3; $j++) {
    @temp = ();
    for($i = 0; $i < $ver_length; $i++){
      @temp[$i] = $ver[$i][$j]
       }
    $move[$j] = ((max @temp) + (min @temp))/2;
}

for ($j=0; $j < 3; $j++) {
    for($i = 0; $i < $ver_length; $i++){
      $ver[$i][$j] = $ver[$i][$j] - $move[$j];
    }}

for ($i=0; $i < $ver_length; $i++) {
    @mag[$i] =
sqrt($ver[$i][0]*$ver[$i][0]+$ver[$i][1]*$ver[$i][1]+$ver[$i][2]*$ver[$
i][2]);
    if ($normal_method) {
      $x = $vnormal_avg[$i][0];
      $y = $vnormal_avg[$i][1];
      $z = $vnormal_avg[$i][2];
#     print "$x\t$y\t$z  \n";
      $temp = sqrt(($x*$x)+($y*$y)+($z*$z));
#     print "$temp \n";
      $vnormal_avg[$i][0] = floor(16384*$x/$temp);
      $vnormal_avg[$i][1] = floor(16384*$y/$temp);
      $vnormal_avg[$i][2] = floor(16384*$z/$temp);
#     print "@{@vnormal_avg[$i]} \n";
    }}



$scale = floor(2048/(max @mag));

for ($j=0; $j < 3; $j++) {
    for($i = 0; $i < $ver_length; $i++){
      $ver[$i][$j] = $ver[$i][$j] * $scale;
    }
}



# $fileout = sprintf("model_rom_%s",$line[1]);
# open(FILEOUT,sprintf(">%s.coe",$fileout));

# print FILEOUT "module $fileout(clk, face_addr, vertex_addr,
normal_addr, face, vertex, normal); \n\t";
# print FILEOUT "input clk; \n\t";
# print FILEOUT "input [11:0] face_addr, vertex_addr, normal_addr;
\n\t";
```

```
# print FILEOUT "output [95:0] face; \n\t";
# print FILEOUT "reg [95:0] face; \n\t";
# print FILEOUT "output [47:0] vertex, normal; \n\t";
# print FILEOUT "reg [47:0] vertex, normal; \n\n\t";
# print FILEOUT "always @ (posedge clk) begin \n\t\t";

# print FILEOUT "case (vertex_addr) \n\t\t\t";
# printf FILEOUT ("12'd0: vertex <= %i;",2048);

# for ($i=0; $i < $ver_length; $i++) {
#     $x = $ver[$i][0];
#     $y = $ver[$i][1];
#     $z = $ver[$i][2];
#     printf FILEOUT ("\n\t\t\t\t12'd%i: vertex <= {",($i+1));
#     if ($x < 0) { printf FILEOUT ("-16'd%i, ",abs($x)); }
#     else {printf FILEOUT ("16'd%i, ",$x); }
#     if ($y < 0) { printf FILEOUT ("-16'd%i, ",abs($y)); }
#     else {printf FILEOUT ("16'd%i, ",$y); }
#     if ($z < 0) { printf FILEOUT ("-16'd%i};",abs($z)); }
#     else {printf FILEOUT ("16'd%i}; \n",$z); }
# }

# print FILEOUT "\n\t\t\tdefault: vertex <= 48'hz;";
# print FILEOUT "\n\t\tendcase \n\t\t";
# print FILEOUT "case (face_addr) \n\t\t\t";

# printf FILEOUT ("12'd0: face <= %i;",($face_length+2));

# for ($j=0 ; $j < $face_length; $j++) {
#     if ($normal_method) {
#     $v1 = $face_ver[$j][0];
#     $v2 = $face_ver[$j][1];
#     $v3 = $face_ver[$j][2];
#     printf FILEOUT ("\n\t\t\t12'd%i: face <=
{24'h%s,",($j+1),@facecolor[$j]);
#     printf FILEOUT (" 12'd%i, 12'd%i, 12'd%i,
",(1+$v1),(1+$v2),(1+$v3));
#     printf FILEOUT ("12'd%i, 12'd%i,
12'd%i};",(1+$v1),(1+$v2),(1+$v3)); }
#     else {
#     $v1 = $face_ver[$j][0];
#     $v2 = $face_ver[$j][1];
#     $v3 = $face_ver[$j][2];
#     printf FILEOUT ("\n\t\t\t12'd%i: face <=
{24'h%s,",($j+1),@facecolor[$j]);
#     printf FILEOUT (" 12'd%i, 12'd%i, 12'd%i,
",(1+$j),($j+1),(1+$j));
#     printf FILEOUT ("12'd%i, 12'd%i,
12'd%i};",(1+$v1),(1+$v2),(1+$v3)); }
# }
# print FILEOUT "\n\t\tdefault: face <= 96'hz;";
# print FILEOUT "\n\t\tendcase \n\t\t";
# print FILEOUT "case (normal_addr) \n\t\t\t";
# print FILEOUT "12'd0: normal <= 0;";

