6.111: Introductory Digital Systems Laboratory

Group #11: Final Project Appendix
User-Friendly Stylus and Video Projection CAD System

Jeremy Schwartz, Paul Peeling, Faraz Ahmad

Contents:

labkit.v modifications by Paul Peeling, Faraz Ahmad, Jeremy Schwartz shown only

Jeremy Schwartz
videotop.v
accum.v, comp_reg.v, comparator.v, data_reg.v, , synchonizer.v, vid_handler.v, videofsm.v

Paul Peeling
command_top.v
area_map.v, command_router.v, draw_circle.v, draw_line.v, draw_point.v, draw_rect.v, move_obj.v, mult2comp.v, radius_calc.v, resize_obj.v, select.v, snap_grid.v, snap_point.v, snapper.v, superplex.v, usersynch.v

Faraz Ahmad
vga.v
control_unit.v, initialise.v, syncgen2.v

Original Code Provided by Nathan Ickes:

labkit.v http://www-mtl.mit.edu/Courses/6.111/labkit/verilog/labkit.v
labkit.ucf http://www-mtl.mit.edu/Courses/6.111/labkit/verilog/labkit.ucf
adv7185init.v http://www-mtl.mit.edu/Courses/6.111/labkit/verilog/004/avtest/adv7185init.v
debounce.v http://www-mtl.mit.edu/Courses/6.111/labkit/verilog/004/avtest/avtest.v
i2c.v http://www-mtl.mit.edu/Courses/6.111/labkit/verilog/004/avtest/i2c.v

```
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////

module labkit (...);

/// Note all previous definitions used by subsequent code
/// have been removed

   ////////////////////////////////////////////////
   ///   DCM for pixel and SRAM clocks
   ////////////////////////////////////////////////


   wire clock27, clock27b;
   wire clkfb_top, reset_line, fpga_clock, reset_out, clock_25mhz;
   wire clk0_top, clk0_bot, lock_top, lock_bot;

   BUFG clock_27mhz_buf (.O(clock27b), .I(clock27));

   DCM pixel_clock_dcm (.CLKFB(clock27b),
                        .CLKIN(clock_27mhz),
                        .RST(1'b0),
                        .CLK0(clock27),
                        .CLKFX(clock_25mhz)
                        );
   // synthesis attribute DLL_FREQUENCY_MODE of pixel_clock_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of pixel_clock_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of pixel_clock_dcm is "TRUE"
   // synthesis attribute DFS_FREQUENCY_MODE of pixel_clock_dcm is "LOW"
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 14
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 13
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of pixel_clock_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of pixel_clock_dcm is 0
   // synthesis attribute clkin_period of pixel_clock_dcm is "37.04ns"

   DCM topdcm (.CLKIN(clock_25mhz), .RST(reset_line), .CLK0(clk0_top),
                            .CLKFB(clkfb_top), .LOCKED(lock_top));
   // synthesis attribute DLL_FREQUENCY_MODE of topdcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of topdcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of topdcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of topdcm is "LOW"
   // synthesis attribute CLKFX_DIVIDE of topdcm is 1
   // synthesis attribute CLKFX_MULTIPLY of topdcm is 1
   // synthesis attribute CLK_FEEDBACK of topdcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of topdcm is "NONE"
   // synthesis attribute PHASE_SHIFT of topdcm is 0
   // synthesis attribute clkin_period of topdcm is "37.04ns"

   DCM botdcm (.CLKIN(clock_25mhz), .RST(reset_line), .CLK0(clk0_bot),
                            .CLKFB(fpga_clock), .LOCKED(lock_bot));
   // synthesis attribute DLL_FREQUENCY_MODE of botdcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of botdcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of botdcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of botdcm is "LOW"
   // synthesis attribute CLKFX_DIVIDE of botdcm is 1
   // synthesis attribute CLKFX_MULTIPLY of botdcm is 1
   // synthesis attribute CLK_FEEDBACK of botdcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of botdcm is "NONE"
   // synthesis attribute PHASE_SHIFT of botdcm is 0
   // synthesis attribute clkin_period of botdcm is "37.04ns"

   IBUF ibufdcm (.I(clock_feedback_in), .O(clkfb_top));
   BUFG btopdcm (.I(clk0_top), .O(clock_feedback_out));
   BUFG bbotdcm (.I(clk0_bot), .O(fpga_clock));

   SRL16 srldcm (.D(1'b0), .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1), .Q(reset_line),
.CLK(clock_27mhz));
```

```
   defparam srldcm.INIT = 16'h000F;

   assign ram0_clk = clock_feedback_out;
   assign ram1_clk = clock_feedback_out;
   assign reset_out = !(lock_top && lock_bot && !reset_line);

////////////////////////////////////////
   ////////////////////////////////////////
   // CAD System
   ////////////////////////////////////////
////////////////////////////////////////

   // Synchronizer for reset button and others
   wire button0_sync, button1_sync;
   usersynch BUTTONS (.clk(fpga_clock), .button({button0, button1}),
                                              .button_sync({button0_sync,
button1_sync}));

   // SRAM 0
   assign ram0_adv_ld = 1'b0;
   assign ram0_cen_b = 1'b0;
   assign ram0_ce_b = 1'b0;
   assign ram0_bwe_b = 4'b0000;
   //assign ram0_oe_b = 1'b0;

   // SRAM 1
   assign ram1_adv_ld = 1'b0; // address comes from input pins
   assign ram1_cen_b = 1'b0; // chip enabled
   assign ram1_ce_b = 1'b0; // chip1 enabled
   assign ram1_oe_b = 1'b0;
   assign ram1_bwe_b = 4'h0;

   wire comb_reset;
   assign comb_reset = (!button0_sync || reset_out);
   wire test_reset;

   wire [18:0] test_address;
   wire [35:0] test_mem_in;
   wire test_mem_write;

   reg [35:0] test_mem_in_clean;
   reg [35:0] vid_mem_in_clean;
   reg [18:0] ram0_address;
   reg [18:0] ram1_address;
   reg ram0_we_b, test_oe, test_oe_delay, test_oe_delay_2;


         wire [18:0] SRAMaddress;
         wire     [8:0] SRAMdata;
         wire [8:0] SRAMdataout;
         wire [3:0] control_unit_state;
         wire OEbar, WEbar, display_busy, CEbar;

         //////////////////////////
         /// Memory flush operation & Grid draw!
         /// Global Reset: test_reset
         //////////////////////////

                  reg flush_done;
                  reg vid_oe;
                  reg vid_mem_write, ram1_we_b, vid_oe_delay;

         always @ (posedge ram0_clk)
                  begin
                           if (comb_reset)
                                    begin
                                             ram0_address <= 0;
                                             ram0_we_b <= 0;
                                             test_mem_in_clean <= 36'h0;
                                             test_oe <= 1;
                                             test_oe_delay <= 0;
                                             flush_done <= 0;
                                             test_oe_delay_2 <= 0;

                                             ram1_address <= 0;
                                             ram1_we_b <= 0;
                                             vid_mem_in_clean <= 36'b0;
```

```
                                        vid_oe <= 1;
                                        vid_oe_delay <= 0;

                                end
                        else if (!flush_done)
                                begin

                if (ram0_address == 19'h7FFFF && ram1_address ==19'h7FFFF)
                                        flush_done <= 1;
                                else
                                        begin
                                flush_done <= 0;
                                ram0_address <= ram0_address + 1;
                                ram1_address <= ram1_address + 1;
                                        end
                                ram0_we_b <= 0;
                                test_mem_in_clean <= 36'h0;
                                test_oe <= 1;
                                test_oe_delay <= 0;
                                test_oe_delay_2 <= 0;

                                ram1_we_b <= 0;
                                vid_mem_in_clean <= 36'b0;
                                vid_oe <= 1;
                                vid_oe_delay <= 0;
                                end
                        else
                                begin
                                        ram0_address <=  test_address;
                                        test_mem_in_clean <= test_mem_in;
                                        ram0_we_b <= !test_mem_write;
                                        test_oe_delay <= !ram0_we_b;
                                        test_oe <= test_oe_delay;
                                        test_oe_delay_2 <= test_oe;


                                        ram1_address <= SRAMaddress;
                                vid_mem_in_clean <= {27'b0, SRAMdataout};
                                        ram1_we_b <= WEbar;
                                        vid_oe_delay <= !ram1_we_b;
                                        vid_oe <= vid_oe_delay;
                                //vid_oe is high when writing - registered
                                end

                end

        assign test_reset = !flush_done;

   ////////////////////////////////////////////////////////////////////////
   //
   // Video Input
   //
   ////////////////////////////////////////////////////////////////////////


// Synthesis Attribute period of clock_27mhz is 37ns;
// Synthesis attribute keep of clock_27mhz is true;

// Synthesis Attribute period of tv_in_line_clock1 is 37ns;
// Synthesis attribute keep of tv_in_line_clock1 is true;


   assign tv_in_fifo_read = 1'b1;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b1;
   assign tv_in_clock = clock_27mhz;
   wire tv_in_reset_b, tv_in_i2c_clock;

   wire svideo;
   assign svideo = 0;  //lock in composite mode
   wire [7:0] state;

   wire reset;
   SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(reset),
                   .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;
```

```
   adv7185init init7185 (reset, clock_27mhz, svideo, tv_in_reset_b,
                           tv_in_i2c_clock, tv_in_i2c_data, state);

         //assign tv_in_reset_b = ~test_reset;

   PULLUP pu_in (.O(tv_in_i2c_data));
//   PULLUP pu_out (.O(tv_out_i2c_data));

   wire [9:0] stylus_x, stylus_y, a_out_x, a_out_y, a_out_z;
   wire [3:0] vidfsm_state, vidh_state;
   wire [19:0] num_passed;
   wire pass_pulse, frame_done;
   wire reset_v;

   assign reset_v = button2;
   videotop V1(tv_in_line_clock1, reset_v, tv_in_ycrcb[19:10], stylus_x,
         stylus_y, vidfsm_state, vidh_state, pass_pulse, frame_done, num_passed,
                                    a_out_x, a_out_y, a_out_z);



         ////////////////////////////////
         /// Camera Interface
         ////////////////////////////////

          wire click_noisy;
          assign click_noisy = !button1_sync;
          wire click_clean, click;

          wire [9:0] pos_x_clicked, pos_y_clicked;

          debounce CLICK_DEBOUNCE (.clock(fpga_clock), .reset(test_reset),
                                    .noisy(click_noisy), .clean(click_clean));

          camera_sync CLICKER (.clk(fpga_clock), .reset(test_reset),
                           .click_in(click_clean), .click_out(click),
                           .stylus_x_in(stylus_x), .stylus_x_out(pos_x_clicked),
                           .stylus_y_in(stylus_y), .stylus_y_out(pos_y_clicked));


         ////////////////////////////////
         ///  CAD System
         ////////////////////////////////
    /*
         testbench_6 TTEST (.clk(fpga_clock), .reset(test_reset),
                                    .mem_address(test_address),
                                    .mem_in(test_mem_in),
                                    .mem_out(ram0_data),
                                    .mem_write(test_mem_write),
                                    .click(click));    */

         wire [3:0] cad_mode;
         assign led = {4'b0, ~cad_mode};

     command_top TSUBSYS (.clk(fpga_clock), .reset(test_reset),
                    .pos_x(pos_x_clicked), .pos_y(pos_y_clicked), .click(click),
                    .mem_address(test_address), .mem_in(test_mem_in),
                    .mem_out(ram0_data), .mem_write(test_mem_write), .mode(cad_mode));

         assign ram0_data = test_oe ? test_mem_in_clean : 36'hZ;
         assign ram0_oe_b = 1'b0;


         ////////////////////////////////////////////
         ///  FIFO Interface:   Command -> Video
         ////////////////////////////////////////////

         reg [27:0] pixel_input;
         wire [27:0] pixel_output;
         wire empty, RE;
         reg [18:0] fifo_address,
                                    fifo_address_delay_a,
                                             fifo_address_delay_b;

         // Only write to the FIFO when the address is in the valid pixel range
         // so object map writes do not go out to the video display
```

```
// test_oe_delay_2 is !ram0_we_b delayed by three cycles, so it is the control signal
// we want

wire fifo_wr_en;
assign fifo_wr_en = test_reset ? 1'b0 :
 ((fifo_address[18:9] < 10'h280) && (fifo_address[8:0] < 9'h1E0) && test_oe_delay_2);

    always @ (posedge fpga_clock)
         begin
         // Need to bring the address back into sync
         // Could have been done more efficiently by
         // outputting sync address and data from
         // the cad modules, but this way is easier to debug
                           fifo_address_delay_a <= ram0_address;
                           //fifo_address_delay_b <= fifo_address_delay_a;
                           fifo_address_delay_b[18:10] <= fifo_address_delay_a[8:0];
                           fifo_address_delay_b[9:0] <= fifo_address_delay_a[18:9];
                           fifo_address <= fifo_address_delay_b;


                  if ((test_reset != 1) && (fifo_wr_en == 1))
                  pixel_input = {test_mem_in[8:0], fifo_address}; // RGB and address
                  else pixel_input = 28'h0;
         end

                  com2vid CVINT
                  (.clk(fpga_clock),
                  .sinit(test_reset),
                  .din(pixel_input),
                  .empty(empty),
                  .dout(pixel_output),
                  .wr_en(fifo_wr_en),
                  .rd_en(RE));



                   ////////////////////////////////////////////
                   //// Video Output
                   ////////////////////////////////////////////////



   assign vga_out_pixel_clock=fpga_clock; // assign vga output clock
/*
wire [9:0] stylus_x_dummy, stylus_y_dummy;
assign stylus_x_dummy = 10'h50;
assign stylus_y_dummy = 10'h50;       */

   wire [7:0] ROMdata;
   wire [10:0] ROMaddress;

vga testmevga(
                           .fpga_clock(fpga_clock),
                           .empty(empty),
                           .display_busy(display_busy),
                           .reset_DAC(reset_DAC),
                           .vga_out_sync_b(vga_out_sync_b),
                           .vga_out_vsync(vga_out_vsync),
                           .vga_out_hsync(vga_out_hsync),
                           .vga_out_blank_b(vga_out_blank_b),
                           .vga_out_red(vga_out_red),
                           .vga_out_green(vga_out_green),
                           .vga_out_blue(vga_out_blue),
                           .reset_sync(test_reset),
                           .frame_buffer_data(pixel_output[27:19]),
                           .address(pixel_output[18:0]),
                           .SRAMaddress(SRAMaddress),
                           .SRAMdata(ram1_data[8:0]),
                           .SRAMdataout(SRAMdataout),
                           .control_unit_state(control_unit_state),
                           .WEbar(WEbar),
                           .CEbar(CEbar),
                           .OEbar(OEbar),
                           .RE(RE),
                           .ROMdata(ROMdata),
                           .ROMaddress(ROMaddress),
```

```
                            .stylus_x(stylus_x), .stylus_y(stylus_y));


   assign ram1_data = vid_oe ? vid_mem_in_clean : 36'hZ;


         ////////////////////////////////////////////////////////
         ////// ROM Interface
         ////////////////////////////////////////////////////////////////

                     toolbar TOOLBAR_ROM(
                                   .clk(fpga_clock),
                                   .addr(ROMaddress),
                                   .dout(ROMdata));

   ////////////////////////////////////////////////
   // Debug Outputs
   ////////////////////////////////////////////////

          /*
         wire [8:0] data_color;
         assign data_color = pos_y_clicked[8:0];
         wire [9:0] data_address;
         assign data_address = pos_x_clicked[9:0];

   assign analyzer1_data = {fifo_wr_en, frame_done,
           data_color, tv_in_reset_b, ram0_we_b, test_reset};
   assign analyzer1_clock = ram0_clk;

   assign analyzer2_data[9:0] = {data_address};
   assign analyzer2_clock = tv_in_line_clock1;*/

         assign user1[0] = vga_out_pixel_clock;
         assign user1[1] = vga_out_hsync;
         assign user1[2] = vga_out_vsync;
         assign user1[3] = vga_out_blank_b;
         assign user1[4] = fpga_clock;

         assign user1[31:5] = 27'b0;
         assign user2 = 36'hZ;
     assign user3 = 36'hZ;


endmodule
```

```
/////// Videotop.v
////// Author: Jeremy Schwartz

module videotop(clk, reset, vid_stream, stylus_x, stylus_y, vidfsm_state, vidh_state,
                  pass_pulse, frame_done, num_passed_pts, a_out_x, a_out_y, a_out_z);
         input clk, reset;
         input[9:0] vid_stream;
//       output[9:0] vid_out;
         output[9:0] stylus_x, stylus_y;
         output [3:0] vidfsm_state, vidh_state;
         output pass_pulse;
         output frame_done;
         output [19:0] num_passed_pts;
         output [9:0] a_out_x, a_out_y, a_out_z;

         // all x & y position values are unsigned.
