Appendix

Group 14: Motion Sensor Baseball Game

Jin Hock Ong
Chris Falling

Some verilog files that are generated with Core Gen is not included in this appendix.

Motion Sensor Interface

```
// Jin Hock
// Writing a top module to connect the motion sensor interface

module motion_sensor(clk, reset, tv_in, //input
                                   Readen,  Pagedone,at_even_field,next,
Y, LEDpos, //output from decoder ::debugging purposes
                                    error, coordinate_done, //output
from coordinator :: debugging purposes
                                    ledx_in,ledx, ledy_in,ledy,
                                    bottom_point_out, top_point1_out,
top_point2_out,
                                    maxluminance,blank_signal, tv_out,
counterx, top_pointup, bottom_point, top_pointdown, backswing,
speed);// from coordinator to mirror :: for debugging purposes

input clk, reset;
input [9:0] tv_in; // from 7185 chip

output [9:0] tv_out;
// from coordinator
output error, coordinate_done, LEDpos;

//from decoder
output Readen, Pagedone,at_even_field;
output [1:0] next;
output [7:0] Y, maxluminance;
output blank_signal;
output [10:0] counterx;

// from coordinator to mirror
output [9:0] ledx_in,ledx;
output [8:0] ledy_in,ledy;

//from register
output [18:0] bottom_point_out, top_point1_out, top_point2_out;
output [18:0] top_pointup, bottom_point, top_pointdown;

output backswing;
output [19:0] speed;


wire multdone, sqrtdone, startmult, startsqrt;
wire [9:0]  xlen;
wire [10:0] ylen;
wire [19:0] sqrdx, sqrdy;
wire [37:0] sum;
wire [18:0] sqrt_val;
wire multdone1;



tv_sync sync(clk, tv_in, tv_out);
//module tv_sync (clk, data_in, data_out);


decoder tofilter(clk, reset, tv_out, ledx_in, ledy_in, Y, Readen,
Pagedone,at_even_field,next, blank_signal,counterx);
//module decoder2(clk, reset, v_in, ledx, ledy, Y, Readen,
Pagedone,at_even_field,next, blank_signal);

maxY maxfinder(clk, Readen, counterx, reset,Y, maxluminance);
//module maxY(clk, Readen, reset,Y, maxluminance);


filter tocoordinator(clk, Readen, ledx_in, ledy_in, Y, ledx, ledy,
LEDpos);
//module filter(clk, Readen, ledx_in, ledy_in, Y, ledx, ledy, LEDpos);

coordinator tomirror(clk, reset, LEDpos, ledx, ledy, Pagedone,
bottom_point, top_pointup, top_pointdown, error,coordinate_done);
//module coordinator(clk, reset, LEDpos, ledx, ledy, Pagedone,
bottom_point, top_point1, top_point2, error,coordinate_done);


register tolatch(clk,reset, error,coordinate_done, bottom_point,
top_pointup, top_pointdown, //inpput
                  bottom_point_out, top_point1_out, top_point2_out);
//output
//     register(clk,reset, error,coordinate_done, bottom_point,
top_pointup, top_pointdown, //input
//                      bottom_point_out, top_point1_out,
top_point2_out); //output

speedometer speedtrack(clk, reset, coordinate_done, bottom_point,
top_pointup, sqrt_val, multdone, sqrtdone, sqrdx, sqrdy,  //input
                         startmult, startsqrt, xlen,ylen, sum,  speed,
backswing); //output

//module speedometer (clk, reset, coordinate_done, bottom_pt, top_pt,
sqrt_val, multdone, sqrtdone, sqrdx, sqrdy,  //input
      //                      startmult, startsqrt, xlen,ylen, sum,
speed, backswing); //output


square  squarex(clk, xlen,    xlen, sqrdx,      startmult,  multdone);
square  squarey(clk, ylen,    ylen, sqrdy,      startmult,  multdone1);
//module square ( clk,  a,    b,    q,    nd,   rdy);

sqrt squareroot(sum, clk, startsqrt, sqrt_val, sqrtdone);
//module sqrt(   x_in,   clk,   ce,   x_out,   rdy   )
```

```
endmodule


// Jin Hock
// tv_in Synchronizer to synchronize with the system clock
// the clock input should be from the 7185 chip clk (tv_line_in_clk1)


module tv_sync (clk, data_in, data_out);
input clk;
input [9:0] data_in;
output [9:0]data_out;
reg [9:0] data_out, temp_data_out;


always @ (posedge clk) begin
temp_data_out<=data_in;
data_out<=temp_data_out;
end

endmodule


// Jin Hock Ong
// Last modified on May 8th 3.19 pm
// Reset is active high
// Decoder takes in video input as the input
//use two counters to track the coordinate

module decoder(clk, reset, v_in, ledx, ledy, Y, Readen,
Pagedone,at_even_field,next, blank_signal, counterx);

input clk, reset;
input [9:0] v_in; //10 bits from AD7185


output [9:0] ledx;
output [8:0] ledy;
output [7:0] Y; // Y refers to luminance value :: connect to
tv_in[19:10]
output Readen; // to be connected to filter since we only want to read
the luminance values :: read every other clock cycle
output Pagedone; // to be connected to coordinator to indicate that it
has finished decoding a page
output at_even_field, blank_signal;
output [1:0] next;
output [10:0] counterx;

reg [10:0] counterx;
reg [7:0]  countery;
reg [1:0] next;
//reg [7:0] Y;
reg at_even_field; // tracking register to note that the decoder is at
even field
reg Pagedone, blank_signal;
```

```