# if ($normal_method) {
#     for ($i=0; $i < $ver_length; $i++) {
```

```
#     $x = $vnormal_avg[$i][0];
#     $y = $vnormal_avg[$i][1];
#     $z = $vnormal_avg[$i][2];
#     printf FILEOUT ("\n\t\t\t12'd%i: normal <= {",($i+1));
#     if ($x < 0) { printf FILEOUT ("-16'd%i, ",abs($x)); }
#     else {printf FILEOUT ("16'd%i, ",$x); }
#     if ($y < 0) { printf FILEOUT ("-16'd%i, ",abs($y)); }
#     else {printf FILEOUT ("16'd%i, ",$y); }
#     if ($z < 0) { printf FILEOUT ("-16'd%i};",abs($z)); }
#     else {printf FILEOUT ("16'd%i};",$z); }
#     }}
# else {
#         for ($i=0; $i < $face_length; $i++) {
#     $x = $vnormal[$i][0];
#     $y = $vnormal[$i][1];
#     $z = $vnormal[$i][2];
#     printf FILEOUT ("\n\t\t\t12'd%i: normal <= {",($i+1));
#     if ($x < 0) { printf FILEOUT ("-16'd%i, ",abs($x)); }
#     else {printf FILEOUT ("16'd%i, ",$x); }
#     if ($y < 0) { printf FILEOUT ("-16'd%i, ",abs($y)); }
#     else {printf FILEOUT ("16'd%i, ",$y); }
#     if ($z < 0) { printf FILEOUT ("-16'd%i};",abs($z)); }
#     else {printf FILEOUT ("16'd%i};",$z); }
#     }}

# print FILEOUT "\n\t\t\tdefault: normal <= 48'hz;";
# print FILEOUT "\n\t\tendcase \n\t";
# print FILEOUT "end \n";
# print FILEOUT "endmodule \n";

$fileout1 = sprintf("%s_vertex_rom",$line[1]);
open(FILEOUT1,sprintf(">%s.coe",$fileout1));
print FILEOUT1 "; $fileout1 \n \n";

print FILEOUT1 "memory_initialization_radix=16; \n";
print FILEOUT1 "memory_initialization_vector= ";
printf FILEOUT1 ("00000000%s, ",dec2hex(2048));

for ($i=0; $i < $ver_length; $i++) {
    $x = $ver[$i][0];
    $y = $ver[$i][1];
    $z = $ver[$i][2];
    printf FILEOUT1 ("%s%s%s",dec2hex($x),dec2hex($y),dec2hex($z));
    if ($i == ($ver_length - 1)) {print FILEOUT1 ";"}
    else {print FILEOUT1 ", "; }
}


close(FILEOUT1);

$fileout2 = sprintf("%s_face_rom",$line[1]);
open(FILEOUT2,sprintf(">%s.coe",$fileout2));
print FILEOUT2 "; $fileout2 \n \n";

print FILEOUT2 "memory_initialization_radix=16; \n";
print FILEOUT2 "memory_initialization_vector= ";
```

```
printf FILEOUT2 ("00000000000000000000%s, ",dec2hex($face_length+2));

 for ($j=0 ; $j < $face_length; $j++) {
        $v1 = substr(dec2hex(1+$face_ver[$j][0]),1);
      $v2 = substr(dec2hex(1+$face_ver[$j][1]),1);
      $v3 = substr(dec2hex(1+$face_ver[$j][2]),1);
      if ($normal_method) {
          printf FILEOUT2
("%s%s%s%s%s%s%s",@facecolor[$j],$v1,$v2,$v3,$v1,$v2,$v3);
      }
      else {
          $temp = dec2hex(1+$j);
          printf FILEOUT2
("%s%s%s%s%s%s%s",@facecolor[$j],$temp,$temp,$temp,$v1,$v2,$v3);
      }
      if ($j == ($face_length - 1)) {print FILEOUT2 ";"}
      else {print FILEOUT2 ", "; }
    }

close FILEOUT2;

$fileout3 = sprintf("%s_normal_rom",$line[1]);

open(FILEOUT3,sprintf(">%s.coe",$fileout3));
print FILEOUT3 "; $fileout3 \n \n";

print FILEOUT3 "memory_initialization_radix=16; \n";
print FILEOUT3 "memory_initialization_vector= ";

print FILEOUT3 "000000000000, ";

if ($normal_method) {
    for ($i=0; $i < $ver_length; $i++) {
      $x = $vnormal_avg[$i][0];
      $y = $vnormal_avg[$i][1];
      $z = $vnormal_avg[$i][2];
      printf FILEOUT3 ("%s%s%s",dec2hex($x),dec2hex($y),dec2hex($z));
      if ($i == ($ver_length - 1)) {print FILEOUT3 ";"}
      else {print FILEOUT3 ", "; }
    }
}
else {
    for ($i=0; $i < $face_length; $i++) {
      $x = $vnormal[$i][0];
      $y = $vnormal[$i][1];
      $z = $vnormal[$i][2];
      printf FILEOUT3 ("%s%s%s",dec2hex($x),dec2hex($y),dec2hex($z));
      if ($i == ($face_length - 1)) {print FILEOUT3 ";"}
      else {print FILEOUT3 ", "; }
    }
    }

close FILEOUT3;

close(MODEL);
```