         wire [9:0] x_pos, y_pos, x_pos2, y_pos2;

         wire reset_sync;
         synchronizer S1(clk, reset, reset_sync);

         //FSM to handle the video input
         wire frame_done, y_reg_le, cr_reg_le, cb_reg_le, pipe_LE, comp_e, cff,
cff_e;
         wire[9:0] vid_out;
         vid_handler V1(clk, reset_sync, vid_stream, frame_done, y_reg_le, cr_reg_le,
                        cb_reg_le, pipe_LE, vid_out, x_pos, y_pos, comp_e,
vidh_state, cff, cff_e);

         //comparator module
         wire[9:0] comp_in_y, comp_in_cr, comp_in_cb;
         wire pass_async;
         comparator C1(comp_in_y, comp_in_cr, comp_in_cb, comp_e, pass_async);

   //FSM for the system
   wire pass_pulse, accum_enable, accum_reset, div_ce, output_LE, raan;
   wire[19:0] num_passed_pts;
   videofsm F1(clk, reset_sync, frame_done, pass_pulse, accum_enable,
                accum_reset, num_passed_pts, div_ce, output_LE, vidfsm_state,
                                                        raan);

         //accumulator and divider modules
         wire[29:0] accum_x, accum_y;
         wire[19:0] remain_x, remain_y;
         wire[29:0] div_out_x, div_out_y;
         assign a_out_x = comp_in_cr[9:0];
         assign a_out_y = comp_in_y[9:0];
         assign a_out_z = comp_in_cb[9:0];

         wire aclr, sclr, rfd_x, rfd_y;  //i don't care about any of these
         assign aclr = 0;
         assign sclr = 0;
         assign ce = 1;
         accum AX(clk, accum_reset, accum_enable, x_pos2, accum_x);
         accum AY(clk, accum_reset, accum_enable, y_pos2, accum_y);
         //divider DX(accum_x, num_passed_pts, div_out_x);
         //divider DY(accum_y, num_passed_pts, div_out_y);
   divider_cg DX(accum_x,    num_passed_pts, div_out_x,   remain_x,
                                                        clk, rfd_x, aclr, sclr, ce);
         divider_cg DY(accum_y,       num_passed_pts, div_out_y,   remain_y,
                                                        clk, rfd_y, aclr, sclr, ce);

         //divider DX(clk, div_start, ready_x, accum_x, num_passed_pts, div_out_x,
remain_x);
         //divider DY(clk, div_start, ready_y, accum_y, num_passed_pts, div_out_y,
remain_y);

         //SYSTEM REGISTERS
         //stylus position output registers
         data_reg  OUTPUT_X(clk, reset_sync, div_out_x[9:0], stylus_x, output_LE);
         data_reg  OUTPUT_Y(clk, reset_sync, div_out_y[9:0], stylus_y, output_LE);

         //registers for video handler output
         data_reg  Y_REG(clk, reset_sync, vid_out, comp_in_y, y_reg_le);
         data_reg  CR_REG(clk, reset_sync, vid_out, comp_in_cr, cr_reg_le);
         data_reg  CB_REG(clk, reset_sync, vid_out, comp_in_cb, cb_reg_le);
```

```
          //pipeline registers (pipe_LE is always 1)
          wire [9:0] x_posA, x_posB, x_posC, y_posA, y_posB, y_posC;
          data_reg  PIPE_REG_X1(clk, reset_sync, x_pos, x_posA, pipe_LE);
          data_reg  PIPE_REG_X2(clk, reset_sync, x_posA, x_posB, pipe_LE);
          data_reg  PIPE_REG_X3(clk, reset_sync, x_posB, x_posC, pipe_LE);
          data_reg  PIPE_REG_X4(clk, reset_sync, x_posC, x_pos2, pipe_LE);

          data_reg  PIPE_REG_Y1(clk, reset_sync, y_pos, y_posA, pipe_LE);
          data_reg  PIPE_REG_Y2(clk, reset_sync, y_posA, y_posB, pipe_LE);
          data_reg  PIPE_REG_Y3(clk, reset_sync, y_posB, y_posC, pipe_LE);
          data_reg  PIPE_REG_Y4(clk, reset_sync, y_posC, y_pos2, pipe_LE);

          //comparator output register
   comp_reg        COMP_REG1(clk, reset_sync, pass_async, pass_pulse, pipe_LE);

endmodule


/// accum.v – Jeremy Schwartz
// x & y position values are unsigned.

module accum(clk, accum_reset, enable, in, out);
          input clk, accum_reset, enable;
          input[9:0] in;
          output[29:0] out;
          reg[29:0] Q;
          wire[29:0] D;

          always @ (posedge clk) begin
                    if (!accum_reset)
                              Q <= 30'b0;
                    else if (!enable)
                              Q <= Q;
                    else
                              Q <= D;
          end

          assign D = Q + in[9:0];
          assign out = Q;
endmodule

/// comp_reg.v – Jeremy Schwartz

module comp_reg(clk, reset, data_in, data_out, LE);

          input clk, reset, LE;
          input data_in;
          output data_out;
          reg data_out;

          always @ (posedge clk) begin
                    if (!reset) data_out <= 0;
                    else if (LE) data_out <= data_in;
                    else data_out <= data_out;
          end
endmodule

/// comparator.v – Jeremy Schwartz


module comparator(vid_in_y, vid_in_cr, vid_in_cb, comp_e, pass_pulse_async);

          input[9:0] vid_in_y, vid_in_cb, vid_in_cr;
          input comp_e;
          output pass_pulse_async;

          // these values represent the filter values
   //we must make sure there is no overflow when
   //we add filter values and sensitivity range
          wire [9:0] filt_y, filt_cr, filt_cb;
          assign filt_y = 400;
          assign filt_cr = 775;
          assign filt_cb = 320;
          // this value represents the range of allowance of the filter.
          parameter RANGE = 50;
          assign pass_pulse_async = comp_e ? ((vid_in_y <= filt_y + RANGE) &&
                                        (vid_in_y >= filt_y - RANGE) &&
```

```
                                        (vid_in_cr <= filt_cr + RANGE) &&
                                        (vid_in_cr >= filt_cr - RANGE) &&
                                        (vid_in_cb <= filt_cb + RANGE) &&
                                        (vid_in_cb >= filt_cb - RANGE)) : 0;

endmodule

/// data_reg.v – Jeremy Schwartz

module data_reg(clk, reset, data_in, data_out, LE);

          input clk, reset, LE;
          input [9:0] data_in;
          output [9:0] data_out;
          reg [9:0] data_out;

          always @ (posedge clk) begin
                    if (!reset) data_out <= 0;
                    else if (LE) data_out <= data_in;
                    else data_out <= data_out;
          end
endmodule

/// synchronizer.v – Jeremy Schwartz

module synchronizer(clk, reset, reset_sync);

          input clk, reset;
          output reset_sync;

          reg reset_int;
          reg reset_sync;
          always @ (negedge clk) begin
                    reset_int <= reset;
                    reset_sync <= reset_int;
          end
endmodule



/// vid_handler.v – Jeremy Schwartz

//*************************************************************
// VID HANDLER
//        this FSM handles the video input coming to the camera.

module vid_handler(vid_clk, reset_sync, vid_stream, frame_done, y_reg_le,
                                                    cr_reg_le, cb_reg_le,
pipe_le, vid_out, x_pos, y_pos, comp_e, state,
                                                    came_from_frame, cff_enable);

          input [9:0] vid_stream;
          input vid_clk, reset_sync;
          output frame_done, comp_e;
          output y_reg_le, cr_reg_le, cb_reg_le, pipe_le;
          output [9:0] x_pos, y_pos;
          output [9:0] vid_out;
          output [3:0] state;
          output came_from_frame, cff_enable;

          reg[3:0] state, next;

          //states defined with parameter keyword

          parameter idle = 0;
          parameter time_check1 = 1;
          parameter time_check2 = 2;
          parameter XY_check = 3;
          parameter store_cb = 4;
          parameter store_y1 = 5;
          parameter store_cr = 6;
          parameter store_y2 = 7;
          parameter error = 4'b1000;
//        parameter wait1 = 0'b1001;

   reg cb_reg_le, cb_reg_le_int, y_reg_le, y_reg_le_int;
   reg cr_reg_le, cr_reg_le_int, pipe_le, pipe_le_int;
```

```
   reg y_inc, x_inc, y_zero, x_zero, frame_done, frame_done_int;
   reg comp_e, comp_e_int;

          reg came_from_frame;        //this is a memory variable used to check if
we've come
          reg came_from_frame_int;      //from active video or not.
          reg cff_enable;

   reg[9:0] x_pos, y_pos;
   reg [9:0] vid_out;

          //state register defined with sequential always block

          always @ (posedge vid_clk or negedge reset_sync) begin
                    if (!reset_sync) begin
             state <= idle;
                              vid_out <= 10'b0;
                              cb_reg_le <= 1;
                              cr_reg_le <= 1;
                              y_reg_le <= 1;
                              pipe_le <= 1;
                              frame_done <= 0;
                              comp_e <=0;
                              x_pos <= 0;
                              y_pos <= 0;
                              came_from_frame <= 0;
                    end
                    else begin

                              state <= next;
                              pipe_le <= pipe_le_int;
                frame_done <= frame_done_int;
                    comp_e <= comp_e_int;

                              if (cff_enable == 1) came_from_frame <=
came_from_frame_int;
                              else came_from_frame <= came_from_frame;

                              if (x_zero == 1) x_pos <= 0;
                              else
                                        x_pos <= x_pos + x_inc;
                if (y_zero == 1) y_pos <= 0;
                else
                y_pos <= y_pos + y_inc + y_inc;

                              //we reset the video registers while we do the time
check.
                              if (next == time_check2) begin
                                        vid_out <= 10'B0;
                                        cb_reg_le <= 1;
                              cr_reg_le <= 1;
                                 y_reg_le <= 1;
                              end
                              else begin
                                        vid_out <= vid_stream;
                                 cb_reg_le <= cb_reg_le_int;
                              cr_reg_le <= cr_reg_le_int;
                                 y_reg_le <= y_reg_le_int;
                              end
                    end
          end

          // next state logic within a combinational always block

    always @ (state or vid_out) begin
                    //defaults
                    y_inc = 0; x_inc = 0; cb_reg_le_int = 0; cr_reg_le_int = 0;
                    y_reg_le_int = 0; pipe_le_int = 1; frame_done_int = 0;
                    x_zero = 0;  y_zero = 0; comp_e_int = 0;
                    came_from_frame_int = 0; cff_enable = 0;

                    case (state)
                              idle: begin
                                        if (vid_out == 10'h3FF)
                                                  next = time_check1;
                                        else next = idle;
                              end
```

```
                                time_check1: begin
                                        if (vid_out == 10'h000)
                                                next = time_check2;
                                        else next = error;
                                end
                                time_check2: begin
                                        if (vid_out == 10'h000)
                                                next = XY_check;
                                        else next = error;
                                end
                                XY_check: begin

                // I believe XY = {1 F V H P3 P2 P1 P0 0 0}

                // F indicates Field (odd or even)

                // V indicates vertical blanking

                // H indicates horizontal blanking

                                        if (vid_out[6] == 1) begin            //
indicates horizontal blanking
                                                next = idle;
                                                y_inc = 1;
                                                x_zero = 1;
                                        end
                                        else if (vid_out[7] == 1)             //
indicates vertical blanking
                                                begin
                                                   x_zero = 1;
                                                   y_zero = 1;
                                                        if (came_from_frame == 1)
begin
                                                                frame_done_int = 1;
                                                                came_from_frame_int
= 0;
                                                                cff_enable = 1;
                                                        end
                                                        next = idle;
                                                end
                                        else
                                                begin
                                                        next = store_y1;
// we jump to store_y1 because the signals are
                                                end

                                        //how important are the parity bits?  how common
are errors?
                                end
//                              wait1: next = store_cb;
                                store_cb: begin
                                        comp_e_int = 1;
                                        cb_reg_le_int = 1;
                                        next = store_y1;
                                        x_inc = 1;
                                end
                                store_y1: begin
                                        if (vid_out == 10'h3FF)
                                                next = time_check1;
                                        else begin
                                                comp_e_int = 1;
                                                next = store_cr;
                                                y_reg_le_int = 1;
                                        end
                                end
                                store_cr: begin
                                        comp_e_int = 1;
                                        cr_reg_le_int = 1;
                                        next = store_y2;
                                        x_inc = 1;
                                        came_from_frame_int = 1;
                                        cff_enable = 1;
                                end
                                store_y2: begin
                                        comp_e_int = 1;
                                        y_reg_le_int = 1;
                                        next = store_cb;
```

```
                        end
                error: begin
                                next = idle;
                        end
                        default: begin
                                next = idle;
                        end
        endcase
        end

endmodule



/// videofsm.v – Jeremy Schwartz
//*************************************************************
// VIDEO FSM
//       this is the FSM for the video input module.  It uses the basic
// template for FSM's given in class.

module videofsm(clk, reset_sync, frame_done, pass_pulse,
                                                        accum_enable, accum_reset,
num_passed,
                                                        divide_ce, output_LE, state,
reset_accum_and_numpassed);

        input clk, reset_sync, frame_done, pass_pulse;
        output accum_enable, accum_reset, output_LE, divide_ce;
        output[19:0] num_passed;
        output[3:0] state;
        output reset_accum_and_numpassed;
        reg[3:0] state, next;

        //states defined with parameter keyword

        parameter idle = 0;
        parameter wait_for_pass = 1;
        parameter store_pass = 2;
        parameter divide = 3;
        parameter register_output = 4;
        parameter reset_stuff = 5;
        parameter wait1 = 6;

   reg [19:0] num_passed;
   reg accum_reset, num_passed_inc, accum_enable, accum_enable_int;
   reg output_LE, output_LE_int, reset_accum_and_numpassed;
   reg divide_ce, divide_ce_int, divide_done_inc;
        reg [5:0] divide_done;

        //state register defined with sequential always block

        always @ (posedge clk or negedge reset_sync) begin
      if (!reset_sync) begin
              state <= idle; accum_enable <= 0; divide_ce <= 0;
                    output_LE <= 0; num_passed <= 20'b0; accum_reset <= 0;
                        divide_done <= 0;
                end
      else begin
         state <= next; accum_enable <= accum_enable_int;
         output_LE <= output_LE_int; divide_ce <= divide_ce_int;

                            if (reset_accum_and_numpassed) begin
                        num_passed <= 20'b0;
                        accum_reset <= 0;
                            divide_done <= 0;
                            end
                    else begin
                        num_passed <= num_passed + num_passed_inc;
                        accum_reset <= 1;
                                    divide_done <= divide_done + divide_done_inc;
                    end
                end
    end

        // next state logic within a combinational always block

    always @ (state or frame_done or pass_pulse or divide_done) begin
```

```
                //defaults
                num_passed_inc = 0; accum_enable_int = 0;
                output_LE_int = 0; reset_accum_and_numpassed = 0;
                divide_ce_int = 0; divide_done_inc = 0;
        case (state)
                        idle: begin
                                next = wait_for_pass;
                        end
                        wait_for_pass: begin
                                if (frame_done) next = divide;
                                else if (pass_pulse)
                                        next = store_pass;
                                else next = wait_for_pass;
                        end
                        store_pass: begin
                                num_passed_inc = 1;
                                accum_enable_int = 1;
                                if (frame_done) next = divide;
                                else next = wait_for_pass;
                        end
                        divide: begin
                                divide_ce_int = 1;
                                divide_done_inc = 1;
                                if (divide_done >= 0)                                   //right n
                                        next = register_output;
            //time frame_done pulsed!