////////////////// state parameters/////////////////
parameter testFF=0;
parameter test00a=1;
parameter test00b=2;
parameter word_test=3;


always @ (posedge clk) begin
if (reset) begin
      next<=testFF; // wait for the next SAV (start active video)
      at_even_field<=0; //random initialization
      end
else
case (next)


testFF: begin
            //to avoid reading FF as a very high output
            Pagedone<=0;
            counterx<=counterx+1;
            if (v_in[9:0]==10'b1111111111) begin
                        next<=test00a;
                        blank_signal<=1;
                        end
            else next<=testFF;
            end

test00a: begin
            counterx<=counterx+1;
            if (v_in[9:0]==10'b0000000000) next<=test00b;
            else // main else
            next<=testFF;
            end // end main else

test00b: begin
             counterx<=counterx+1;
             if (v_in[9:0]==10'b0000000000)next<=word_test;
             else next<=testFF;
             end
/////////// states to track the EAV and SAV of all possible different
lines////////
// F in data sheet corresponds to v_in[8] :: high F indicates even
field
// V in data sheet corresponds to v_in[7] :: high V indicates vertical
blanking lines
// H in data sheet corresponds to v_in[6] :: high H indicates EAV (end
active video)
// preamble tracker

word_test: begin
               counterx<=0;
               next<=testFF;
               // every vertical blank lines, and in between EAV and
SAV, blank signal=1
```

```
                  if (v_in[8] & v_in[7] & !v_in[6] & at_even_field)
      Pagedone<=1;

               // SAV of even_field from even field
                  else if (v_in[8] & !v_in[7] & !v_in[6] &
at_even_field) begin
                        countery<=countery+2;
                        blank_signal<=0;
                        at_even_field<=1;
                        end

               // SAV of even_field from odd field
               else if (v_in[8] & !v_in[7] & !v_in[6] & !at_even_field)
begin
                        blank_signal<=0;
                        countery<=2; //resetting countery back to 2
                        at_even_field<=1;
                        end

                  // SAV of odd_field from odd field
                else if (!v_in[8] & !v_in[7] & !v_in[6]
& !at_even_field) begin
                         countery<=countery+2;
                         blank_signal<=0;
                         at_even_field<=0;
                         end

                  // SAV of odd_field from even field
                  else if (!v_in[8] & !v_in[7] & !v_in[6] &
at_even_field) begin
                        blank_signal<=0;
                        countery<=1; //resetting counter 1 back to 1
                        at_even_field<=0; //indicating it is at odd
field
                        end


                  else if (v_in[7] | v_in[6]) blank_signal<=1;
                  end // end word test

endcase

end // end always

assign Y=v_in[9:2];
assign ledy=countery;
assign ledx=counterx[10:1];
assign Readen=((counterx==0) | blank_signal)? 1'b0: counterx[0];

endmodule


// Jin Hock Ong
// Filter takes in luminance value, and coordinate of the luminance
from decoder module
// Filter will have a threshold value that determines if the pixel
represents the LED or not
```

```
// LEDpos will go high if the pixel is an LED pixel and low otherwise
// Filter will be connected to coordinator module
module filter(clk, Readen, ledx_in, ledy_in, Y, ledx, ledy, LEDpos);

//input blank_field;
input clk;
input Readen;
input [8:0] ledy_in;
input [9:0] ledx_in; // coordinates for the sample being sent from the
decoder.v
input [7:0] Y; //8 bits from decoder : the values for the luminance

output [9:0] ledx;
output [8:0] ledy; // coordinates of the LED pixel
output LEDpos; // a high means the pixel


reg LEDpos;
reg [9:0] ledx;
reg [8:0] ledy;

// output will be sent to coordinator
// output is in the form of coordinates and LEDpos
// limit the movement to certain part of the screen only
// LEDpos is high if the pixel represents the LED and low if it does
not

parameter threshold=210;


always @ (posedge clk) begin
if ((threshold<=Y) & Readen)
      begin
      LEDpos<=1;
      ledx<=ledx_in;
      ledy<=ledy_in;
      end
else
      begin
      LEDpos<=0;
      ledx<=ledx; // retain the old coordinates
      ledy<=ledy;
      end
end // end main else

endmodule


// Jin Hock Ong
// Last modified on May 9th 6.50 pm
// Adjusted the adjacency test to detect pixels that represent the
identical LED

// coordinator determines which points correspond to the part of the
baseball bat
// incorporates Pagedone (from decoder) to avoid infinite loop if we
cannot find the three points
```

```
// The LED maybe represented by a few pixels and not just one
// Therefore, we should eliminate neighbouring pixels.
// Represent each point with one pixel only

module coordinator(clk, reset, LEDpos, ledx, ledy, Pagedone,
bottom_point, top_point1, top_point2, error,coordinate_done);

input clk, reset;
input LEDpos; // from filter module
input Pagedone;
// sent by decoder to signify that the page has finished decoding the
page: the whole screen
// if pagedone is high, then output the coordinate of the three points
and the speed of the swing can be calculated
input [9:0] ledx;
input [8:0] ledy;


output [18:0] bottom_point, top_point1, top_point2;  // combine x and y
coordinate together:: first 10 bits for x coordinate
output error; // to indicate that not LED pixels are found, or less
than 3 LED is found
output coordinate_done;

reg [18:0] bottom_point, top_point1, top_point2;
reg [1:0] next;
reg coordinate_done, error;


// remove points of the adjacent pixel: always take the
bottom,rightmost pixel for specific LED
// doing this allow us to  keep track of the adjacent points since we
are decoding from top to bottom and left to right

// pagedone :: for error handling:: in case the camera is not able to
detect the three necessary points
// after the decoder has finished decoding a page, coordinator will
start over thus the coordinates of the last point
// may not change

parameter firstpoint =0;
parameter secondpoint=1;
parameter thirdpoint =2;
parameter foundall   =3; //as a buffer until the next page

parameter adjac=25;

always @ (posedge clk) begin
      if (reset) begin
            next<=firstpoint;
            end

else
      begin

case (next)
```

```
firstpoint: begin //parameter secondpoint=1;
                  coordinate_done<=0;
                  if(Pagedone) begin
                        error<=1;
                        next<=firstpoint;
                        coordinate_done<=1;
                        end
                  if(LEDpos) begin // begin 1
                        bottom_point<={ledy, ledx};
                        next<=secondpoint;
                        end // end begin 1
                  else
                  next<=firstpoint;
                  end // end begin first point

secondpoint: begin // parameter secondpoint=1;
                   if (Pagedone) begin
                                          next<=firstpoint;  //if fail
to find second point before the completion of decoder
                                          error<=1;
                                          coordinate_done<=1;
                                       end
                   else if (!LEDpos) next<=secondpoint; //continue
waiting for second LEDpixel
                   //adjacency test
                   else if(
                           ((bottom_point[9:0]-ledx<=20)  |(ledx-
bottom_point[9:  0]<=adjac)) &
                           ((bottom_point[18:10]-ledy<=20)|(ledy-
bottom_point[18:10]<=adjac)))
                                          begin //if it is an adjacent
pixel: replace old point with new coordinate
                                                next<=secondpoint; //if
coordinates of the new found pixel is actually from the same LED,
continue to search for second pt
                                                if
(ledy>=bottom_point[18:10]) bottom_point<={ledy,ledx};
                                          end

                  // if not adjacent
                        else if (bottom_point[9:0]<=ledx) begin

top_point1<={ledy,ledx};
                                                 next<= thirdpoint;
                                          end
                        else if (bottom_point[9:0]>=ledx) begin

      top_point1<=bottom_point;

      bottom_point<={ledy,ledx};
                                                next<=thirdpoint;
                                                end // end else 3
                        else next<=secondpoint;
                        end // end second point begin

thirdpoint: begin
                   if (Pagedone) begin
```

```
                                          next<=firstpoint;
                                          error<=1;
                                          coordinate_done<=1;
                                          end
                   else if (!LEDpos) next<=thirdpoint; //continue
waiting for second LEDpixel
                   else if (
                              ((bottom_point[9:0]-ledx<=20)  | (ledx-
bottom_point[9:0]<=adjac)) &
                              ((ledy-bottom_point[18:10]<=20)|
(bottom_point[18:10]-ledy<=adjac))) //adjacency test with bottom_point
                                    begin
                                     next<=thirdpoint;       //
continue searching for the third point
                                     if (ledy>=bottom_point[18:10])
bottom_point<={ledy,ledx};
                                    end
                  else if (
                              ((top_point1[9:0]-ledx<=20)  | (ledx-
top_point1[9:0]<=adjac)) &
                    ((ledy-top_point1[18:10]<=20)| (top_point1[18:10]-
ledy<=adjac))) // adjacency test with top_point1
                                    begin
                                     next<=thirdpoint;       //
continue searching for the third point
                                     if (ledy>=top_point1[18:10])
top_point1<={ledy[8:0], ledx[9:0]};
                                    end
                        else if ((bottom_point[9:0]>=ledx) &
(top_point1[18:10]>=ledy)) begin
                                                top_point2<={ledy,ledx};
                                                next<= foundall;
                                          end
                        else if
((bottom_point[9:0]>=ledx)&(top_point1[18:10]<=ledy)) begin
                                               top_point2<=  top_point1;