                                else next = divide;
                        end
                        register_output: begin
                                output_LE_int = 1;
                                next = wait1;               // just to be sure
                        end
                        wait1: next = reset_stuff;
                        reset_stuff: begin
                                reset_accum_and_numpassed = 1;
                                next = wait_for_pass;
                end
           default:
                                next = idle;
           endcase
           end
endmodule
```

```
/// command_top.v  - Paul Peeling

// Top Module for Command Subsystem

module command_top
                  (clk, reset,
                   pos_x, pos_y, click,
                   mem_address, mem_in, mem_out, mem_write, mode);

input clk, reset, click;
input [9:0] pos_x;
input [8:0] pos_y;
input [35:0] mem_out;

output [18:0] mem_address;
output [35:0] mem_in;
output mem_write;
output [3:0] mode;

wire [3:0] mode;

wire display_ready;
wire [18:0] mem_address, position_address;
wire [35:0] mem_in;
wire mem_write;
wire [3:0] command;
wire click_snap, click_comm;

wire [9:0] pos_x_sp, pos_x_sg, pos_x_snapped;
wire [8:0] pos_y_sp, pos_y_sg, pos_y_snapped;

wire dp_next, dp_busy;
wire dr_next, dr_busy;
wire dl_next, dl_busy;
wire dc_next, dc_busy;
wire sg_start, sg_busy;
wire sp_start, sp_busy;
wire sel_start, sel_busy, sel_draw;
wire [2:0] sel_click;
wire move_start, move_busy, move_del, move_draw, move_sel;
wire resize_start, resize_busy, resize_del, resize_draw, resize_sel;

assign sel_draw = (dp_busy || dr_busy || dl_busy || dc_busy);

wire dp_we, dr_we, dl_we, dc_we;
wire [18:0] dp_add, dr_add, dl_add, dc_add, sp_add, sel_add;
wire [35:0] dp_data, dr_data, dl_data, dc_data, A, B, C, D;

wire [2:0] sel_obj_type;
wire [8:0] color, sel_color;
wire [9:0] obj_num, sel_obj_num;
wire obj_inc, obj_inc_dp, obj_inc_dr, obj_inc_dl, obj_inc_dc;
assign obj_inc = obj_inc_dp || obj_inc_dr || obj_inc_dl || obj_inc_dc;

wire nullify;
assign nullify = ((command == 8) || move_del || resize_del);      // Delete command
wire color_change;
assign color_change = (command > 10);

wire snap_force_mem;

// Module declarations

snapper TSNAP (.clk(clk), .reset(reset), .pos_x_in(pos_x), .pos_y_in(pos_y),
                            .pos_x_out(pos_x_snapped), .pos_y_out(pos_y_snapped),
                  .click_in(click), .click_out(click_snap), .command(command),
                                    .sg_start(sg_start), .sg_busy(sg_busy),
                             .pos_x_sg(pos_x_sg), .pos_y_sg(pos_y_sg),
                            .sp_start(sp_start), .sp_busy(sp_busy),
                            .pos_x_sp(pos_x_sp), .pos_y_sp(pos_y_sp),
                                                 .snap_force_mem(snap_force_mem));

snap_grid TSG (.clk(clk), .reset(reset), .start(sg_start), .busy(sg_busy),
                  .pos_x_in(pos_x), .pos_y_in(pos_y),
                  .pos_x_out(pos_x_sg), .pos_y_out(pos_y_sg));
```

```
snap_point TSP
                  (.clk(clk), .reset(reset), .start(sp_start), .busy(sp_busy),
.pos_x_in(pos_x), .pos_y_in(pos_y),   .pos_x_out(pos_x_sp), .pos_y_out(pos_y_sp),
.mem_get(mem_out), .mem_add(sp_add));

area_map TAREA (.clk(clk), .reset(reset),
           .click(click_snap), .click_pass(click_comm), .pos_x(pos_x_snapped),
                             .pos_y(pos_y_snapped), .command(command),
          .position(position_address), .mode(mode));

command_router TCOMMR
                   (.clk(clk), .reset(reset), .click(click_comm),
                    .command(command),
                    .dp_next(dp_next), .dp_busy(dp_busy),
                    .dr_next(dr_next), .dr_busy(dr_busy),
                    .dl_next(dl_next), .dl_busy(dl_busy),
                    .dc_next(dc_next), .dc_busy(dc_busy),
                    .sel_next(sel_start), .sel_busy(sel_busy),
                    .color(color), .obj_num(obj_num),
                     .obj_inc(obj_inc),
                     .move_start(move_start), .move_busy(move_busy),
                     .resize_start(resize_start), .resize_busy(resize_busy));

superplex TSPLEX
                   (.clk(clk), .command(command),
                    .mem_write(mem_write), .memory_in(mem_in),
                    .memory_address(mem_address),
                    .dp_we(dp_we), .dp_data(dp_data), .dp_add(dp_add),
                    .dr_we(dr_we), .dr_data(dr_data), .dr_add(dr_add),
                    .dl_we(dl_we), .dl_data(dl_data), .dl_add(dl_add),
                    .dc_we(dc_we), .dc_data(dc_data), .dc_add(dc_add),
                    .sp_add(sp_add),
                    .sel_busy(sel_draw), .obj_type(sel_obj_type), .sel_add(sel_add),
                    .snap_force_mem(snap_force_mem));

draw_point TDP
          (.clk(clk), .reset(reset),
           .next(dp_next), .busy(dp_busy),
           .write(dp_we),
           .pos_in(position_address),
           .data_out(dp_data), .add_out(dp_add),
           .color(color), .obj_num(obj_num), .obj_inc(obj_inc_dp),
           .sel_click(sel_click), .sel_obj_num(sel_obj_num),
           .obj_type(sel_obj_type), .A(A),
            .sel_color(sel_color), .color_change(color_change),
            .move_draw(move_draw));

draw_rect TDR
          (.clk(clk), .reset(reset),
           .next(dr_next), .busy(dr_busy),
           .write(dr_we),
           .pos_in(position_address),
           .data_out(dr_data), .add_out(dr_add),
           .color(color), .obj_num(obj_num), .obj_inc(obj_inc_dr),
           .sel_click(sel_click), .sel_obj_num(sel_obj_num),
           .obj_type(sel_obj_type), .A(A), .B(B),
            .sel_color(sel_color), .color_change(color_change),
            .move_draw(move_draw), .resize_draw(resize_draw));

draw_line TDL
          (.clk(clk), .reset(reset),
           .next(dl_next), .busy(dl_busy),
           .write(dl_we),
           .pos_in(position_address),
           .data_out(dl_data), .add_out(dl_add), .color(color),
           .obj_num(obj_num), .obj_inc(obj_inc_dl),
           .sel_click(sel_click), .sel_obj_num(sel_obj_num),
           .obj_type(sel_obj_type), .A(A), .B(B),
            .sel_color(sel_color), .color_change(color_change),
            .move_draw(move_draw), .resize_draw(resize_draw));

draw_circle TDC
          (.clk(clk), .reset(reset),
           .next(dc_next), .busy(dc_busy),
           .write(dc_we),
           .pos_in(position_address),
           .data_out(dc_data), .add_out(dc_add),
```

```
          .color(color), .obj_num(obj_num), .obj_inc(obj_inc_dc),
          .sel_click(sel_click), .sel_obj_num(sel_obj_num),
          .obj_type(sel_obj_type), .A(A), .B(B),
           .sel_color(sel_color), .color_change(color_change),
           .move_draw(move_draw), .resize_draw(resize_draw));

select TSELECT
         (.clk(clk), .reset(reset), .next(sel_start), .busy(sel_busy),
          .pos_in(position_address), .info_in(mem_out), .add_out(sel_add),
          .A(A), .B(B), .C(C), .D(D), .nullify(nullify),
          .sel_next(sel_click), .sel_busy(sel_draw),
          .obj_type(sel_obj_type), .obj_num(sel_obj_num),
           .color(sel_color), .move_del(move_del), .move_sel(move_sel),
           .color_change(color_change), .new_color(color));

move_obj TMOVE
   (.clk(clk), .reset(reset), .start(move_start), .busy(move_busy),
    .move_del(move_del), .move_draw(move_draw), .move_sel(move_sel),
    .sel_busy(sel_busy), .sel_draw(sel_draw));

resize_obj TRESIZE
   (.clk(clk), .reset(reset), .start(resize_start), .busy(resize_busy),
    .resize_del(resize_del), .resize_draw(resize_draw), .resize_sel(resize_sel),
    .sel_busy(sel_busy), .sel_draw(sel_draw));

endmodule

/// area_map.v – Paul Peeling

// Area and command map
// Description:    Maps the pointer position
//                                    to either a command or drawing position
//                                    and outputs on a click

// 640 x 480 resolution


module area_map
                  (clk, reset,
                   click, click_pass, pos_x, pos_y,
                   command, position, mode);

input clk, reset, click;
input [9:0] pos_x;          // 0 to 639
input [8:0] pos_y;          // 0 to 479

output [3:0] command;
output [18:0] position;
output click_pass;
output [3:0] mode;

reg [3:0] command;
reg [3:0] mode;                       // MODE REGISTER
reg click_pass;                       // Delay so clicks can be detected by command
router

wire [18:0] position;
assign position = {pos_x, pos_y};

always @ (posedge clk)
         begin

                  click_pass <= click;

                  if (reset)
                           begin
                  command <= 0;               // 0 is the default no_op command
                  mode <= 0;
                           end
                  else if (click)
                           begin

                                    // Toolbars
                  if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
                   && (pos_y >= 10'h0) && (pos_y < 10'h20))
         // Draw point button
                                    begin
```

```
                                    mode <= 4'h1;               // Draw point mode
                                    command <= 4'd0;   // No-operation
                                    end
                else if ((pos_x >= 10'h0) && (pos_x <= 10'h20) && (pos_y >= 10'h20) &&
                (pos_y < 10'h40))       // Draw rect button
                                            begin
                                    mode <= 4'h2;               // Draw rect mode
                                    command <= 4'd0;   // No-operation
                                            end
            else if ((pos_x >= 10'h0) && (pos_x <= 10'h20) && (pos_y >= 10'h40) &&
(pos_y < 10'h60))  // Draw line button
                                            begin
                                    mode <= 4'h3;               // Draw line mode
                                    command <= 4'd0;   // No-operation
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'h60) && (pos_y < 10'h80))       // Draw circle button
                                    begin
                                    mode <= 4'h4;               // Draw circle mode
                                    command <= 4'd0;   // No-operation
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'h80) && (pos_y < 10'hA0))       // Snap to grid button
                                    begin
                                                    mode <= mode;
                                                    command <= 4'd5;
            // Tell snapper module to activate
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'hA0) && (pos_y < 10'hC0))       // Snap to point button
                                            begin
                                                    mode <= mode;
                                                    command <= 4'd6;
            // Tell snapper module to activate
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'hC0) && (pos_y < 10'hE0))       // Select button
                                            begin
                                            mode <= 4'h7;
            // Select mode
                                command <= 4'd0;            // No-operation
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'hE0) && (pos_y < 10'h100))      // Delete button
                                            begin
                                            mode <= 4'h8;
            // Select mode
                                                    command <= 4'd8;
            // Delete currently selected object
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'h100) && (pos_y < 10'h120))     // Move button
                                            begin
                                            mode <= 4'h9;
            // Select mode
                                                    command <= 4'd0;
            // Move point
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'h120) && (pos_y < 10'h140))     // Resize button
                                            begin
                                            mode <= 4'hA;
            // Select mode
                                                    command <= 4'd0;
            // Resize object
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'h140) && (pos_y < 10'h160))     // Red button
                                            begin
                                            mode <= 4'hB;
            // Select mode
                                                    command <= 4'd11;
            // Red
                                            end
                                    else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'h160) && (pos_y < 10'h180))     // Green button
                                            begin
```

```
                                                mode <= 4'hC;
          // Select mode
                                                        command <= 4'd12;
          // Green
                                                end
                                        else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'h180) && (pos_y < 10'h1A0))     // Blue button
                                                begin
                                                mode <= 4'hD;
          // Select mode
                                                        command <= 4'd13;
          // Blue
                                                end
                                        else if ((pos_x >= 10'h0) && (pos_x <= 10'h20)
&& (pos_y >= 10'h1A0) && (pos_y < 10'h1C0))     // White button
                                                begin
                                                mode <= 4'hE;
          // Select mode
                                                        command <= 4'd14;
          // White
                                                end

                                        // Drawing area click

                                        else
                                                begin
                                                        case (mode)

                                                0: command <= 4'h0;// No_op
                                                1: command <= 4'h1;// Draw_point
                                                2: command <= 4'h2;// Draw_rect
                                                3: command <= 4'h3;// Draw_line
                                                4: command <= 4'h4;// Draw_circle
                                                7: command <= 4'h7;// Select - normal
                                                9: command <= 4'h9; // Move
                                                10: command <= 4'hA; // Resize

                                                default: command <= 4'h0;    // No_op
                                                        endcase
                                                        mode <= mode;
                                                end
                                        end

                    else begin command <= command; mode <= mode; end         // No_op
          end

endmodule

/// camera_sync.v

// Synchronizer with camera module - takes in position and click inputs
// and outputs pulsed click and registered positions

module camera_sync(clk, reset, click_in, click_out,
                    stylus_x_in, stylus_x_out, stylus_y_in, stylus_y_out);

          input clk, reset, click_in;
          input [9:0] stylus_x_in, stylus_y_in;

          output click_out;
          output [9:0] stylus_x_out;
          output [8:0] stylus_y_out;
          reg click_bar, click_out;
          reg [9:0] stylus_x_out;
          reg [8:0] stylus_y_out;

          always @ (posedge clk)
                    begin
                              if (reset) click_bar <= 1;
                              else click_bar <= !click_in;

                              if (stylus_x_in != 10'h3FF) stylus_x_out <= stylus_x_in;
                              else stylus_x_out <= stylus_x_out;
                              if (stylus_y_in != 10'h3FF) stylus_y_out <=
stylus_y_in[8:0];
                              else stylus_y_out <= stylus_y_out;
```

```
                        click_out <= (click_in && click_bar);
                end

endmodule

/// command_router.v – Paul Peeling

// Command router
// Description:    Controlling module to run commands

module command_router
                (clk, reset, click, command,
                 dp_next, dp_busy,                  // draw_point
                 dr_next, dr_busy,                  // draw_rect
                 dl_next, dl_busy,                  // draw_line
                 dc_next, dc_busy,                  // draw_circle
                 sel_next, sel_busy,                // select
                 color, obj_num, obj_inc,
                 move_start, move_busy,   // move_object
                 resize_start, resize_busy);   // resize_object

input clk, reset, click, obj_inc;
input [3:0] command;
input dp_busy, dr_busy, dl_busy, dc_busy, sel_busy, move_busy, resize_busy;

output dp_next, dr_next, dl_next, dc_next, sel_next, move_start, resize_start;
reg dp_next, dr_next, dl_next, dc_next, sel_next, move_start, resize_start;

output [8:0] color;         // 9 bits of color provides for 512 colors - adequate!
reg [8:0] color;

output [9:0] obj_num;       // 1024 possible objects
reg [9:0] obj_num;

reg command_busy;           // Generic next and busy signals which will be
multiplexed
reg command_next;

reg [1:0] state, next;

parameter IDLE = 0;
parameter COMMAND_READY = 1;
parameter COMMAND_TRIGGER = 2;
parameter COMMAND_DONE = 3;


// Colors - 9 bits
parameter WHITE = 9'o777;
parameter RED = 9'o700;
parameter GREEN = 9'o070;
parameter BLUE = 9'o007;


always @ (posedge clk)
         begin

                if (reset)
                        begin
                                state <= 0;
                        color <= WHITE;             // White is the default color
                                obj_num <= 0;
                        end
                else
                        begin
                                state <= next;
                        end

                if (obj_inc) obj_num <= obj_num + 1;


// Command Routing map for next and busy signals

                case (command)

                        0: command_busy <= 0;
         // No-op
```

```
1:
        begin
                dp_next <= command_next;
                command_busy <= dp_busy;
        end

2:
        begin
                dr_next <= command_next;
                command_busy <= dr_busy;
        end

3:
        begin
                dl_next <= command_next;
                command_busy <= dl_busy;
        end

4:
        begin
                dc_next <= command_next;
                command_busy <= dc_busy;
        end

7:
        begin
                sel_next <= command_next;
                command_busy <= sel_busy;
        end

8:
        begin
                sel_next <= command_next;
                command_busy <= sel_busy;
        end

9:
        begin
                move_start <= command_next;
                command_busy <= move_busy;
        end

10:
        begin
                resize_start <= command_next;
                command_busy <= resize_busy;
        end


11:
        begin
                sel_next <= command_next;
                command_busy <= sel_busy;
                color <= RED;
        end

12:
        begin
                sel_next <= command_next;
                command_busy <= sel_busy;
                color <= GREEN;
        end

13:
        begin
                sel_next <= command_next;
                command_busy <= sel_busy;
                color <= BLUE;
        end

14:
        begin
                sel_next <= command_next;
                command_busy <= sel_busy;
                color <= WHITE;
        end
```

```
                        default: command_busy <= 0;

                endcase

        end

always @ (state or command or click or command_busy)
        begin
                if ((command != 0) && (command != 5) && (command != 6))
        // No-op, and snap commands
                        begin
                                case (state)

                                        IDLE:
                                                begin
                                                command_next = 0;
                                        if (click) next = COMMAND_READY;
                                                        else next = IDLE;
                                                end

                                        COMMAND_READY:
                                                begin
                                                command_next = 1;
                                        if (command_busy) next = COMMAND_READY;
                                                else next = COMMAND_TRIGGER;
                                                end

                                        COMMAND_TRIGGER:
                                                begin
                                                        command_next = 0;
                                        if (command_busy) next = COMMAND_DONE;
                                                else next = COMMAND_TRIGGER;
                                                end

                                        COMMAND_DONE:
                                                begin
                                                        command_next = 0;
                                        if (command_busy) next = COMMAND_DONE;
                                                        else next = IDLE;
                                                end

                                        default: next = IDLE;

                                endcase
                        end
                else next = IDLE;
        end
endmodule


/// draw_circle.v – Paul Peeling

//COMMAND: draw_circle
// Description: Draws a circle using first point as a center
//                                      and the second as a radius indicator

module draw_circle
         (clk, reset, next, busy,
          write, pos_in, data_out, add_out, color, obj_num, obj_inc,
          sel_click, sel_obj_num, obj_type, A, B, sel_color, color_change,
          move_draw, resize_draw);

input clk, reset, next;
input [2:0] sel_click;
input [18:0] pos_in;
input [8:0] color, sel_color;
input color_change, move_draw, resize_draw;
input [9:0] obj_num, sel_obj_num;
input [2:0] obj_type;
input [35:0] A, B;          // A is center, B is radius reference

output busy;
output write;                           // WE
output [35:0] data_out;     // Point identifier
output [18:0] add_out;

reg [2:0] mode, moden;
```

```
reg [3:0] state, state_next;

reg [9:0] cx, cxn, cy, cyn;                              // Center coordinates
reg [9:0] rx, rxn, ry, ryn;                              // Radius-point coords
reg [9:0] pos_x, pos_xn, pos_y, pos_yn;         // Current positions
reg [9:0] x1, x1n, y1, y1n, p, pn;    // Offset positions, and algorithm params
reg [3:0] oct, octn;
wire [7:0] rad_calc;
reg [9:0] r;                // Zero padded so we have a ten-bit radius

reg [1:0] point_type;

// States
parameter IDLE = 0;
parameter CENTER_POINT = 1;
parameter IDLE2 = 2;
parameter RESIZE_LOAD = 3;      // Need to calculate new radius
parameter RADIUS_READY = 4;
parameter RADIUS_TRIGGER = 5;
parameter RADIUS_DONE = 6;
parameter SELECT_LOAD = 7;
parameter DRAW_CENTER = 8;
parameter CIRCLE_DRAW = 9;
parameter ADMIN_1 = 10;
parameter ADMIN_2 = 11;
parameter ADMIN_3 = 12;
parameter ADMIN_4 = 13;
parameter DATA_WAIT_1 = 14;                     // Pipeline data wait states for busy
signal
parameter DATA_WAIT_2 = 15;

parameter SELECT_COLOR = 9'o770;   // Yellow is the selection color

wire rc_start;
assign rc_start = (state == RADIUS_TRIGGER);
wire rc_busy;

output obj_inc;
wire obj_inc;
assign obj_inc = ((state == DATA_WAIT_1) && (mode == 0));

// Radius calculator (8-bit)
radius_calc TRADC
         (.clk(clk), .reset(reset), .cx(cx[7:0]), .cy(cy[7:0]),
 .rx(rx[7:0]), .ry(ry[7:0]), .radius(rad_calc), .start(rc_start), .busy(rc_busy));

reg busy;

wire write;
assign write = ((state == CIRCLE_DRAW)
           || (state == ADMIN_1) || (state == ADMIN_2) || (state == ADMIN_3));

reg [35:0] data_start;
reg [35:0] data_late, data_late_2;
reg [35:0] data_out;                                                  // Pipelined data

reg  [18:0] add_out;

reg [3:0] data_type;         // circle centre: 6, radius ref. 7, normal 8
always @ (point_type)
         begin