                                                top_point1<={ledy,
ledx};

                                                next<=foundall;
                                                end
                        else
if((bottom_point[9:0]<=ledx)&(top_point1[18:10]>=bottom_point[18:10]))
begin
                                                bottom_point<={ledy,
ledx};

      top_point2<=bottom_point;
                                                next<=foundall;
                                                end
                        else if
((bottom_point[9:0]<=ledx)&(top_point1[18:10]<=bottom_point[18:10]))
begin
                                                bottom_point<={ledy,
ledy};

      top_point1<=bottom_point;
                                                top_point2<=top_point1;
                                                next<=foundall;
                                                end
                        else next<=thirdpoint;
                  end // end thirdpoint begin

// have to check adjacency test with all three points even though we
have all three points already
//:: note we are taking the bottom rightmost

foundall: begin
                  error<=0; //all three points have been found
                  if (Pagedone) begin next<=firstpoint;
coordinate_done<=1; end
            else if (!LEDpos) next<=foundall; //continue waiting for
second LEDpixel
              else if (
                              ((bottom_point[9:0]-ledx<=20)  | (ledx-
bottom_point[9:0]<=adjac)) &
                              ((ledy-bottom_point[18:10]<=20)|
(bottom_point[18:10]-ledy<=adjac))) // adjacency test with bottom pt
                                    begin
                                     next<=foundall;   // continue
searching for the third point
                                     if (ledy>=bottom_point[18:0])
bottom_point<={ledy,ledx}; //if adjacent with bottom pt, update the
coordinate
                                    end

                  else if (
                              ((top_point1[9:0]-ledx<=20)  | (ledx-
top_point1[9:0]<=adjac)) &
                    ((ledy-top_point1[18:10]<=20)| (top_point1[18:10]-
ledy<=adjac))) // adjacency test with top_point1
                                    begin
                                     next<=foundall;
                                     if (ledy>=top_point1[18:10])
top_point1<={ledy[8:0], ledx[9:0]};
                                    end

                  else if (
                              ((top_point2[9:0]-ledx<=20)  | (ledx-
top_point2[9:0]<=adjac)) &
                    ((ledy-top_point2[18:10]<=20)| (top_point2[18:10]-
ledy<=adjac))) // adjacency test with top_point2
                                    begin
                                     next<=foundall;
                                    if    (ledy>=top_point2[18:10])
top_point1<={ledy[8:0], ledx[9:0]};
                                    end
                  else next<=foundall;
                  end

endcase
end // end else
end // end always

endmodule
```

```
// Jin Hock Ong
// Register the inputs so that video encoder can get values


module register(clk,reset, error,coordinate_done, bottom_point,
top_pointup, top_pointdown, //input
                        bottom_point_out, top_point1_out,
top_point2_out); //output


input clk, reset;
input coordinate_done, error;
input [18:0] bottom_point, top_pointup, top_pointdown;

output [18:0] bottom_point_out, top_point1_out, top_point2_out;
reg    [18:0] bottom_point_out, top_point1_out, top_point2_out;


always @ (posedge clk) begin
if (reset) begin
//default points before game is triggered
      bottom_point_out<=19'b1011010000111110100;    //y-360d :: x-500d
      top_point1_out  <=19'b1010000000100101100;     //y-320d :: x-300d
      top_point2_out  <=19'b1100100000100101100;     //y-400d ::  x-
300d
      end
else if (coordinate_done & !error) begin //if not error: update new
points
            bottom_point_out<=bottom_point [18:0];
            top_point1_out  <=top_pointup  [18:0];
            top_point2_out  <=top_pointdown[18:0];
            end
else
      begin //if error:: retain the coordinate of old points and set
start_write<=0 since we do not need to update data in RAM
      bottom_point_out<=bottom_point_out;
      top_point1_out  <=top_point1_out;
      top_point2_out  <=top_point2_out;
      end
end//end always
endmodule
```

```
// Jin Hock Ong

module speedometer (clk, reset, coordinate_done, bottom_pt, top_pt,
sqrt_val, multdone, sqrtdone, sqrdx, sqrdy,  //input
                              startmult, startsqrt, xlen,ylen, sum,
speed, backswing); //output

input clk, reset, coordinate_done;
// start to be connected to coordinate done

// from register
input [18:0] bottom_pt, top_pt;

// from square module :: a multiplier
input [19:0] sqrdx;
input [17:0] sqrdy;
input multdone;

// from sqrt module :: square root module
input [19:0] sqrt_val;
input sqrtdone;

// to sqrt module
output [37:0] sum;
output startsqrt;

// to square module
output [9:0] xlen;
output [8:0] ylen;
output startmult;

// the main output
output [19:0] speed;
output backswing;


reg [19:0] speed;
reg [19:0] old_length;
reg backswing;
reg [9:0] xlen;
reg [8:0] ylen;
reg startmult;
reg [1:0] next;
reg [37:0] sum;
reg startsqrt;




parameter idle=0;
parameter substract=1;
parameter square=2;
parameter square_rt=3;

always @ (posedge clk) begin
```

```
if (reset) begin
      speed=0;
      old_length=0;
      next<=idle;
      end
else
begin

case(next)
idle: begin
            if (coordinate_done) next<=substract;
            else next<=idle;
         end
substract: begin
                  next<=square;
                  xlen<=bottom_pt[9:0]-top_pt[9:0];
                  startmult<=1;
                  if (bottom_pt[18:10]>=top_pt[18:10])