                  case (point_type)
                           0: data_type = 0;
                           1: data_type = 6;
                           2: data_type = 7;
                           3: data_type = 8;
                           default: data_type = 0;
                  endcase
         end

always @ (posedge clk)
         begin
                  if (reset)
                           begin
                                    state <= IDLE;
                           end
                  else state <= state_next;
```

```
                    cx <= cxn; cy <= cyn; rx <= rxn; ry <= ryn;
                    pos_x <= pos_xn; pos_y <= pos_yn;
                    x1 <= x1n; y1 <= y1n; p <= pn; oct <= octn;

                    data_late <= data_start;
                    data_late_2 <= data_late;
                    data_out <= data_late_2;

                    mode <= moden;
                    busy <= ((state != IDLE) && (state != IDLE2));

                    if (state == SELECT_LOAD) r <= B[21:12];
                    else if (state == RADIUS_DONE) r <= {2'b00, rad_calc};
          end

always @ (state or next or sel_click or rc_busy or oct or move_draw
                     or resize_draw or color_change)
          begin

                    cxn = cx; cyn = cy;rxn = rx; ryn = ry;
                    pos_xn = pos_x; pos_yn = pos_y;
                    x1n = x1; y1n = y1; pn = p; octn = oct;
                    point_type = 0;             // No point
                    add_out = 0;
                    data_start = 0;
                    moden = mode;

                    case (state)

                              IDLE:
                                        begin
                                                  if (next)
                                                            begin
                                                            state_next = CENTER_POINT;
                                                                      moden = 0;
                                                            end
                                                  else if ((sel_click == 1) && (obj_type == 4))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                                      moden = 1;
                                                            end
                                                  else if ((sel_click == 2) && (obj_type == 4))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 2;      // Deselect
                                                            end
                                                  else if ((sel_click == 3) && (obj_type == 4))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 3;      // Delete
                                                            end
                                                            else if (move_draw && (obj_type == 4))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 4;      // Move
                                                            end
                                        else if (resize_draw && (obj_type == 4))
                                                            state_next = RESIZE_LOAD;
                                                            moden = 5;      // Resize
                                                            end
                                                            else state_next = IDLE;
                                        end

                              CENTER_POINT:
                                        begin
                                                  state_next = IDLE2;
                                        cxn = pos_in[18:9]; cyn = {1'b0, pos_in[9:0]};
                                        pos_xn = pos_in[18:9]; pos_yn = pos_in[9:0];
                                        end

                              IDLE2:
                                        begin
                                                  if (next) state_next = RADIUS_READY;
                                                  else state_next = IDLE2;
                                        end
```

```
                    RESIZE_LOAD:
                       begin
                           cxn = A[21:12]; cyn = A[11:3];
                           state_next = RADIUS_READY;
                       end

                    RADIUS_READY:               // Square root calculation
                             begin
                             rxn = pos_in[18:9]; ryn = {1'b0, pos_in[8:0]};
                             if (rc_busy) state_next = RADIUS_READY;
                                      else state_next = RADIUS_TRIGGER;
                             end

                    RADIUS_TRIGGER:
                             begin
                                      if (rc_busy) state_next = RADIUS_DONE;
                                      else state_next = RADIUS_TRIGGER;
                             end

                    RADIUS_DONE:
                             begin
                                      if (rc_busy) state_next = RADIUS_DONE;
                                      else state_next = ADMIN_1;
                    x1n = 0; y1n = r; pn = 3 + (~(2*r) + 1); octn = 0;
                             end

                    SELECT_LOAD:
                       begin
                if (mode == 4) begin cxn = pos_in[18:9]; cyn = pos_in[8:0]; end
                             else begin cxn = A[21:12]; cyn = A[11:3]; end
                             // Radius was stored in B[21:12];
          x1n = 0; y1n = B[21:12]; pn = 3 + (~(2*B[21:12]) + 1); octn = 0;
                                      state_next = ADMIN_1;
                             end

                    DRAW_CENTER:
                       begin
                           pos_xn = cx; pos_yn = cy;
                           state_next = CIRCLE_DRAW;
                       end

                    CIRCLE_DRAW:
                             begin
                                      if (x1 <= y1)
                                               begin
                                                        if (oct == 8)
                                                                 begin
                                      octn = 0;
                                      if (p[9]) pn = p + (4 * x1) + 6;
                                      else
                                      begin
                                      pn = p + (4 * (x1 + (~y1 + 1))) + 10;
                                      y1n = y1 + (~10'b1 + 1);
                                      end
                                      x1n = x1 + 1;
                                      end
                             else
                                      begin

case (oct)

0: begin pos_xn = cx + x1; pos_yn = cy + y1; end

1: begin pos_xn = cx + (~x1 + 1); pos_yn = cy + y1; end

2: begin pos_xn = cx + x1; pos_yn = cy + (~y1 + 1); end

3: begin pos_xn = cx + (~x1 + 1); pos_yn = cy + (~y1 + 1); end

4: begin pos_xn = cx + y1; pos_yn = cy + x1; end

5: begin pos_xn = cx + (~y1 + 1); pos_yn = cy + x1; end
6: begin pos_xn = cx + y1; pos_yn = cy + (~x1 + 1); end
7: begin pos_xn = cx + (~y1 + 1); pos_yn = cy + (~x1 + 1); end

endcase
```

```
if ((pos_x == cx) && (pos_y == cy)) point_type = 1;

else if ((x1 == r) || (y1 == r)) point_type = 2;

else point_type = 3;

octn = oct + 1;
                                                                    end
                                                 state_next = CIRCLE_DRAW;
                                                 end
                                       else state_next = DATA_WAIT_1;

                                       if (mode == 1)     // Select color
                     data_start = {sel_obj_num, data_type, SELECT_COLOR};
                     else if (color_change)      // Deselect to new color
                              data_start = {sel_obj_num, data_type, color};
                     else if ((mode == 2) || (mode == 4) || (mode == 5))
// Deselect back to original color
                     data_start = {sel_obj_num, data_type, sel_color};
                     else if (mode == 3)// Delete completely
                              data_start = 36'h0;
                              else data_start = {obj_num, data_type, color};
                                       add_out = {pos_x[9:0], pos_y[8:0]};
                              end

                     ADMIN_1:
                        begin
                           state_next = ADMIN_2;
                           if (mode == 3)
                              begin
                                 data_start = 36'h0;
                                 add_out = 19'h50911 + sel_obj_num;
                              end
                           else if (color_change)
                              begin
                               data_start = {color, cx[9:0], cy[8:0], 3'b100};
                                 add_out = 19'h50911 + sel_obj_num;
                              end
                           else if (mode == 0)
                              begin
                               data_start = {color, cx[9:0], cy[8:0], 3'b100};
                                 add_out = 19'h50911 + obj_num;
                              end
                           else
                              begin
                           data_start = {sel_color, cx[9:0], cy[8:0], 3'b100};
                                    add_out = 19'h50911 + sel_obj_num;
                                          end
                              end

                     ADMIN_2:
                        begin

                           state_next = ADMIN_3;
                           if (mode == 3)
                              begin
                                 data_start = 36'h0;
                                 add_out = 19'h55731 + sel_obj_num;
                              end
                           else if (color_change)
                              begin
                                 data_start = {color, r, 9'b0, 3'b100};
                                 add_out = 19'h55731 + sel_obj_num;
                              end
                           else if (mode == 0)
                              begin
                                 data_start = {color, r, 9'b0, 3'b100};
                                 add_out = 19'h55731 + obj_num;
                              end
                           else
                              begin
                                 data_start = {sel_color, r, 9'b0, 3'b100};
                                     add_out = 19'h55731 + sel_obj_num;
                                          end
                              end
```

```
                              ADMIN_3:
                                 begin


                                    state_next = ADMIN_4;
                                    if (mode == 3)
                                       begin
                                          data_start = 36'h0;
                                          add_out = 19'h5A551 + sel_obj_num;
                                       end
                                    else if (color_change)
                                       begin
                                          data_start = {color, cx, cy, 3'b100};
                                          add_out = 19'h5A551 + sel_obj_num;
                                       end
                                    else if (mode == 0)
                                       begin
                                          data_start = {color, cx, cy, 3'b100};
                                          add_out = 19'h5A551 + obj_num;
                                       end
                                    else
                                       begin
                                         data_start = {sel_color, cx, cy, 3'b100};
                                              add_out = 19'h5A551 + sel_obj_num;
                                                     end
                                        end

                              ADMIN_4:
                                 begin
                                    state_next = DRAW_CENTER;
                                    if (mode == 3)
                                       begin
                                          data_start = 36'h0;
                                          add_out = 19'h5F371 + sel_obj_num;
                                       end
                                    else if (color_change)
                                       begin
                                          data_start = {color, rx, ry, 3'b100};
                                          add_out = 19'h5F371 + sel_obj_num;
                                       end
                                    else if (mode == 0)
                                       begin
                                          data_start = {color, rx, ry, 3'b100};
                                          add_out = 19'h5F371 + obj_num;
                                       end
                                    else
                                       begin
                                          data_start = {sel_color, rx, ry, 3'b100};
                                              add_out = 19'h5F371 + sel_obj_num;
                                                     end
                                        end

                              DATA_WAIT_1: state_next = DATA_WAIT_2;
                              DATA_WAIT_2: state_next = IDLE;

                              default: state_next = IDLE;

                              endcase
          end

endmodule


/// draw_line.v – Paul Peeling

// COMMAND: draw_line
// Description: Draws a line between the clicked positions
//                                    using Bresenham's algorithm


module draw_line
          (clk, reset, next, busy,
           write, pos_in, data_out, add_out, color, obj_num, obj_inc,
           sel_click, sel_obj_num, obj_type, A, B, sel_color, color_change,
           move_draw, resize_draw);

input clk, reset, next;
```

```
input [2:0] sel_click;
input [18:0] pos_in;
input [8:0] color, sel_color;
input color_change, move_draw, resize_draw;
input [9:0] obj_num, sel_obj_num;
input [2:0] obj_type;
input [35:0] A, B;          // endpoints of selected line

output busy, write;
output [35:0] data_out;
output [18:0] add_out;

reg [2:0] mode, moden;
reg [3:0] state, state_next;

// Y-coords are normally 9 bits but will zero pad to make
// all this arithmetic simpler!

reg [9:0] x1, x1n, x2, x2n;          // End points
reg [9:0] y1, y1n, y2, y2n;

reg [9:0] pos_x, pos_xn;                       // Current positions
reg [9:0] pos_y, pos_yn;

// States
parameter IDLE = 0;
parameter FIRST_POINT = 1;
parameter IDLE2 = 2;
parameter SECOND_POINT = 3;
parameter SELECT_LOAD = 4;
parameter SWAP = 5;
parameter OCTANT = 6;
parameter ANG_HOLD = 7;
parameter VERT_LINE = 8;
parameter HOR_LINE = 9;
parameter ANG_LINE = 10;
parameter ADMIN_1 = 11;
parameter ADMIN_2 = 12;
parameter ADMIN_3 = 13;
parameter DATA_WAIT_1 = 14;                    // Pipeline data wait states for busy
signal
parameter DATA_WAIT_2 = 15;

parameter SELECT_COLOR = 9'o770;   // Yellow is the selection color

wire busy;
assign busy = ((state != IDLE) && (state != IDLE2));
wire write;
assign write = ((state > ANG_HOLD) && (state < DATA_WAIT_1));

reg [35:0] data_start;
reg [35:0] data_late, data_late_2;
reg [35:0] data_out;                                          // Pipelined data
reg [1:0] point_type;
reg [18:0] add_out;

// Bresenham variables
parameter INCR = 1;
parameter DECR = 0;
parameter PREDX = 1;
parameter PREDY = 0;
reg [9:0] dx, dxn, dy, dyn, e, en, e_inc, e_incn, e_noinc, e_noincn;
reg pred, predn, incdec, incdecn;

output obj_inc;
wire obj_inc;
assign obj_inc = ((state == DATA_WAIT_1) && (mode == 0));

reg [3:0] data_type;
always @ (point_type)
         begin
                  case (point_type)
                           0: data_type = 0;
                           1: data_type = 4;
                           2: data_type = 5;
                           default: data_type = 0;
                  endcase
```

```
          end

always @ (posedge clk)
          begin
                    if (reset) state <= IDLE;
                    else state <= state_next;

                    x1 <= x1n; x2 <= x2n; y1 <= y1n; y2 <= y2n;
          dx <= dxn; dy <= dyn; e <= en; e_inc <= e_incn; e_noinc <= e_noincn;
                    pred <= predn; incdec <= incdecn; pos_x <= pos_xn; pos_y <= pos_yn;

                    data_late <= data_start;
                    data_late_2 <= data_late;
                    data_out <= data_late_2;

                    mode <= moden;
          end

always @ (state or next or pos_x or pos_y or sel_click
                    or x1 or x2 or y1 or y2 or e or dx or dy or e_inc
                     or e_noinc or pred or incdec or obj_type or move_draw
                      or resize_draw or pos_in or mode or B or A or sel_obj_num
                       or data_type or color_change or color or sel_color or obj_num)
          begin
                    x1n = x1; x2n = x2; y1n = y1; y2n = y2; en = e;
                    pos_xn = pos_x; pos_yn = pos_y;
                    dxn = dx; dyn = dy; e_incn = e_inc; e_noincn = e_noinc;
                    predn = pred; incdecn = incdec;

                    point_type = 0;             // No point
                    add_out = 0;
                    data_start = 0;

                    case (state)

                              IDLE:
                                        begin
                                                  if (next)
                                                            begin
                                                            state_next = FIRST_POINT;
                                                                      moden = 0;
                                                            end
                                        else if ((sel_click == 1) && (obj_type == 3))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                                      moden = 1;
                                                            end
                                        else if ((sel_click == 2) && (obj_type == 3))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 2;      // Deselect
                                                            end
                                        else if ((sel_click == 3) && (obj_type == 3))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 3;      // Delete
                                                            end
                                                  else if (move_draw && (obj_type == 3))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 4;      // Move
                                                            end
                                        else if (resize_draw && (obj_type == 3))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 5;      // Resize
                                                            end
                                                  else state_next = IDLE;
                                        end

                              FIRST_POINT:
                                        begin
                                                  state_next = IDLE2;
                                        x1n = pos_in[18:9]; y1n = {1'b0, pos_in[8:0]};
                                        end

                              IDLE2:
```

```
                                begin
                                        if (next) state_next = SECOND_POINT;
                                        else state_next = IDLE2;
                                end

          `SECOND_POINT:     // No swapping of points here - want to preserve
// the Breshenham algorithm as much as possible
                                begin
                                        state_next = SWAP;
                                x2n = pos_in[18:9]; y2n = {1'b0, pos_in[8:0]};
                                end

                        SELECT_LOAD:
                           begin
                              if (mode == 4)
                                 begin
      x1n = pos_in[18:9];    x2n = B[21:12] + pos_in[18:9] + (~A[21:12] + 1);
      y1n = pos_in[8:0];     y2n = B[11:3] + pos_in[8:0] + (~A[11:3] + 1);
                                    end
                              else if (mode == 5)
                                 begin
                                    x1n = A[21:12];  x2n = pos_in[18:9];
                                        y1n = A[11:3];        y2n = pos_in[8:0];
                                    end
                                else
                                    begin
                                    x1n = A[21:12];  x2n = B[21:12];
                                        y1n = A[11:3];        y2n = B[11:3];
                                    end
                                state_next = SWAP;
                                end

                        SWAP:
                                begin
                                        if (x1 > x2)
                                                begin
                                                        x2n = x1; y2n = y1;
                                                        x1n = x2; y1n = y2;
                                                dxn = x1 + (~x2 + 1);
                                                dyn = y1 + (~y2 + 1);
                                                end
                                        else
                                                begin
                                                        x1n = x1; y1n = y1;
                                                        x2n = x2; y2n = y2;
                                                dxn = x2 + (~x1 + 1);
                                                dyn = y2 + (~y1 + 1);
                                                end
                                        state_next = OCTANT;
                                end

                        OCTANT:
                                begin
                                        if (dx == 0)       // Vertical line
                                                begin
                                                        if (y1 > y2)
                                                                begin
                                                                y2n = y1;
                                                                y1n = y2;
                                                                end
                                                        state_next = VERT_LINE;
                                                        end
                                        else if (dy == 0)  // Horizontal line
                                                        state_next = HOR_LINE;
                                else if ((dx >= dy) && (!dy[9])) // 0 < m <= 1
                                                        begin
                                                        e_noincn = (2 * dy);
                                                        en = (2 * dy) + (~dx + 1);
                                                e_incn = 2 * (dy + (~dx + 1));
                                                predn = PREDX; incdecn = INCR;
                                                        state_next = ANG_HOLD;
                                                        end
                                else if ((dy > dx) && (!dy[9])) // 1 < m < inf.
                                                        begin
                                                        e_noincn = (2 * dx);
                                                        en = (2 * dx) + (~dy + 1);
                                                e_incn = 2 * (dx + (~dy + 1));
```

```
                                predn = PREDY; incdecn = INCR;
                                state_next = ANG_HOLD;
                                        end
                        else if ((dx >= (~dy + 1)) && (dy[9])) // 0 > m >= -1
                                        begin
                                                dyn = (~dy + 1);
                                        e_noincn = (~10'h2 + 1) * dy;
                                en = ((~10'h2 + 1) * dy) + (~dx + 1);
                                e_incn = 2 * ((~dy + 1) + (~dx + 1));
                                predn = PREDX; incdecn = DECR;
                                        state_next = ANG_HOLD;
                                        end
                        else if (((~dy + 1) > dx) && (dy[9])) // -1 > m > inf.