ylen<=bottom_pt[18:10]-top_pt[18:10];
                  else ylen<=top_pt[18:10]-bottom_pt[18:10];
               end
square: begin
                  startmult<=0;
            if(multdone) begin
                  sum<=sqrdx+sqrdy;
                  startsqrt<=1;
                  next<=square_rt;
                  end
            else next<=square;
            end // end square
square_rt: begin
                  if (sqrtdone) begin
                        old_length<=sqrt_val;
                        next<=idle;
                              if(sqrt_val>=old_length) begin
                                    backswing<=0;
                                    speed<=sqrt_val-old_length;
                                    end
                              else
                                    begin
                                    backswing<=1;
                                    speed<=old_length-sqrt_val;
                                    end
                              end
                  else next<=square_rt;
                        end // end square_rt;
endcase
end // end else
end // end always
endmodule
```

Video and Game Interface

```
module ball_track(clk, reset, vsync, pitch, pitch_sel, video_x, video_y, 
visable,
                     vga_red_out, vga_green_out, vga_blue_out, 
in_window, rom_address);

      // Define the colors for the ball.

      parameter BALL_RED = 240;
      parameter BALL_GREEN = 240;
      parameter BALL_BLUE = 240;

      parameter EDGE_RED = 5;
      parameter EDGE_GREEN = 5;
      parameter EDGE_BLUE = 5;

      parameter STITCH_RED = 240;
      parameter STITCH_GREEN = 10;
      parameter STITCH_BLUE = 10;


      input clk;
      input reset;
      input vsync;
      input pitch;
      input [1:0] pitch_sel;

      input[9:0] video_x;
      input[9:0] video_y;

      output in_window;
      output[12:0] rom_address;

      output visable;
      output[7:0] vga_red_out, vga_green_out, vga_blue_out;
      reg[7:0] vga_red_out, vga_green_out, vga_blue_out;


      wire[1:0] rom_data;
      reg[12:0] rom_address;
      reg[12:0] offset;
      reg visable;

      wire[3:0] ball_sel;
      wire[9:0] x_location;
      wire[9:0] y_location;
      wire update, update_size, in_window;



      always @ (in_window)
      begin
      // first visable check via box
      if (in_window)
            case (ball_sel)
                  4'h0:
                  begin
                        offset = 12'h0;
                  end
                  4'h1:
                  begin
                        offset = 12'h240;
                  end
                  4'h2:
                  begin
                        offset = 12'h480;
                  end
                  4'h3:
                  begin
                        offset = 12'h6C0;
                  end
                  4'h4:
                  begin
                        offset = 12'h900;
                  end
                  4'h5:
                  begin
                        offset = 12'hB40;
                  end
                  4'h6:
                  begin
                        offset = 12'hD80;
                  end
                  4'h7:
                  begin
                        offset = 12'hFC0;
                  end
                  4'h8:
                  begin
                        offset = 12'h1200;
                  end
                  4'h9:
                  begin
                        offset = 12'h1440;
                  end
                  default:
                  begin
                        offset = 12'h0;
                  end
            endcase
      end

      bb_rom rom (.addr(rom_address), .clk(clk), .dout(rom_data));

      divider div
(.clk(clk), .reset(reset), .vsync(vsync), .update(update), .update_size
(update_size),
                    .pitch_sel(pitch_sel));
      pitch_fsm
fsm(.clk(clk), .reset(reset), .pitch(pitch), .pitch_sel(pitch_sel),
          .ball_sel(ball_sel), .x(x_location), .y(y_location), .update(
update),
                        .update_size(update_size));

      ball_window window
(.pixel_count(video_x), .line_count(video_y), .clk(clk), .visable(in_wi
ndow),
                      .x(x_location), .y(y_location));

      always @ (posedge clk)
      begin
            if (vsync)
                  rom_address <= offset;
            else if (in_window)
                  rom_address <= rom_address + 1;
            else rom_address <= rom_address;
      end

      always @ (rom_address)
      begin
         case (rom_data)
            2'h0:
            begin
                  visable = 0;
                  vga_red_out = BALL_RED;
                  vga_blue_out = BALL_BLUE;
                  vga_green_out = BALL_GREEN;
            end
            2'h1:
            begin
                  visable = 1;
                  vga_red_out = BALL_RED;
                  vga_blue_out = BALL_BLUE;
                  vga_green_out = BALL_GREEN;
            end
            2'h2:
            begin
                  visable = 1;
                  vga_red_out = EDGE_RED;
                  vga_blue_out = EDGE_BLUE;
                  vga_green_out = EDGE_GREEN;
            end
            2'h3:
            begin
                  visable = 1;
                  vga_red_out = STITCH_RED;
                  vga_blue_out = STITCH_BLUE;
                  vga_green_out = STITCH_GREEN;
            end
            default:
            begin
                  visable = 0;
                  vga_red_out = BALL_RED;
                  vga_blue_out = BALL_BLUE;
                  vga_green_out = BALL_GREEN;
            end
         endcase
      end
endmodule


// This module is used to determine if a point and some border is
visable.  Currently
// generates a square around the point.