                                        begin
                                                dxn = (~dx + 1);
                                                e_noincn = 2 * dx;
                                en = ((~10'h2 + 1) * dx) + (~dy + 1);
                        e_incn = (~10'h2 + 1)  * ((~dx + 1) + (~dy + 1));
                                predn = PREDY; incdecn = DECR;
                                state_next = ANG_HOLD;
                                        end
                                end

                        VERT_LINE:
                                begin
                        if ((pos_y == y1) || (pos_x == y2)) point_type = 1;
                                else point_type = 2;

                                if (pos_y == y2)  state_next = ADMIN_1;
                                else
                                        begin
                                                pos_xn = pos_x;
                                                pos_yn = pos_y + 1;
                                        state_next = VERT_LINE;
                                        end

                                        if (mode == 1)     // Select color
                                data_start = {sel_obj_num, data_type, 9'o707};
                        else if (color_change)       // Deselect to new color
                                data_start = {sel_obj_num, data_type, color};
                        else if ((mode == 2) || (mode == 4) || (mode == 5))
// Deselect back to original color
                        data_start = {sel_obj_num, data_type, sel_color};
                                else if (mode == 3)// Delete completely
                                        data_start = 36'h0;
                                else data_start = {obj_num, data_type, color};
                                        add_out = {pos_x[9:0], pos_y[8:0]};
                                end

                        HOR_LINE:
                                begin
                        if ((pos_x == x1) || (pos_x == x2)) point_type = 1;
                                        else point_type = 2;

                                        if (pos_x == x2) state_next = ADMIN_1;
                                        else
                                                begin
                                                        pos_xn = pos_x + 1;
                                                        pos_yn = pos_y;
                                                state_next = HOR_LINE;
                                                end

                                        if (mode == 1)     // Select color
                                data_start = {sel_obj_num, data_type, 9'o707};
                        else if (color_change)       // Deselect to new color
                                data_start = {sel_obj_num, data_type, color};
                        else if ((mode == 2) || (mode == 4) || (mode == 5))
// Deselect back to original color
                        data_start = {sel_obj_num, data_type, sel_color};
              else if (mode == 3)    // Delete completely
                                        data_start = 36'h0;
                                else data_start = {obj_num, data_type, color};
                                        add_out = {pos_x[9:0], pos_y[8:0]};
                                end

                        ANG_HOLD:
                                begin
```

```
                                        pos_xn = x1;
                                        pos_yn = y1;
                                        state_next = ANG_LINE;
                                end

                        ANG_LINE:
                                begin
if ((pos_x == x1)&&(pos_y == y1) || (pos_x == x2)&&(pos_y == y2)) point_type = 1;
                                        else point_type = 2;

                if ((pos_xn == x2) && (pos_yn == y2)) state_next = ADMIN_1;
                                        else
                                                begin
                                                if (e[9]) en = e + e_noinc;
                                                        else
                                                                begin

        if ((pred == PREDX) && (incdec == INCR)) pos_yn = pos_y + 1;

        else if ((pred == PREDX) && (incdec == DECR)) pos_yn = pos_y + (~10'h1 + 1);

        else if (incdec == INCR) pos_xn = pos_x + 1;

        else pos_xn = pos_x + (~10'h1 + 1);

        en = e + e_inc;
                                                                end
                                        if (pred == PREDX) pos_xn = pos_x + 1;
                                        else pos_yn = pos_y + 1;
                                                state_next = ANG_LINE;
                                                end

                                        if (mode == 1)     // Select color
                                data_start = {sel_obj_num, data_type, 9'o707};
                        else if (color_change)       // Deselect to new color
                                data_start = {sel_obj_num, data_type, color};
                        else if ((mode == 2) || (mode == 4) || (mode == 5))
        // Deselect back to original color
                        data_start = {sel_obj_num, data_type, sel_color};
                                else if (mode == 3)// Delete completely
                                                data_start = 36'h0;
                                else data_start = {obj_num, data_type, color};
                                        add_out = {pos_x[9:0], pos_y[8:0]};
                                end

                        ADMIN_1:
                           begin
                              state_next = ADMIN_2;
                              if (mode == 3)
                                 begin
                                    data_start = 36'h0;
                                    add_out = 19'h50911 + sel_obj_num;
                                 end
                              else if (color_change)
                                 begin
                                    data_start = {color, x1, y1, 3'b011};
                                    add_out = 19'h50911 + sel_obj_num;
                                 end
                              else if (mode == 0)
                                 begin
                                    data_start = {color, x1, y1, 3'b011};
                                    add_out = 19'h50911 + obj_num;
                                 end
                              else
                                 begin
                                    data_start = {sel_color, x1, y1, 3'b011};
                                           add_out = 19'h50911 + sel_obj_num;
                                              end
                                 end

                        ADMIN_2:
                           begin
                              state_next = ADMIN_3;
                              if (mode == 3)
                                 begin
                                    data_start = 36'h0;
                                    add_out = 19'h55731 + sel_obj_num;
```

```
                                    end
                                else if (color_change)
                                    begin
                                        data_start = {color, x1, y1, 3'b011};
                                        add_out = 19'h55731 + sel_obj_num;
                                    end
                                else if (mode == 0)
                                    begin
                                        data_start = {color, x2, y2, 3'b011};
                                        add_out = 19'h55731 + obj_num;
                                    end
                                else
                                    begin
                                        data_start = {sel_color, x2, y2, 3'b011};
                                            add_out = 19'h55731 + sel_obj_num;
                                                end
                                     end

                            ADMIN_3:
                                begin
                                    state_next = DATA_WAIT_1;
                                    data_start = 36'h0;
                                    add_out = 19'h5A551 + sel_obj_num;
                                end

                            DATA_WAIT_1: state_next = DATA_WAIT_2;
                            DATA_WAIT_2: state_next = IDLE;

                            default: state_next = IDLE;

                    endcase

            end

endmodule


/// draw_point.v – Paul Peeling

// COMMAND: draw_point
// Description: Draws a point at the clicked position

module draw_point
          (clk, reset, next, busy,
           write, pos_in, data_out, add_out, color, obj_num, obj_inc,
           sel_click, sel_obj_num, obj_type, A, sel_color, color_change,
           move_draw);

input clk, reset, next;
input [2:0] sel_click;
input [18:0] pos_in;
input [8:0] color, sel_color;
input color_change, move_draw;
input [9:0] obj_num, sel_obj_num;
input [2:0] obj_type;
input [35:0] A;

output busy, write;
output [35:0] data_out;      // Point identifier
output [18:0] add_out;

reg [18:0] add_out;
reg [35:0] data_start;                                  // Pipelined data
reg [35:0] data_late, data_late_2;
reg [35:0] data_out;

// No resize possible
reg [2:0] mode, moden;                // 0: draw, 1: select 2: deselect 3: delete 4:
move
reg [2:0] state, state_next;

// States

parameter IDLE = 0;
parameter SELECT_DRAW = 1;
parameter DRAW = 2;
parameter ADMIN_1 = 3;
```

```
parameter ADMIN_2 = 4;
parameter DATA_WAIT_1 = 5;
parameter DATA_WAIT_2 = 6;

parameter SELECT_COLOR = 9'o770;   // Yellow is the selection color

wire busy, write;
assign busy = (state != IDLE);
assign write = ((state == DRAW) || (state == ADMIN_1));

output obj_inc;
wire obj_inc;
assign obj_inc = ((state == DATA_WAIT_1) && (mode == 0));

always @ (posedge clk)
         begin
                  if (reset) state <= IDLE;
                  else state <= state_next;

                  data_late <= data_start;
                  data_late_2 <= data_late;
                  data_out <= data_late_2;

                  mode <= moden;
         end

always @ (state or next or sel_click or move_draw
                  or obj_type or mode or obj_num or color_change
                   or color or sel_color or pos_in or A or sel_obj_num)
         begin

                  data_start = 0;
                  add_out = 0;

                  case (state)

                           IDLE:
                                    begin
                                             if (next)
                                                      begin
                                                               state_next = DRAW;
                                                               moden = 0;
                                                      end
                                    else if ((sel_click == 1) && (obj_type == 1))
                                                      begin
                                                      state_next = SELECT_DRAW;
                                                               moden = 1;
                                                      end
                                    else if ((sel_click == 2) && (obj_type == 1))
                                                      begin
                                                      state_next = SELECT_DRAW;
                                                      moden = 2;      // Deselect
                                                      end
                                    else if ((sel_click == 3) && (obj_type == 1))
                                                      begin
                                                      state_next = SELECT_DRAW;
                                                      moden = 3;      // Delete
                                                      end
                                             else if (move_draw && (obj_type == 1))
                                                      begin
                                                      state_next = SELECT_DRAW;
                                                      moden = 4;      // Move
                                                      end
                                             else state_next = IDLE;
                                    end

                  SELECT_DRAW:
                     begin
                         state_next = ADMIN_1;
                                  if (mode == 1) data_start = {obj_num, 1'b1, 9'o707};
                             else if (color_change) data_start = {obj_num, 1'b1, color};
                                                       else if ((mode == 2) || (mode == 4))
data_start = {obj_num, 1'b1, sel_color};
                                                      else if (mode == 3) data_start = 36'h0;

                                                      if (mode == 4) add_out = pos_in;
                                                      else add_out = A[21:3];
```

```
                                        end

                                DRAW:
                                        begin
                                                state_next = ADMIN_1;
                                        data_start = {obj_num, 1'b1, color};
        // Type of a point is 1
                                                add_out = pos_in;
                                        end

                                ADMIN_1:
                                   begin
                                      state_next = ADMIN_2;
                                      if (mode == 3)
                                         begin
                                            data_start = 36'h0;
                                            add_out = 19'h50911 + sel_obj_num;
                                         end
                                      else if (color_change)
                                         begin
                                            data_start = {color, A[21:3], 3'b001};
                                            add_out = 19'h50911 + sel_obj_num;
                                         end
                                      else if (mode == 0)
                                         begin
                                            data_start = {color, pos_in, 3'b001};
                                            add_out = 19'h50911 + obj_num;
                                         end
                                      else if (mode == 4)
                                         begin
                                            data_start = {sel_color, pos_in, 3'b001};
                                            add_out = 19'h50911 + sel_obj_num;
                                         end
                                      else
                                         begin
                                            data_start = {sel_color, A[21:3], 3'b001};
                                                    add_out = 19'h50911 + sel_obj_num;
                                             end
                                         end

                                ADMIN_2:
                                   begin
                                      state_next = DATA_WAIT_1;
                                      data_start = 36'h0;
                                      add_out = 19'h5A551 + sel_obj_num;
                                   end

                                DATA_WAIT_1: state_next = DATA_WAIT_2;
                                DATA_WAIT_2: state_next = IDLE;
                                default: state_next = IDLE;

                        endcase
        end

endmodule

// draw_rect.v – Paul Peeling

// COMMAND: draw_rect
// Description: Draws a rectangle between the clicked positions

module draw_rect
         (clk, reset, next, busy,
          write, pos_in, data_out, add_out, color, obj_num, obj_inc,
          sel_click, sel_obj_num, obj_type, A, B, sel_color, color_change,
          move_draw, resize_draw);

input clk, reset, next;
input [2:0] sel_click;
input [18:0] pos_in;
input [8:0] color, sel_color;
input color_change, move_draw, resize_draw;
input [9:0] obj_num, sel_obj_num;
input [2:0] obj_type;
input [35:0] A, B;          // A and B are the (now!) two corners of a rectangle
object
```

```
output busy;
output write;                           // WE
output [35:0] data_out;      // Point identifier
output [18:0] add_out;

reg [2:0] mode, moden;       // 0: draw, 1: select 2: deselect 3: delete 4: move
         5: resize
reg [3:0] state, state_next;

reg [9:0] x1, x2, x1n, x2n;                     // Upper-left and lower-right corners
reg [8:0] y1, y2, y1n, y2n;

reg [9:0] pos_x, pos_xn;                        // current positions
reg [8:0] pos_y, pos_yn;

// States
parameter IDLE = 0;
parameter FIRST_POINT = 1;
parameter IDLE2 = 2;
parameter SECOND_POINT = 3;
parameter SELECT_LOAD = 4;                      // Points are already in correct
ordering
parameter VERT_WAIT = 5;
parameter VERTICAL = 6;
parameter HORIZONTAL = 7;
parameter ADMIN_1 = 8;
parameter ADMIN_2 = 9;
parameter ADMIN_3 = 10;
parameter ADMIN_4 = 11;
parameter DATA_WAIT_1 = 12;           // Pipeline data wait states for busy signal
parameter DATA_WAIT_2 = 13;

parameter SELECT_COLOR = 9'o770;   // Yellow is the selection color

wire busy;
assign busy = ((state != IDLE) && (state != IDLE2));
wire write;
assign write = ((state > VERT_WAIT) && (state < DATA_WAIT_1));

reg [35:0] data_start;
reg [35:0] data_late, data_late_2;
reg [35:0] data_out;                                                  // Pipelined data
reg [1:0] point_type;

reg [18:0] add_out;

output obj_inc;
wire obj_inc;
assign obj_inc = ((state == DATA_WAIT_1) && (mode == 0));

reg [3:0] data_type;
always @ (point_type)
         begin
                  case (point_type)
                           0: data_type = 0;
                           1: data_type = 2;
                           2: data_type = 3;
                           default: data_type = 0;
                  endcase
         end

always @ (posedge clk)
         begin

                  if (reset)
                           begin
                                    state <= IDLE;
                           end
                  else state <= state_next;

                  pos_x <= pos_xn; pos_y <= pos_yn;
                  x1 <= x1n; x2 <= x2n; y1 <= y1n; y2 <= y2n;
                  mode <= moden;

                  data_late <= data_start;
                  data_late_2 <= data_late;
                  data_out <= data_late_2;
```

```
          end

always @ (state or next or pos_x or pos_y or sel_click
                    or x1 or x2 or y1 or y2 or mode or obj_type
                     or move_draw or resize_draw or pos_in or A or B
                      or sel_obj_num or data_type or color_change or color or sel_color
or obj_num)
          begin

                    x1n = x1; x2n = x2; y1n = y1; y2n = y2;
                    pos_xn = pos_x; pos_yn = pos_y;

                    data_start = 0;
                    add_out = 0;
                    moden = mode;

                    case (state)

                              IDLE:
                                        begin
                                                  if (next)
                                                            begin
                                                            state_next = FIRST_POINT;
                                                                      moden = 0;
                                                            end
                                        else if ((sel_click == 1) && (obj_type == 2))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                                      moden = 1;
                                                            end
                                        else if ((sel_click == 2) && (obj_type == 2))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 2;      // Deselect
                                                            end
                                        else if ((sel_click == 3) && (obj_type == 2))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 3;      // Delete
                                                            end
                                                  else if (move_draw && (obj_type == 2))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 4;      // Move
                                                            end
                                        else if (resize_draw && (obj_type == 2))
                                                            begin
                                                            state_next = SELECT_LOAD;
                                                            moden = 5;      // Resize
                                                            end
                                                  else state_next = IDLE;
                                        end

                              FIRST_POINT:
                                        begin
                                                  x1n = pos_in[18:9];
                                                  y1n = pos_in[8:0];
                                                  state_next = IDLE2;
                                        end

                              IDLE2:
                                        begin
                                                  if (next) state_next = SECOND_POINT;
                                                  else state_next = IDLE2;
                                        end

SECOND_POINT:       // Swap to give upper-left and lower-right corners if necessary
                                        begin
                                        if ((pos_in[18:9] < x1) && (pos_in[8:0] < y1))
                                                            begin
                                                                      x2n = x1; y2n = y1;
                                                  x1n = pos_in[18:9];y1n = pos_in[8:0];
                                                            end
                                        else if ((pos_in[18:9] < x1) && (pos_in[8:0] > y1))
                                                            begin
                                                                      x2n = x1; y1n = y1;
```

```
                    y2n = pos_in[8:0]; x1n = pos_in[18:9];
                              end
else if ((pos_in[18:9] > x1) && (pos_in[8:0] < y1))
                              begin
                                        y2n = y1; x1n = x1;
                    x2n = pos_in[18:9]; y1n = pos_in[8:0];
                              end
                    else
                              begin
                                        x1n = x1; y1n = y1;
                    x2n = pos_in[18:9]; y2n = pos_in[8:0];
                              end

                    state_next = VERT_WAIT;
                    pos_xn = x1;
                    pos_yn = y1;
          end

SELECT_LOAD:                //all points are pre-ordered
          begin
             if (mode == 4)
                begin
      x1n = pos_in[18:9]; pos_xn = pos_in[18:9];
      x2n = B[21:12] + pos_in[18:9] + (~A[21:12] + 1);
      y1n = pos_in[8:0]; pos_yn = pos_in[8:0];
      y2n = B[11:3] + pos_in[8:0] + (~A[11:3] + 1);
                end
             else if (mode == 5)
                begin
      x1n = A[21:12]; pos_xn = A[21:12];
                   x2n = pos_in[18:9];
                   y1n = A[11:3]; pos_yn = A[11:3];
                   y2n = pos_in[8:0];
                end
             else
                begin
      x1n = A[21:12]; pos_xn = A[21:12];
                   x2n = B[21:12];
                   y1n = A[11:3]; pos_yn = A[11:3];
                   y2n = B[11:3];
                end
             state_next = VERTICAL;
          end

VERT_WAIT:
   begin
      state_next = VERTICAL;
      point_type = 1;
   end

VERTICAL:
          begin
                    if ((pos_x == x1)&&(pos_y == y1) ||
                    (pos_x == x2)&&(pos_y == y2) ||
                    (pos_x == x1)&&(pos_y == y2) ||
          (pos_x == x2)&&(pos_y == y1)) point_type = 1;
                    else point_type = 2;

                    if ((pos_x == x2) && (pos_y == y2))
                              begin
                              state_next = HORIZONTAL;
                                        pos_xn = x1;
                                        pos_yn = y1;
                              end
                    else if (pos_y == y2)
                              begin
                                        pos_yn = y1;
                                        pos_xn = x2;
                              state_next = VERTICAL;
                              end
                    else
                              begin
                                        pos_yn = pos_y + 1;
                                        pos_xn = pos_x;
                              state_next = VERTICAL;
                              end
```

```
                                        if (mode == 1)     // Select color
                                    data_start = {sel_obj_num, data_type, 9'o707};
                            else if (color_change)       // Deselect to new color
                                    data_start = {sel_obj_num, data_type, color};
                            else if ((mode == 2) || (mode == 4) || (mode == 5))
        // Deselect back to original color
                            data_start = {sel_obj_num, data_type, sel_color};
                                    else if (mode == 3)// Delete completely
                                                data_start = 36'h0;
                                    else data_start = {obj_num, data_type, color};
                                            add_out = {pos_x, pos_y};
                                    end

                            HORIZONTAL:
                                    begin
                                            if ((pos_x == x1)&&(pos_y == y1) ||
                                            (pos_x == x2)&&(pos_y == y2) ||
                                            (pos_x == x1)&&(pos_y == y2) ||
                                    (pos_x == x2)&&(pos_y == y1)) point_type = 1;
                                            else point_type = 2;

                                            if ((pos_x == x2) && (pos_y == y2))
                                                    begin
                                    state_next = ADMIN_1;
                                                            pos_xn = 0;
                                                            pos_yn = 0;
                                                    end
                                            else if (pos_x == x2)
                                                    begin
                                                            pos_xn = x1;
                                                            pos_yn = y2;
                                                    state_next = HORIZONTAL;
                                                    end
                                            else
                                                    begin
                                                            pos_xn = pos_x + 1;
                                                            pos_yn = pos_y;
                                                    state_next = HORIZONTAL;
                                                    end

                                            if (mode == 1)     // Select color
                                    data_start = {obj_num, data_type, 9'o707};
                            else if (color_change)       // Deselect to new color
                                    data_start = {sel_obj_num, data_type, color};
                            else if ((mode == 2) || (mode == 4) || (mode == 5))
        // Deselect back to original color
                            data_start = {sel_obj_num, data_type, sel_color};
                            else if (mode == 3)// Delete completely
                                                data_start = 36'h0;
                                    else data_start = {obj_num, data_type, color};
                                            add_out = {pos_x, pos_y};
                                    end

                            ADMIN_1:
                               begin
                                  state_next = ADMIN_2;
                                  if (mode == 3)
                                     begin
                                        data_start = 36'h0;
                                        add_out = 19'h50911 + sel_obj_num;
                                     end
                                  else if (color_change)
                                     begin
                                        data_start = {color, x1, y1, 3'b010};
                                        add_out = 19'h50911 + sel_obj_num;
                                     end
                                  else if (mode == 0)
                                     begin
                                        data_start = {color, x1, y1, 3'b010};
                                        add_out = 19'h50911 + obj_num;
                                     end
                                  else
                                     begin
                                        data_start = {sel_color, x1, y1, 3'b010};
                                            add_out = 19'h50911 + sel_obj_num;
                                                end
                                    end
```

```
ADMIN_2:
   begin
      state_next = ADMIN_3;
      if (mode == 3)
         begin
            data_start = 36'h0;
            add_out = 19'h55731 + sel_obj_num;
         end
      else if (color_change)
         begin
            data_start = {color, x2, y2, 3'b010};
            add_out = 19'h55731 + sel_obj_num;
         end
      else if (mode == 0)
         begin
            data_start = {color, x2, y2, 3'b010};
            add_out = 19'h55731 + obj_num;
         end
      else
         begin
            data_start = {sel_color, x2, y2, 3'b010};
                add_out = 19'h55731 + sel_obj_num;
                       end
          end

ADMIN_3:
   begin
      state_next = ADMIN_4;
      if (mode == 3)
         begin
            data_start = 36'h0;
            add_out = 19'h5A551 + sel_obj_num;
         end
      else if (color_change)
         begin
            data_start = {color, x1, y2, 3'b010};
            add_out = 19'h5A551 + sel_obj_num;
         end
      else if (mode == 0)
         begin
            data_start = {color, x1, y2, 3'b010};
            add_out = 19'h5A551 + obj_num;
         end
      else
         begin
            data_start = {sel_color, x1, y2, 3'b010};
                add_out = 19'h5A551 + sel_obj_num;
                       end
          end

ADMIN_4:
   begin
      state_next = DATA_WAIT_1;
      if (mode == 3)
         begin
            data_start = 36'h0;
            add_out = 19'h5F371 + sel_obj_num;
         end
      else if (color_change)
         begin
            data_start = {color, x2, y1, 3'b010};
            add_out = 19'h5F371 + sel_obj_num;
         end
      else if (mode == 0)
         begin
            data_start = {color, x2, y1, 3'b010};
            add_out = 19'h5F371 + obj_num;
         end
      else
         begin
            data_start = {sel_color, x2, y1, 3'b010};
                add_out = 19'h5F371 + sel_obj_num;
                       end
          end

DATA_WAIT_1: state_next = DATA_WAIT_2;
```

```
                            DATA_WAIT_2: state_next = IDLE;

                            default: state_next = IDLE;

                    endcase
            end

endmodule



/// move_obj.v – Paul Peeling

module move_obj
   (clk, reset, start, busy, move_del, move_draw, move_sel,
    sel_busy, sel_draw);

input clk, reset, start, sel_busy, sel_draw;

output busy, move_del, move_draw, move_sel;
reg move_del, move_draw, move_sel;

reg [3:0] state, next;

// states

parameter IDLE = 0;
parameter DELETE_READY = 1;
parameter DELETE_TRIGGER = 2;
parameter DELETE_DONE = 3;
parameter DRAW_READY = 4;
parameter DRAW_TRIGGER = 5;
parameter DRAW_DONE = 6;
parameter SEL_READY = 7;
parameter SEL_TRIGGER = 8;
parameter SEL_DONE = 9;

wire busy;
assign busy = (state != IDLE);

always @ (posedge clk)
   begin
       if (reset) state <= IDLE;
       else state <= next;
   end

always @ (state or start or sel_busy or sel_draw)
   begin

       move_del = 0;
       move_draw = 0;
       move_sel = 0;

        case (state)

            IDLE:
               begin
                   if (start) next = DELETE_READY;
                   else next = IDLE;
               end

                            DELETE_READY:
                                     begin
                                              move_del = 1;
                                              if (sel_busy) next = DELETE_READY;
                                              else next = DELETE_TRIGGER;
                                     end

                            DELETE_TRIGGER:
                                     begin
                                              if (sel_busy) next = DELETE_DONE;
                                              else next = DELETE_TRIGGER;
                                     end

                            DELETE_DONE:
                                     begin
                                              if (sel_busy) next = DELETE_DONE;
```

```
                                        else next = DRAW_READY;
                                end

                        DRAW_READY:
                                begin
                                        move_draw = 1;
                                        if (sel_draw) next = DRAW_READY;
                                        else next = DRAW_TRIGGER;
                                end

                        DRAW_TRIGGER:
                                begin
                                        if (sel_draw) next = DRAW_DONE;
                                        else next = DRAW_TRIGGER;
                                end

                        DRAW_DONE:
                                begin
                                        if (sel_draw) next = DRAW_DONE;
                                        else next = SEL_READY;
                                end

                        SEL_READY:
                                begin
                                        move_sel = 1;
                                        if (sel_busy) next = SEL_READY;
                                        else next = SEL_TRIGGER;
                                end

                        SEL_TRIGGER:
                                begin
                                        if (sel_busy) next = SEL_DONE;
                                        else next = SEL_TRIGGER;
                                end

                        SEL_DONE:
                                begin
                                        if (sel_busy) next = SEL_DONE;
                                        else next = IDLE;
                                end

                        default: next = IDLE;

                        endcase
        end

endmodule

/// mult2comp.v – Paul Peeling
/// 8x8 bit twos complement multiplier

module mult2comp (X, Y, Z);
        input [7:0] X, Y;
        output [15:0] Z;

        wire [15:0] Z, Zres;
        wire [7:0] Xus, Yus;

        assign Xus = (~X + 1);      // unsigned X and Y
        assign Yus = (~Y + 1);

        assign Zres = (X[7] ? Xus : X) * (Y[7] ? Yus : Y);      // Selects positive
X and Y, multiplies


        assign Z = (X[7] ^ Y[7]) ? (~Zres + 1) : Zres;  // Either X or Y but not both
are negative


endmodule

/// radius_calc.v – Paul Peeling

// Calculates radius of circle

// Implemented as 8-bit square root until I think through this algorithm properly
// Therefore max. radius of circle is limited
```

```
// Megitt algorithm (1962)

module radius_calc
         (clk, reset, cx, cy, rx, ry, radius, start, busy);

input clk, reset, start;
input [7:0] cx, cy, rx, ry;
output [7:0] radius;
output busy;

wire [7:0] xdist, ydist;
assign xdist = rx - cx;
assign ydist = ry - cy;

wire [15:0] xsq, ysq;
mult2comp TMULTX (.X(xdist), .Y(xdist), .Z(xsq));
mult2comp TMULTY (.X(ydist), .Y(ydist), .Z(ysq));

wire [15:0] radsq;
assign radsq = xsq + ysq;

reg [7:0] radius;
reg [23:0] rem;
wire [10:0] diff;

assign diff = rem[23:14] - {radius, 2'b01};

reg [3:0] count;
reg busy;

always @ (posedge clk)
         begin

                  if (reset)
                           begin
                                    busy <= 0;
                                    count <= 0;
                                    radius <= 0;
                                    rem <= radsq;
                           end
                  else if (start)
                           begin
                                    busy <= 1;
                                    count <= 0;
                                    radius <= 0;
                                    rem <= radsq;
                           end
                  else if (busy)
                           begin
                                    if (count < 8)
                                             begin
                                                      busy <= 1;
                                                      count <= count + 1;
                                             if (diff[10] == 1) // diff negative
                                                               begin
                                                      rem <= {rem[21:0], 2'b00};
                                             radius <= {radius[6:0], 1'b0};
                                                               end
                                                      else
                                                               begin
                                             rem <= {diff[7:0], rem[13:0], 2'b00};
                                             radius <= {radius[6:0], 1'b1};
                                                               end
                                             end
                                    else busy <= 0;
                           end
         end

endmodule

/// resize_obj.v – Paul Peeling

module resize_obj
   (clk, reset, start, busy, resize_del, resize_draw, resize_sel,
    sel_busy, sel_draw);

input clk, reset, start, sel_busy, sel_draw;
```

```
output busy, resize_del, resize_draw, resize_sel;
reg resize_del, resize_draw, resize_sel;

reg [3:0] state, next;

// states

parameter IDLE = 0;
parameter DELETE_READY = 1;
parameter DELETE_TRIGGER = 2;
parameter DELETE_DONE = 3;
parameter DRAW_READY = 4;
parameter DRAW_TRIGGER = 5;
parameter DRAW_DONE = 6;
parameter SEL_READY = 7;
parameter SEL_TRIGGER = 8;
parameter SEL_DONE = 9;

wire busy;
assign busy = (state != IDLE);

always @ (posedge clk)
   begin
       if (reset) state <= IDLE;
       else state <= next;
   end

always @ (state or start or sel_busy or sel_draw)
   begin

       resize_del = 0;
            resize_draw = 0;
       resize_sel = 0;

        case (state)

            IDLE:
               begin
                   if (start) next = DELETE_READY;
                   else next = IDLE;
               end

                           DELETE_READY:
                                    begin
                                             resize_del = 1;
                                             if (sel_busy) next = DELETE_READY;
                                             else next = DELETE_TRIGGER;
                                    end

                           DELETE_TRIGGER:
                                    begin
                                             if (sel_busy) next = DELETE_DONE;
                                             else next = DELETE_TRIGGER;
                                    end

                           DELETE_DONE:
                                    begin
                                             if (sel_busy) next = DELETE_DONE;
                                             else next = DRAW_READY;
                                    end

                           DRAW_READY:
                                    begin
                                             resize_draw = 1;
                                             if (sel_draw) next = DRAW_READY;
                                             else next = DRAW_TRIGGER;
                                    end

                           DRAW_TRIGGER:
                                    begin
                                             if (sel_draw) next = DRAW_DONE;
                                             else next = DRAW_TRIGGER;
                                    end

                           DRAW_DONE:
                                    begin
```

```
                                        if (sel_draw) next = DRAW_DONE;
                                        else next = SEL_READY;
                                end

                        SEL_READY:
                                begin
                                        resize_sel = 1;
                                        if (sel_busy) next = SEL_READY;
                                        else next = SEL_TRIGGER;
                                end

                        SEL_TRIGGER:
                                begin
                                        if (sel_busy) next = SEL_DONE;
                                        else next = SEL_TRIGGER;
                                end

                        SEL_DONE:
                                begin
                                        if (sel_busy) next = SEL_DONE;
                                        else next = IDLE;
                                end

                        default: next = IDLE;

                        endcase
        end

endmodule


/// select.v – Paul Peeling

// COMMAND: Select
// Selects an object from the memory and redraws in a new color
// also offering details of object to other commands

// Need to implement deselection of original object too!

module select
         (clk, reset, next, busy, pos_in, info_in, add_out,
          A, B, C, D, nullify, sel_next, sel_busy,
           obj_type, obj_num, color, move_del, move_sel,
           color_change, new_color);

input clk, reset, next, move_del, move_sel, color_change;
input [18:0] pos_in;
input [35:0] info_in;                // Object data from SRAM
input nullify;                                  // indicates when the object should be
deleted instead
input sel_busy;                                 // OR-ed busy signal from the drawing
modules

output busy;
output [18:0] add_out;
output [35:0] A, B, C, D;                       // A,B,C,D are the selected object
nodes
output [8:0] color;
input [8:0] new_color;

output [2:0] obj_type;
output [9:0] obj_num;
output [2:0] sel_next;                          // Next signal to drawing modules
                        // 1 select, 2 deselect, 3 delete

reg [8:0] color, colorn;
reg [9:0] obj_num, obj_numn;
reg [2:0] obj_type, obj_typen;

reg [35:0] A, An, B, Bn, C, Cn, D, Dn;
reg [18:0] add_out;

// states
// This is clumsy due to read pipelining

reg [4:0] state, state_next;
```

```
parameter IDLE = 0;
parameter DESEL_READY = 1;
parameter DESEL_TRIGGER = 2;
parameter DESEL_DONE = 3;
parameter GET_INFO = 4;
parameter INFO_WAIT_1 = 5;
parameter INFO_WAIT_2 = 6;
parameter INFO_WAIT_3 = 7;
parameter INFO_EXTRACT = 8;
parameter REQ_A = 9;
parameter REQ_B = 10;
parameter REQ_C = 11;
parameter REQ_D = 12;
parameter GET_A = 13;
parameter GET_B = 14;
parameter GET_C = 15;
parameter GET_D = 16;
parameter SEL_READY = 17;
parameter SEL_TRIGGER = 18;
parameter SEL_DONE = 19;
parameter RELOAD_MOVE = 20;

wire busy;
assign busy = (state != IDLE);
reg [2:0] sel_next;

always @ (posedge clk)
          begin
                    if (reset)
                              begin
                                        state <= IDLE;
                                        A <= 0; B <= 0; C <= 0; D <= 0;
                                        color <= 0; obj_num <= 0; obj_type <= 0;
                              end
                    else
                              begin
                                        state <= state_next;
                                        A <= An; B <= Bn; C <= Cn; D <= Dn; color <=
colorn;
                                        obj_num <= obj_numn; obj_type <= obj_typen;
                              end
          end

always @ (state or next or sel_busy or move_del or move_sel or color_change)
          begin
                    An = A; Bn = B; Cn = C; Dn = D;
                    obj_numn = obj_num; obj_typen = obj_type;
                    add_out = 0; sel_next = 0;

                    if (color_change) colorn = new_color;
                    else colorn = color;

                    case (state)

                              IDLE:
                                        begin
                                   // Deselect currently selected object first if there is one
          if ((next || move_del) && (obj_type != 0)) state_next = DESEL_READY;
                                        // Otherwise move straight to selection
                              else if (next && (obj_type == 0)) state_next = GET_INFO;
                              else if (move_sel) state_next = RELOAD_MOVE;
                                                  else state_next = IDLE;
                                        end

                              DESEL_READY:
                                        begin
                              if (sel_busy) state_next = DESEL_READY;
                                                  else state_next = DESEL_TRIGGER;
                                                  if (nullify) sel_next = 3;
                                                  else sel_next = 2;
                                        end

                              DESEL_TRIGGER:
                                        begin
                                                  if (sel_busy) state_next = DESEL_DONE;
                                                  else state_next = DESEL_TRIGGER;
                                        end
```

```
                    DESEL_DONE:
                              begin
                                        if (sel_busy) state_next = DESEL_DONE;
                              else if (!nullify) state_next = GET_INFO;
                                        else
                                           begin
                                               obj_typen = 0;
                                               obj_numn = 0;
                                               colorn = 0;
                                               state_next = IDLE;
                                           end
                              end

                    GET_INFO:
                              begin
                                        add_out = pos_in;
                                        state_next = INFO_WAIT_1;
                              end

                    INFO_WAIT_1:
                              begin
                                        state_next = INFO_WAIT_2;
                                        add_out = pos_in;
                              end

                    INFO_WAIT_2: state_next = INFO_WAIT_3;
                    INFO_WAIT_3: state_next = INFO_EXTRACT;

                    INFO_EXTRACT:
                              begin
                                if (info_in[12:9] == 0) state_next = IDLE;
                                else
                                 begin
                                        colorn = info_in[8:0];
                                        case (info_in[12:9])
                                        0: obj_typen = 0;           // Null
                                        1: obj_typen = 1;           // Point
                              2: obj_typen = 2;           // Rectangle
                                                  3: obj_typen = 2;
                                        4: obj_typen = 3;           // Line
                                                  5: obj_typen = 3;
                                        6: obj_typen = 4;           // Circle
                                                  7: obj_typen = 4;
                                                  8: obj_typen = 4;
                                                  default: obj_typen = 0;
                                        endcase

                                                   state_next = REQ_A;
                                              obj_numn = info_in[22:13];
end
                              end

                    REQ_A:
                              begin
                                        add_out = 19'h50911 + obj_num;
                                        state_next = REQ_B;
                              end

                    REQ_B:
                              begin
                                        add_out = 19'h55731 + obj_num;
                                        state_next = REQ_C;
                              end

                    REQ_C:
                              begin
                                        add_out = 19'h5A551 + obj_num;
                                        state_next = REQ_D;
                              end

                    REQ_D:
                              begin
                                        add_out = 19'h5F371 + obj_num;
                                        state_next = GET_A;
                              end
```

```
                                GET_A:
                                        begin
                                                state_next = GET_B;
                                                An = info_in;
                                        end

                                GET_B:
                                        begin
                                                state_next = GET_C;
                                                Bn = info_in;
                                        end

                                GET_C:
                                        begin
                                                state_next = GET_D;
                                                Cn = info_in;
                                        end

                                GET_D:
                                        begin
                                                state_next = SEL_READY;
                                                Dn = info_in;
                                        end

                                SEL_READY:
                                        begin
                                                if (sel_busy) state_next = SEL_READY;
                                                else state_next = SEL_TRIGGER;
                                                sel_next = 1;
                                        end

                                SEL_TRIGGER:
                                        begin
                                                if (sel_busy) state_next = SEL_DONE;
                                                else state_next = SEL_TRIGGER;
                                        end

                                SEL_DONE:
                                        begin
                                                if (sel_busy) state_next = SEL_DONE;
                                                else state_next = IDLE;
                                        end

                                RELOAD_MOVE:
                                   begin
                                       An[20:3] = pos_in;
                                       Bn[20:3] = B[20:3] + pos_in + (~A[20:3] + 1);
                                       Cn[20:3] = C[20:3] + pos_in + (~A[20:3] + 1);
                                       Dn[20:3] = D[20:3] + pos_in + (~A[20:3] + 1);
                                       state_next = IDLE;
                                   end

                        endcase
                end

endmodule

// snap_grid.v – Paul Peeling

//COMMAND: Snap_to_grid
// Description: Takes an input point, and snaps it to nearest grid point
// Default grid spacing: every 16 pixels - use of binary arithmetic

module snap_grid
                        (clk, reset, start, busy, pos_x_in, pos_y_in, pos_x_out,
pos_y_out);

input clk, reset, start;
input [9:0] pos_x_in;
input [8:0] pos_y_in;

output busy;
output [9:0] pos_x_out;
output [8:0] pos_y_out;

reg busy;
reg [9:0] pos_x_out;
```

```
reg [8:0] pos_y_out;

wire [9:0] pos_x_grid;
wire [8:0] pos_y_grid;
assign pos_x_grid = {pos_x_in[9:4],4'h0};
assign pos_y_grid = {pos_y_in[8:4],4'h0};

always @ (posedge clk)
         begin
                  if (reset)
                           begin
                                    busy <= 0;
                                    pos_x_out <= 10'hZ;
                                    pos_y_out <= 9'hZ;
                           end
                  else if (start)
                           begin
                                    busy <= 1;
                                    if (pos_x_in[3:0] < 4'h8)
                                                      pos_x_out <= pos_x_grid;
                                    else pos_x_out <= pos_x_grid + 5'h10;

                                    if (pos_y_in[3:0] < 4'h8)
                                                       pos_y_out <= pos_y_grid;
                                    else pos_y_out <= pos_y_grid + 5'h10;
                           end
                  else if (busy) busy <= 0;
         end

endmodule

/// snap_point.v – Paul Peeling

//COMMAND: Snap_to_point

// Description: Takes an input point, and snaps it to nearest object point
// by searching through memory using a prescribed pattern in ROM

module snap_point
                  (clk, reset, start, busy, pos_x_in, pos_y_in,
                  pos_x_out, pos_y_out, mem_get, mem_add);

input clk, reset, start;
input [9:0] pos_x_in;
input [8:0] pos_y_in;
input [35:0] mem_get;

output busy;
output [9:0] pos_x_out;
output [8:0] pos_y_out;
output [18:0] mem_add;

reg busy;
wire [9:0] pos_x_out;
wire [8:0] pos_y_out;
wire [18:0] mem_add;

parameter PATTERN_SIZE = 177;         // Max search depth
parameter DATA_DELAY = 5;    // Pipeline delay from here to SRAM for data

// States
reg [1:0] state, next;

parameter IDLE = 0;
parameter SEARCH = 1;
parameter FOUND = 2;
parameter NOT_FOUND = 3;

reg [7:0] count, count_n;
wire [18:0] shift;

pattern PAR_ROM (.addr(count), .clk(clk), .dout(shift)); //Contains address offsets in
order
//assign shift = count;

always @ (posedge clk)
         begin
```

```
                    if (reset)
                              begin
                                        state <= IDLE;
                                        count <= 0;
                              end
                    else
                              begin
                                        state <= next;
                                        count <= count_n;
                              end
          end

always @ (state or count or start or mem_get)
          begin



                    case (state)
                              IDLE:
                                        begin
                                                  busy = 0;
                                                  if (start)
                                                            begin
                                                                      next = SEARCH;
                                                                      count_n = 0;
                                                            end
                                                  else
                                                            begin
                                                                      next = IDLE;
                                                                      count_n = count;
                                                            end
                                        end

                              SEARCH:
                                        begin
                                                  busy = 1;
                                                  /*        if ((mem_get[12:9] == 2) ||                     // Special
                                                            (mem_get[12:9] == 4) ||
                                                            (mem_get[12:9] == 6) ||
                                                            (mem_get[12:9] == 7))
                                                            begin   */
                                                  if (mem_get != 0) begin
          count_n = count + (~DATA_DELAY + 1);  // Data came from 3 addresses ago!
                                                                      next = FOUND;
                                                            end
                                        else if (count == PATTERN_SIZE + DATA_DELAY)
                                                            begin
                                                                      count_n = 0;
                                                                      next = NOT_FOUND;
                                                            end
                                                  else
                                                            begin
                                                            count_n = count + 1;
                                                                      next = SEARCH;
                                                            end
                                        end

                              FOUND:
                                        begin
                                                  busy = 1;
                                                  count_n = count;
                                                  next = IDLE;
                                        end

NOT_FOUND:                                        // Returns unsnapped positions
                                        begin
                                                  busy = 1;
                                                  count_n = 0;
                                                  next = IDLE;
                                        end

                    endcase
          end

assign pos_x_out = pos_x_in + shift[18:9];
assign pos_y_out = pos_y_in + shift[8:0];
assign mem_add = busy ? {pos_x_out, pos_y_out} : 19'h0;
```

```
endmodule

// snapper.v – Paul Peeling

// Snapper module
// Description: Takes input position from synchronizer
// and snaps it, outputting to the area map

// Need to pass and click signals through this module
// to delay until the snapping command has been carried out

module snapper
          (clk, reset, pos_x_in, pos_y_in,
           pos_x_out, pos_y_out, click_in, click_out, command,
           pos_x_sg, pos_y_sg, pos_x_sp, pos_y_sp,
           sg_start, sg_busy, sp_start, sp_busy,
           snap_force_mem);

input clk, reset, click_in;
input [3:0] command;
input [9:0] pos_x_in, pos_x_sg, pos_x_sp;
input [8:0] pos_y_in, pos_y_sg, pos_y_sp;

reg [9:0] pos_x_snap;
reg [8:0] pos_y_snap;

input sg_busy, sp_busy;
output sg_start, sp_start;

output [9:0] pos_x_out;
output [8:0] pos_y_out;
output click_out;

reg click_out;
reg [9:0] pos_x_out;
reg [8:0] pos_y_out;

reg snap_busy;              // Generic next and busy signals which will be
multiplexed
reg snap_start;
reg sg_start, sp_start;

reg [1:0] mode, mode_next;            // Snap mode register
reg [2:0] state, next;

parameter IDLE = 0;
parameter SNAP_READY = 1;
parameter SNAP_TRIGGER = 2;
parameter SNAP_DONE = 3;
parameter SNAP_HOLD = 4;

output snap_force_mem;
wire snap_force_mem;
assign snap_force_mem = (state != IDLE);

always @ (posedge clk)
          begin
                    if (reset)
                              begin
                                        state <= IDLE;
                                        mode <= 0;
                              end
                    else
                              begin
                                        state <= next;
                                        mode <= mode_next;
                              end

// Command Routing map for next and busy signals

                    case (mode)

                              1:
                                        begin
                                                  sg_start <= snap_start;
                                                  snap_busy <= sg_busy;
```

```
                                        pos_x_snap <= pos_x_sg;
                                        pos_y_snap <= pos_y_sg;
                                end

                        2:
                                begin
                                        sp_start <= snap_start;
                                        snap_busy <= sp_busy;
                                        pos_x_snap <= pos_x_sp;
                                        pos_y_snap <= pos_y_sp;
                                end


                default: snap_busy <= 0;                        // No snapping

                endcase

                if ((mode == 0) || (pos_x_in < 10))
                        begin
                                click_out <= click_in;
                                pos_x_out <= pos_x_in;
                                pos_y_out <= pos_y_in;
                        end
                else if (state == SNAP_HOLD)
                        begin
                                click_out <= 1;
                                pos_x_out <= pos_x_snap;
                                pos_y_out <= pos_y_snap;
                        end
                else
                        begin
                                click_out <= 0;
                                pos_x_out <= pos_x_snap;
                                pos_y_out <= pos_y_snap;
                        end

        end

always @ (state or command or click_in or mode or pos_x_in or pos_y_in
                        or snap_busy or reset)
        begin

                snap_start = 0;

                case (command)
                        5: mode_next = 1;                                                       // Snap-to
                        6: mode_next = 2;                                                       // Snap-to
                        default: mode_next = mode;
                endcase

                if ((mode != 0) && (pos_x_in >= 10))   // Do not snap on toolbars
                        begin
                                case (state)

                                        IDLE:
                                                begin
                                                        snap_start = 0;
                                        if (click_in) next = SNAP_READY;
                                                        else next = IDLE;
                                                end

                                        SNAP_READY:
                                                begin
                                                        snap_start = 1;
                                        if (snap_busy) next = SNAP_READY;
                                        else next = SNAP_TRIGGER;
                                                end

                                        SNAP_TRIGGER:
                                                begin
                                                        snap_start = 0;
                                        if (snap_busy) next = SNAP_DONE;
                                        else next = SNAP_TRIGGER;
                                                end

                                        SNAP_DONE:
                                                begin
```

```
                                                  snap_start = 0;
                                        if (snap_busy) next = SNAP_DONE;
                                        else next = SNAP_HOLD;
                                                  end

                                        SNAP_HOLD:
                                                  begin
                                                  snap_start = 0;
                                                  next = IDLE;
                                                  end

                                        default: next = IDLE;

                                    endcase
                              end

                    else next = IDLE;

          end
endmodule

/// superplex.v – Paul Peeling

// Multiplexer for memory
// Description:             A large multiplexing unit between the
//                                        command and other modules
//                                        providing memory output lines

module superplex
                    (clk, command,
                     mem_write, memory_in, memory_address,
                     dp_we, dp_data, dp_add,              // Draw_point
                     dr_we, dr_data, dr_add,              // Draw_rect
                     dl_we, dl_data, dl_add,              // Draw_line
                     dc_we, dc_data, dc_add,              // Draw_circle
                     sp_add,                                                  //
Snap_to_point
                     sel_busy, obj_type, sel_add,//Select signals required for re-draw
                     snap_force_mem);  // Aquire memory use for snap_to_point

input clk;

// This is the multiplexing signal
// combined from

input [3:0] command;

output mem_write;                                     // SRAM WE
output [35:0] memory_in;                        // SRAM data in
output [18:0] memory_address;                   // SRAM address

reg mem_write;
reg [35:0] memory_in;
reg [18:0] memory_address;

input dp_we, dr_we, dl_we, dc_we, sel_busy, snap_force_mem;
input [35:0] dp_data, dr_data, dl_data, dc_data;
input [18:0] dp_add, dr_add, dl_add, dc_add, sp_add, sel_add;
input [2:0] obj_type;

// Reserve 0 for NO_OP
parameter DRAW_POINT = 1;
parameter DRAW_RECT = 2;
parameter DRAW_LINE = 3;
parameter DRAW_CIRCLE = 4;
parameter SELECT = 7;

// Object Types
parameter POINT = 1;
parameter RECTANGLE = 2;
parameter LINE = 3;
parameter CIRCLE = 4;

always @ (posedge clk)
          begin

             if (snap_force_mem)
```

```
      begin
                                        mem_write <= 0;
                                        memory_address <= sp_add;
                                        memory_in <= 0;
                  end
            else if (command > 6)
               begin
                                        if (sel_busy)
                                                 begin
                                                 case (obj_type)
                                                          POINT:
                                                          begin
                                                          mem_write <= dp_we;
                                                 memory_address <= dp_add;
                                                 memory_in <= dp_data;
                                                                  end
                                                                  RECTANGLE:
                                    begin
                                                          mem_write <= dr_we;
                                                 memory_address <= dr_add;
                                                 memory_in <= dr_data;
                                                                  end
                                                                  LINE:
                                                 begin
                                                 mem_write <= dl_we;
                                                 memory_address <= dl_add;
                                                 memory_in <= dl_data;
                                                                  end
                                                                  CIRCLE:
                                                                  begin
                                                 mem_write <= dc_we;
                                                 memory_address <= dc_add;
                                                 memory_in <= dc_data;
                                                                  end
                                                                  default:
                                                                  begin
                                                 mem_write <= 0;
                                                 memory_address <= 0;
                                                 memory_in <= 0;

end
                                                          endcase
                                                 end
                                    else
                                                 begin
                                                          mem_write <= 0;
                                                 memory_address <= sel_add;
                                                          memory_in <= 0;
                                                 end
                                    end
else
   begin
case (command)

                        DRAW_POINT:
                                    begin
                                                 mem_write <= dp_we;
                                                 memory_address <= dp_add;
                                                 memory_in <= dp_data;
                                    end

                        DRAW_RECT:
                                    begin
                                                 mem_write <= dr_we;
                                                 memory_address <= dr_add;
                                                 memory_in <= dr_data;
                                    end

                        DRAW_LINE:
                                    begin
                                                 mem_write <= dl_we;
                                                 memory_address <= dl_add;
                                                 memory_in <= dl_data;
                                    end

                        DRAW_CIRCLE:
                                    begin
```

```
                                        mem_write <= dc_we;
                                        memory_address <= dc_add;
                                        memory_in <= dc_data;
                                end

                        SELECT:
                                begin


                                end

                        default:  // No_op (includes snap-to-grid)
                                begin
                                        mem_write <= 0;
                                        memory_address <= 0;
                                        memory_in <= 0;
                                end

                endcase
          end
        end

endmodule



/// user_synch.v – Paul Peeling

// Synchronizer for user interface

module usersynch (clk, button, button_sync);

// Synchronization of all asynchronous inputs
// Two back-to-back registers used

        input clk;
        input [1:0] button;

        output [1:0] button_sync;
        reg [1:0]  button_sync;

        reg [1:0] button_sync2;

        always @ (posedge clk)
        begin
                button_sync2 <= button;
                button_sync <= button_sync2;
        end

endmodule
```

```
/// vga.v – Faraz Ahmad

`timescale  1 ps / 1 ps

module vga(fpga_clock, empty, display_busy, reset_DAC, vga_out_sync_b,
                                                vga_out_vsync, vga_out_hsync,
vga_out_blank_b,
                                                vga_out_red, vga_out_green,
vga_out_blue, reset_sync,
                                                frame_buffer_data, address,
SRAMaddress, SRAMdata, SRAMdataout,
                                                control_unit_state, WEbar, CEbar,
OEbar, RE, ROMdata, ROMaddress,
                                                stylus_x, stylus_y);

          input [9:0] stylus_x, stylus_y;
          wire [18:0] stylus_pos;
          assign stylus_pos = {stylus_y[8:0], stylus_x[9:0]};

          input fpga_clock, empty, reset_sync;
          input [8:0] frame_buffer_data;
          input [18:0] address;
          input [7:0] ROMdata;
          output [10:0] ROMaddress;
          output display_busy, reset_DAC, vga_out_sync_b, vga_out_blank_b,
vga_out_hsync,
                           vga_out_vsync;
          output [7:0]vga_out_red;
          output [7:0]vga_out_green;
          output [7:0]vga_out_blue;
          output WEbar, CEbar, OEbar, RE;

          output [18:0] SRAMaddress;
          input [8:0] SRAMdata;
          output [8:0] SRAMdataout;

          output [3:0] control_unit_state;

          wire [10:0] hsync_active_video_counter;
          wire [10:0] linecounter;
          wire [10:0] rom_pixel_count;
          wire [10:0] rom_line_count;
          wire [10:0] test_pixel_count;
          wire [10:0] test_line_count;


          wire [23:0] rgb;

          wire [7:0] vga_out_red, vga_out_blue, vga_out_green;
          wire [8:0] SRAMdataout;

          wire vga_out_pixel_clock;

          assign vga_out_red = rgb[23:16];
          assign vga_out_green = rgb[15:8];
          assign vga_out_blue = rgb[7:0];
          wire vsync, hsync, blank_control;

          //assign reset_sync = ~button0;         // assign reset


    //
     // Generate the pixel clock (25.10MHz)
     // synthesis attribute period of clock_27mhz is 10ns;

//   BUFG clock_27mhz_buf (.O(clock27b), .I(clock27));

/*   DCM pixel_clock_dcm (.CLKFB(clock27b),
                          .CLKIN(clock_27mhz),
                          .RST(1'b0),
                          .CLK0(clock27),
                          .CLKFX(pixel_clock)
                          );                                          */
     // synthesis attribute DLL_FREQUENCY_MODE of pixel_clock_dcm is "LOW"
     // synthesis attribute DUTY_CYCLE_CORRECTION of pixel_clock_dcm is "TRUE"
     // synthesis attribute STARTUP_WAIT of pixel_clock_dcm is "TRUE"
     // synthesis attribute DFS_FREQUENCY_MODE of pixel_clock_dcm is "LOW"
```

```
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 14
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 13
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of pixel_clock_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of pixel_clock_dcm is 0
   // synthesis attribute clkin_period of pixel_clock_dcm is "37.04ns"


          syncgen2 test_syncgen2
                              (
                              .clock(fpga_clock),
                              .vsync(vsync),
                              .hsync(hsync),
                              .blank(blank_control),
                              .sync_start(sync_start),
                              .reset_sync(reset_sync),
                              .line_count(linecounter),
                              .vblank(vblank),
                              .pixel_count(hsync_active_video_counter),
                              .rom_line_count(rom_line_count),
                              .rom_pixel_count(rom_pixel_count),
                              .rom_vblank(rom_vblank),
                              .test_line_count(test_line_count),
                              .test_pixel_count(test_pixel_count));

          control_unit test_control_unit(

                              .clock(fpga_clock),
                              .reset_sync(reset_sync),
                              .hsync(hsync),
                              .vsync(vsync),
                              .vga_out_vsync(vga_out_vsync),
                              .vga_out_hsync(vga_out_hsync),
                              .blank(blank_control),
                              .sync_start(sync_start),
                              .vga_out_blank_b(vga_out_blank_b),
                              .vga_out_sync_b(vga_out_sync_b),
                              .reset_DAC(reset_DAC),
                              .empty(empty),
                              .display_busy(display_busy),
                              .rgb(rgb),
                              .linecounter(linecounter),
                              .hsync_active_video_counter(hsync_active_video_counter),
                              .vblank(vblank),
                              .address(address),
                              .frame_buffer_data(frame_buffer_data),
                              .SRAMdatain(SRAMdata),
                              .SRAMdataout(SRAMdataout),
                              .SRAMaddress(SRAMaddress),
                              .state(control_unit_state),
                              .WEbar(WEbar),
                              .OEbar(OEbar),
                              .CEbar(CEbar),
                              .RE(RE),
                              .rom_pixel_count(rom_pixel_count),
                              .rom_line_count(rom_line_count),
                              .rom_vblank(rom_vblank),
                              .ROMdata(ROMdata),
                              .ROMaddress(ROMaddress),
                              .test_line_count(test_line_count),
                              .test_pixel_count(test_pixel_count),
                              .stylus_pos(stylus_pos));



endmodule


/// control_unit – Faraz Ahmad w/ modifications by Paul Peeling

`timescale  1 ps / 1 ps

module control_unit(sync_start, hsync, vsync, blank, vblank,
                    vga_out_blank_b, vga_out_sync_b, reset_DAC, display_busy, empty,
                    clock, reset_sync, rgb, linecounter, hsync_active_video_counter,
vga_out_vsync,
```

```
                                    vga_out_hsync,
                                    frame_buffer_data, SRAMdatain, SRAMdataout,
SRAMaddress, address,
                                    state, WEbar, OEbar, CEbar, RE,
                                    rom_pixel_count, rom_line_count,
                                    ROMdata, ROMaddress, rom_vblank,
                                    test_pixel_count, test_line_count, stylus_pos);

      input [18:0] stylus_pos;

    input blank, vsync, hsync, empty, clock, reset_sync, vblank, rom_vblank;
    input [10:0] linecounter;
    input [10:0] hsync_active_video_counter;
    input [10:0] rom_pixel_count;
    input [10:0] rom_line_count;

    input [10:0] test_pixel_count;
    input [10:0] test_line_count;

    input [7:0] ROMdata;
    output [10:0] ROMaddress;
    output display_busy, reset_DAC, vga_out_blank_b, vga_out_sync_b, sync_start,
vga_out_vsync, vga_out_hsync;
    input [8:0] frame_buffer_data;
    input [18:0] address;
    output [18:0] SRAMaddress;
    input [8:0] SRAMdatain;
    output [8:0] SRAMdataout;
    output [23:0] rgb;
    output WEbar, OEbar, CEbar, RE;
    output [3:0]state;
    reg initialise;

    reg WEbar, OEbar, CEbar, RE;

          reg display_busy, reset_DAC, sync_start;
          reg [18:0] SRAMaddress, SRAMaddress_2;
          reg [8:0] SRAMdataout, SRAMdataout_1, SRAMdataout_2, SRAMdataout_3,
SRAMdataout_4, SRAMdataout_5;
          reg [18:0] linecounter2;
          reg [18:0] linecounter3;
          reg [10:0] ROMaddress;
          reg [23:0] rgb;
          reg sync_b;
          reg blank_b1;
          reg [3:0] state;
          reg [3:0] next;
          reg pixel_bit;
          reg [18:0]counter;

   reg vga_out_sync_b, vga_out_blank_b;
   reg hsync2, vga_out_hsync, vsync2, vga_out_vsync;

   wire [8:0] initialised_data;


          parameter RESET             =        0;
          parameter Active_Video   =           1;
          parameter Check             =                  2;
          parameter Transfer_Data  =           3;
          parameter ROM_Screen        =                  4;


          parameter frame_height  =     479;


          // initialise ROM SCreen block
              initialise_mod test_initialise(
                             .clock(clock),
                             .reset_sync(reset_sync),
                             .rom_line_count(rom_line_count),
                             .rom_pixel_count(rom_pixel_count),
                             .initialised_data(initialised_data),
                             .rom_vblank(rom_vblank),
                             .initialise(initialise),
                             .ROMdata(ROMdata));
```

```
   ////////////////////////////////////////////////////////////////////////
   //
   // Blanking & Sync Signals processing
   //
   ////////////////////////////////////////////////////////////////////////

 always @ (posedge clock)
 begin
          sync_start<=1; // starts sync gen
          // Blanking
           blank_b1 <= blank;
           vga_out_blank_b <= blank_b1;

          // Composite sync

           sync_b <= hsync ^ vsync;              // could be NAND
           vga_out_sync_b <= sync_b;


                    // Delay hsync and vsync by two cycles to compensate for 2 cycles
of
                    // pipeline delay in the DAC.