module ball_window(pixel_count, line_count, clk, visable, x, y);

input [9:0] line_count, pixel_count, x, y;
input clk;

output visable;

parameter BORDER = 25;  // The number of pixels to color around a x, y
point absolute greater than so real
                            // border is 24.
reg visable;

always @ (posedge clk)
begin
   visable <= 0;  // default not visable.
   // this is only visible if the current count is in the box defined
by the x, y point.
   if (line_count > y)
    if (line_count < (y + BORDER))
         if (pixel_count > x)
            if (pixel_count < (x + BORDER))
               visable <= 1;

end
endmodule

module bat_logic(clk, reset, x1, y1, move, m_1, m_2, m_3);

input reset;
//input [19:0] p1, p2, p3;
input [9:0] x1;
input [9:0] y1;
input clk;

output [3:0] move;  // bit encoded movement left, right, up, down

output [3:0] m_1, m_2, m_3;  // bit encoded movement left, right, up,
down

parameter LEFT_THRESHOLD = 270;
parameter RIGHT_THRESHOLD = 290;                 // noise problem
parameter UP_THRESHOLD = 210;          /// good thresholds
parameter DOWN_THRESHOLD = 230;

reg [3:0] move, m_1, m_2, m_3;  // require signal high on three clock
cycles.
```

```
wire[9:0] x;
wire[9:0] y;
assign x = x1; // p1[9:0];
assign y = y1[8:0]; //p1[18:10];

always @ (reset or x or y)// p1 or p2 or p3)
begin
      // default.
      m_3 = 4'b0;
      if (reset)
            m_3 = 4'b0;
      else
      begin
            if (x < LEFT_THRESHOLD)
              m_3[0] = 1'b1;
            if (x > RIGHT_THRESHOLD)
              m_3[1] = 1'b1;
            if (y < UP_THRESHOLD)
              m_3[2] = 1'b1;
            if (y > DOWN_THRESHOLD)
              m_3[3] = 1'b1;

      end
end

always @ (posedge clk)
begin
      m_2 <= m_3;
      m_1 <= m_2;
      move <= m_1 & m_2 & m_3;
//    move <= 4'b1010;
end

endmodule


// This module is used to generate the bat tracking pixels.  This is
assumed to be
// ran inside of the vga module, thus the reset and count signals are
synchornized to
// the pixel clock.
module bat_track( clk, reset, pixel_count, line_count, vga_out_red,
vga_out_green, vga_out_blue,
                    p1, update, visible, move, bd);

   parameter BAT_RED = 169;
   parameter BAT_GREEN = 126;
   parameter BAT_BLUE = 54;

   parameter BAT_HEIGHT = 15;
   parameter BAT_WIDTH = 80;

   parameter MIN_BAT_X = 5;
   parameter MAX_BAT_X = 715;

   parameter MIN_BAT_Y = 300;
   parameter MAX_BAT_Y = 580;
   parameter DEFAULT_Y = 400;
   parameter DEFAULT_X = 300;

   input reset; // Active high reset, synchronous with pixel clock
   input clk; // pixel clock
   input update;  // active high load enable
   input [10:0] pixel_count; // Current pixel in the line
   input [10:0] line_count;  // Current lines in each frame
   input [19:0] p1;  // location of three points representing the bat.