                    hsync2 <= hsync;
                    vga_out_hsync <= hsync2;
                    vsync2 <= vsync;
                    vga_out_vsync <= vsync2;

          SRAMdataout_3 <= SRAMdataout_4;
          SRAMdataout_2 <= SRAMdataout_3;
          SRAMdataout_1 <= SRAMdataout_2;
          SRAMdataout <= SRAMdataout_1;

          SRAMaddress <= SRAMaddress_2;
end

   ////////////////////////////////////////////////////////////////////////
   //
   // FSM
   //
   ////////////////////////////////////////////////////////////////////////

always @ (posedge reset_sync or posedge clock)

          if (reset_sync)
                    state <= RESET;
//        else if (flush_set)
//        state <= FLUSH;
          else   state <= next;


always @ (state or vblank or hsync_active_video_counter or linecounter or SRAMdatain
                                    or address or frame_buffer_data or
rom_line_count or rom_pixel_count or pixel_bit or empty or ROMdata)
begin

CEbar <=0;

          case (state)

          RESET:

                    begin
                                        initialise <=1;
                                        next <= ROM_Screen;
                                        RE<=0;
                                        sync_start <=1;

                                        //TEST SETUP
                                        //next <=Active_Video;
                    end

          ROM_Screen:

                    begin

                                        RE <=0;
                                        WEbar <=0; // writing to SRAM
```

```
                                        OEbar <=1; // writing to SRAM
                                        SRAMaddress_2 <= {rom_line_count[8:0], 5'b00000,
rom_pixel_count[4:0]};          //19bit address
                                        ROMaddress <= {rom_line_count[8:0],
rom_pixel_count[4:3]};

                                        //linecounter2 <= {3'b000, rom_line_count[8:0],
2'b00, rom_pixel_count[4:0]};
                                        //linecounter3 <= {1'b0, rom_line_count[8:0],
9'b000000000};

                                        //SRAMaddress  <= linecounter2 + linecounter3;
                                        SRAMdataout_4 <= initialised_data;

                                        rgb <= {8'hFF, 8'h00, 8'h00}; // RED TEST SCREEN
SHOWING INITIALISATION

                                        if (rom_vblank) //if finished initialising
screen
                                                next <= Active_Video;
                                        else
                                                next <= ROM_Screen;

                    end

          Active_Video: //reading from FSM

          begin
                    if (vblank==1)
                                        next <= Transfer_Data;
                    else
                                        begin
                                        RE<=0; // deactivate FIFO
                                        display_busy <=1;
                                        OEbar <=0;         //reading
                                        WEbar <=1;         //reading

                                        if (({linecounter[8:0],
hsync_active_video_counter[9:0]} == stylus_pos) ||
                                                ({linecounter[8:0],
hsync_active_video_counter[9:0]} == (stylus_pos + 19'h1)) ||
                                                ({linecounter[8:0],
hsync_active_video_counter[9:0]} == (stylus_pos + 19'h200)) ||
                                                ({linecounter[8:0],
hsync_active_video_counter[9:0]} == (stylus_pos + 19'h201)))
                                                        rgb <= 24'hFFFF00;
                                        else if ((linecounter[4:0] != 5'b0) &&
(hsync_active_video_counter[4:0] != 5'b0))
                                                        rgb <= {SRAMdatain[2:0], 5'b00000,
SRAMdatain[5:3], 5'b00000, SRAMdatain[8:6], 5'b00000};
                                        else rgb <= 24'h0000FF;

                                        SRAMaddress_2 <= {linecounter[8:0],
hsync_active_video_counter[9:0]} + 19'h2;

                                        next <= Active_Video;
                                        end
          end

          Transfer_Data: //writing to SRAM during v blanking
          begin
          display_busy <=0;

                    if (vblank==0)
                              begin
                                        next <= Active_Video;
                                        RE <= 0;
                                        WEbar <= 1;
                                        OEbar <= 0;
                              end

                    else
                              begin

                                                  if (empty==0) // we have data to
update
                                                            begin
```

```
                                                                SRAMaddress_2 <=
address;
                                                                SRAMdataout_4 <=
frame_buffer_data;
                                                                next <=
Transfer_Data;
                                                                RE <=1; // activate
FIFO
                                                                OEbar<=1; //writing
                                                                WEbar<=0; //writing

                                                        end
                                                else
                                                        begin
                                                                RE <=0;
                                                                WEbar <=1;
//reading from SRAM
                                                                OEbar <=0;
//reading
                                                                next <=
Transfer_Data;                          //keep in this state till vblank goes off
                                                        end
                                end


          end

endcase
end // always


endmodule




/// initialise.v – Faraz Ahmad

module initialise_mod (rom_line_count, rom_pixel_count, initialised_data, clock,
initialise, rom_vblank,  reset_sync, ROMdata);

input [10:0] rom_pixel_count;
input [10:0] rom_line_count;
input [7:0] ROMdata;
output [8:0] initialised_data;
reg [8:0] initialised_data;
input initialise, rom_vblank, reset_sync;
input clock;

reg done;
reg pixel_bit;
reg [3:0] state;
reg [3:0] next;
reg [8:0] SRAMdataout;


parameter RESET =0;
parameter PIXEL_BIT =1;

parameter blue =  9'b000000111;
parameter red =   9'b111000000;
parameter green = 9'b000111000;
parameter white = 9'b111111111;
parameter black = 9'b000000000;
parameter cyan = 9'b000111111;
parameter random  = 9'b000101010;

always @(posedge clock)

          if (reset_sync || rom_vblank)
                    begin
                              state <= RESET;
                    end
          else
                    state <= next;
```

```
always @(state)

begin

	case (state)

	RESET:

	begin
		done <=1;
		pixel_bit <= 0;
		initialised_data <= black;
		if (initialise ==1)
			next <= PIXEL_BIT;
		else
			next <= RESET;
	end

	PIXEL_BIT:
	begin

	//	initialised_data <= 9'b111111111;
		case (rom_pixel_count[2:0])

					000:
					pixel_bit <= ROMdata[7];

					001:
					pixel_bit <= ROMdata[6];

					010:
					pixel_bit <= ROMdata[5];

					011:
					pixel_bit <= ROMdata[4];

					100:
					pixel_bit <= ROMdata[3];

					101:
					pixel_bit <= ROMdata[2];

					110:
					pixel_bit <= ROMdata[1];

					111:
					pixel_bit <= ROMdata[0];

	endcase


		if (pixel_bit) //active colour
			begin
			if ((rom_line_count >= 0) && (rom_line_count < 32))  //
draw point
				begin
					initialised_data <=  green;
				end
			else if ((rom_line_count >= 32) && (rom_line_count < 64))
	// draw rectangle
				begin
					initialised_data <= cyan;
				end
			else if ((rom_line_count >= 64) && (rom_line_count < 96))
	// draw line
				begin
					initialised_data <= green;
				end
			else if ((rom_line_count >= 96) && (rom_line_count <
128))    // draw circle
				begin
					initialised_data <= cyan;
				end
			else if ((rom_line_count >= 128) && (rom_line_count <
160))    // snap to point
				begin
					initialised_data <= green;
```

```
                                end
                            else if ((rom_line_count >= 160) && (rom_line_count <
192))    // select button
                                begin
                                        initialised_data <= random;
                                end
                            else if ((rom_line_count >= 192) && (rom_line_count <
224))    // delete button
                                begin
                                        initialised_data <= cyan;
                                end
                            else if ((rom_line_count >= 224) && (rom_line_count <
256))    // move
                                begin
                                        initialised_data <= green;
                                end
                            else if ((rom_line_count >= 256) && (rom_line_count
<288))   // resize
                                begin
                                        initialised_data <= random;
                                end
                            else if ((rom_line_count >= 288) && (rom_line_count
<320))   // red button
                                begin
                                        initialised_data <= cyan;
                                end
                            else if ((rom_line_count >= 320) && (rom_line_count
<352))   // green button
                                begin
                                        initialised_data <= green;
                                end
                            else if ((rom_line_count >= 352) && (rom_line_count
<384))   // blue button
                                begin
                                        initialised_data <= random;
                                end
                            else if ((rom_line_count > 384))      // white button
                                begin
                                        initialised_data <= white;
                                end
                            end //if(pixel_bit)
                    else    initialised_data <= white;    //background is going to be
white

                    end //PIXEL_BIT
            endcase

end //always

endmodule
```

```
/// syncgen2.v – Faraz Ahmad

module syncgen2 (clock, vsync, hsync, blank, sync_start, reset_sync, line_count,
                                                pixel_count, vblank, rom_pixel_count,
rom_line_count, rom_vblank, test_pixel_count, test_line_count);
     input clock, sync_start, reset_sync;
     output hsync, vsync, blank, vblank, rom_vblank;
          wire blank;
          wire vblank;
          output [10:0] pixel_count;
          output [10:0] line_count;
          output [10:0] rom_pixel_count;
          output [10:0] rom_line_count;
          output [10:0] test_pixel_count;
          output [10:0] test_line_count;
          reg [10:0] pixel_count; // Counts pixels in each line
          reg [10:0] line_count; // Counts lines in each frame
          reg [10:0] rom_pixel_count;
          reg [10:0] rom_line_count;
          reg [10:0] test_pixel_count;
```

```
         reg [10:0] test_line_count;


`define H_ACTIVE   640 // pixels
`define H_FRONT_PORCH       16 // pixels
`define H_SYNCH             96 // pixels
`define H_BACK_PORCH        48 // pixels
`define H_TOTAL             800 // pixels

`define TOOLBAR_WIDTH                32 // pixels
`define TOOLBAR_LENGTH      448 // pixels
`define ROM_V_TOTAL         524 // pixels
//`define H_BACK_PORCH      48 // pixels
//`define H_TOTAL           800 // pixels

`define V_ACTIVE            480 // actual lines 3// for simulation
`define V_FRONT_PORCH         11 // lines
`define V_SYNCH                2 // lines
`define V_BACK_PORCH          31 // lines
`define V_TOTAL             524 // lines

         // Blanking
   assign blank = ((pixel_count<`H_ACTIVE) & (line_count<`V_ACTIVE));
   assign vblank = (line_count > `V_ACTIVE);
   assign rom_vblank = (rom_line_count > `V_ACTIVE);



         parameter hsync_front_porch =   16;
   parameter hsync_end     =       959;
   parameter hsync_duration =      96;
   parameter hsync_back_porch =    48;
   parameter row_period    =      800;
   parameter frame_height  =                    479;        // actual
   parameter vsync_start   =       600;
   parameter vsync_end     =       604;
   parameter frame_period  =       627;
   parameter vsync_front_porch =   11;
   parameter vsync_back_porch =       31;          //actual
   parameter vsync_duration   =    2;
   parameter hsync_active_video =  640;
   parameter vsync_active_video =  480;

         ////////////////////////////////////////////////////////////////////////
   //
   // Internal signals
   //
   ////////////////////////////////////////////////////////////////////////

   reg hsync, vsync;

         always @(posedge clock)
     if (reset_sync)
       begin
            pixel_count <= 0;
            line_count <= 0;
       end
     else if (pixel_count == (`H_TOTAL-1)) // last pixel in the line
       begin
            pixel_count <= 0;
            if (line_count == (`V_TOTAL-1)) // last line of the frame
              line_count <= 0;
            else
              line_count <= line_count + 1;
       end
     else
       pixel_count <= pixel_count +1;


// Block to produce counters for ROM initialisation

always @(posedge clock)
     if (reset_sync)
       begin
            rom_pixel_count <= 0;
            rom_line_count <= 0;
       end
```

```
     else if (rom_pixel_count == (`TOOLBAR_WIDTH-1)) // last pixel in the line
       begin
            rom_pixel_count <= 0;
            if (rom_line_count == (`ROM_V_TOTAL-1)) // last line of the frame
              rom_line_count <= 0;
            else
              rom_line_count <= rom_line_count + 1;
       end
     else
       rom_pixel_count <= rom_pixel_count + 1;

            // Block to produce counters for test screen
always @(posedge clock)
     if (vblank==0) // needs to only work when vblank is high
       begin
            test_pixel_count <= 0;
            test_line_count <= 0;
       end
     else if (test_pixel_count == (`H_TOTAL-1)) // last pixel in the line
       begin
            test_pixel_count <= 0;
            if (test_line_count == (`V_TOTAL-1)) // last line of the frame
              test_line_count <= 0;
            else
              test_line_count <= test_line_count + 1;
       end
     else
       test_pixel_count <= test_pixel_count +1;


 always @ (posedge clock)
     begin
         if (reset_sync)
           begin
              hsync <= 1;
              vsync <= 1;
           end
         else
           begin

              // Horizontal sync
              if (pixel_count == (`H_ACTIVE+`H_FRONT_PORCH))
                hsync <= 0; // start of h_sync
              else if (pixel_count == (`H_ACTIVE+`H_FRONT_PORCH+`H_SYNCH))
                hsync <= 1; // end of h_sync

              // Vertical sync
              if (pixel_count == (`H_TOTAL-1))
                begin
                     if (line_count == (`V_ACTIVE+`V_FRONT_PORCH))
                       vsync <= 0; // start of v_sync
                     else if (line_count == (`V_ACTIVE+`V_FRONT_PORCH+`V_SYNCH))
                       vsync <= 1; // end of v_sync
              end

           end

end

endmodule
```