   output visible;
   reg visible;
   output [7:0] vga_out_red, vga_out_green, vga_out_blue; // Outputs to
DAC


   // debug

   output [3:0] move;
   output bd;

   // values
   assign vga_out_red = BAT_RED;
   assign vga_out_green = BAT_GREEN;
   assign vga_out_blue = BAT_BLUE;

   reg[10:0] batx, baty;
   reg bd;

   // move logic and enables
   wire [3:0] move;  // bit encoded movement left, right, up, down
   bat_logic logic
(.clk(clk), .reset(reset), .x1(p1[9:0]), .y1({1'b0,p1[18:10]}),
                        .move(move));

   wire move_bat;
   assign move_bat = update && !bd;

   always @ (posedge clk)
   begin
   // defaults
     batx <= batx;
     baty <= baty;
     bd <= bd;

      if (reset)
      begin
         batx <= DEFAULT_X;
         baty <= DEFAULT_Y;
         bd <= 1'b0;
      end else if (move_bat)
      begin
            if (move[0] && (batx > MIN_BAT_X))
                  batx <= batx - 1;  // left
            if (move[1] && (batx < MAX_BAT_X))
                  batx <= batx + 1;  // right
            if (move[2] && (baty > MIN_BAT_Y))
```

```
                  baty <= baty - 1;  // up
            if (move[3] && (baty < MAX_BAT_Y))
                  baty <= baty + 1;  // down
            bd <= 1'b1;
      end
      else if (!update)
            bd <= 1'b0;
   end

   always @ (pixel_count or line_count)
      begin
     visible = 0;  // default not visable.
         // this is only visible if the current count is in the box
defined by the x, y point.
      if (line_count > (baty))
            if (line_count < (baty + BAT_HEIGHT))
                  if (pixel_count > (batx))
                        if (pixel_count < (batx + BAT_WIDTH))
                              visible = 1;
      end
endmodule

module divider(clk, reset, vsync, update, update_size, pitch_sel);

parameter FAST_PITCH_WAIT = 7;
parameter CURVE_PITCH_WAIT = 11;
parameter KNUCKLE_PITCH_WAIT = 19;

parameter WAIT_HIGH = 0;
parameter WAIT_LOW = 1;
parameter WAIT_HIGH_2 = 2;
parameter WAIT_LOW_2 = 3;

parameter FAST_BALL = 0;
parameter CURVE_BALL_LEFT = 1;
parameter CURVE_BALL_RIGHT = 2;
parameter KNUCKLE_BALL = 3;


  // System Clk
input clk;

// Reset signal, assumed to be synchronized

input[1:0] pitch_sel;

input reset;

input vsync;
// update the pitch
output update;
output update_size;

reg update;
reg update_size;

reg[1:0] state, next;
```

```
reg[6:0] count;
reg count_en;

reg [4:0] timing;

always @ (pitch_sel)
begin
      case  (pitch_sel)
            FAST_BALL: begin timing <= FAST_PITCH_WAIT; end
            CURVE_BALL_LEFT: begin timing <= CURVE_PITCH_WAIT; end
            CURVE_BALL_RIGHT: begin timing <= CURVE_PITCH_WAIT; end
            KNUCKLE_BALL:     begin timing <= KNUCKLE_PITCH_WAIT; end
      endcase
end
always @ (posedge clk)
begin
      if (reset)
      begin
            count <= 0;
            state <= WAIT_HIGH;
      end
      else
      begin
            state <= next;
            if (count_en)
                  count <= count + 1;
            else
                  if (update || update_size)
                        count <= 0;
                  else
                        count <= count;
      end
end

always @ (vsync or state)
begin
      // defaults
      update = 0;
      update_size = 0;
      count_en = 0;
      case (state)
            WAIT_HIGH:
            begin
                  if (vsync)
                  begin
                        next = WAIT_LOW;
                        if (count == timing)
                              update = 1;
                        else
                              count_en = 1;
                  end
                  else
                        next = WAIT_HIGH;
            end

            WAIT_LOW:
            begin
```

```
                  if (!vsync)
                        next = WAIT_HIGH_2;
                  else
                        next = WAIT_LOW;
            end
            WAIT_HIGH_2:
            begin
                  if (vsync)
                  begin
                        next = WAIT_LOW_2;
                        if (count == timing)
                              update_size = 1;
                        else
                              count_en = 1;
                  end
                  else
                        next = WAIT_HIGH_2;
            end
            WAIT_LOW_2:
            begin
                  if (!vsync)
                        next = WAIT_HIGH;
                  else
                        next = WAIT_LOW_2;
            end
      endcase

end

endmodule

// This is a memory coordination module that integrates the game state
between the two
// different timing relms, namely the 27 MHz video input and game logic
calculations and
// the 49.5 MHz pixel clock and image generation code.
module mem_cord(clk_27, w_address, w_value, pixel_clk, r_data,
r_address);


      game_state state
(.addra(w_address), .addrb(r_address), .clka(clk_27), .clkb(pixel_clk),
module game_state (
      addra,
      addrb,
      clka,
      clkb,
      dina,
      doutb,
      wea);


endmodule

// This module is used to generate the bat tracking pixels.  This is
assumed to be
```

```
// ran inside of the vga module, thus the reset and count signals are
synchornized to
// the pixel clock.
module pic_gen(reset, pixel_count, line_count, clk, vga_out_red,
vga_out_green, vga_out_blue,
                   p1, p2, p3, update, pitch, pitch_sel, v4, in_window,
rom_address, move, bat_done);

   parameter SKY_RED = 94;
   parameter SKY_GREEN = 152;
   parameter SKY_BLUE = 195;

   parameter GROUND_RED = 25;
   parameter GROUND_GREEN = 188;
   parameter GROUND_BLUE = 20;

   parameter HORIZON = 500;

   parameter BAT_RED = 240;
   parameter BAT_GREEN = 240;
   parameter BAT_BLUE = 240;


   input reset; // Active high reset, synchronous with pixel clock
   input clk; // pixel clock
   input update;  // active high load enable

   input [10:0] pixel_count; // Current pixel in the line
   input [10:0] line_count;  // Current lines in each frame

   input [19:0] p1, p2, p3;  // location of three points representing
the bat.

   input pitch;
   input [1:0] pitch_sel;

   output [7:0] vga_out_red, vga_out_green, vga_out_blue; // Outputs to
DAC
   reg [7:0] vga_out_red, vga_out_blue, vga_out_green;
   wire[7:0] ball_red, ball_blue, ball_green;
   wire[7:0] bat_red, bat_blue, bat_green;

   // debug

   output v4;
      output in_window;
      output[12:0] rom_address;
   output [3:0] move;
   output bat_done;

   reg [19:0] point1, point2, point3;  // load enabled registers.



   // define three point tracks
   wire v1, v2, v3, v4, v5;
   point_track pt1 (.pixel_count(pixel_count), .line_count(line_count),
         .clk(clk), .visable(v1), .x(point1[9:0]), .y({1'b0,
point1[18:10]}));
   point_track pt2 (.pixel_count(pixel_count), .line_count(line_count),

         .clk(clk), .visable(v2), .x(point2[9:0]), .y({1'b0,
point2[18:10]}));
   point_track pt3 (.pixel_count(pixel_count), .line_count(line_count),

         .clk(clk), .visable(v3), .x(point3[9:0]), .y({1'b0,
point3[18:10]}));

   ball_track
bt(.clk(clk), .reset(reset), .vsync(update), .pitch(pitch), .pitch_sel(
pitch_sel),

          .video_x(pixel_count), .video_y(line_count), .visable(v4),

          .vga_red_out(ball_red), .vga_green_out(ball_green), .vga_blue
_out(ball_blue),
                .in_window(in_window), .rom_address(rom_address));

  bat_track
bat_t(.clk(clk), .reset(reset), .pixel_count(pixel_count), .line_count(
line_count),

              .vga_out_red(bat_red), .vga_out_green(bat_green), .vga_ou
t_blue(bat_blue), .p1(point1),

         .update(update), .visible(v5), .move(move), .bd(bat_done));

   always @ (posedge clk)
     begin
            if (update)
            begin
                  point1 <= p1;
                  point2 <= p2;
                  point3 <= p3;
            end
            else
            begin
                  point1 <= point1;
                  point2 <= point2;
                  point3 <= point3;
            end

            if (v1 || v2 || v3)
            begin
                  vga_out_red <= BAT_RED;
                  vga_out_blue <= BAT_BLUE;
                  vga_out_green <= BAT_GREEN;
            end
            else
            if (v5)
            begin
                  vga_out_red <= bat_red;
                  vga_out_blue <= bat_blue;
```

```
                        vga_out_green <= bat_green;
                end
                else
                if (v4)
                begin
                        vga_out_red <= ball_red;
                        vga_out_blue <= ball_blue;
                        vga_out_green <= ball_green;
                end
                else
                if (line_count > HORIZON)
                begin
                        vga_out_red <= GROUND_RED;
                        vga_out_blue <= GROUND_BLUE;
                        vga_out_green <= GROUND_GREEN;
                end
                else
                begin
                        vga_out_red <= SKY_RED;
                        vga_out_blue <= SKY_BLUE;
                        vga_out_green <= SKY_GREEN;
                end
        end

endmodule

module pitch_fsm(clk, reset, pitch, pitch_sel, ball_sel, x, y, update,
update_size);
// This code is a simple finate state machine that controls the pitches
for the baseball game
// it uses a major / minor paradiam so it signals finished at the end
of its conversion

// System Clk
input clk;

// Reset signal, assumed to be synchronized

input reset;

// update the pitch
input update;
input update_size;

// FSM Interface

//input start;  // signals that the data is availble
//output busy;    // signals that the conversion is complete.  This
signal can glitch.

// pitch Interface
input pitch;   // control signal to start a pitch
input [1:0] pitch_sel; // control signal for the type of pitch to use

output[3:0] ball_sel;    // signals the size of the baseball
output[9:0] x;    // signals that x location of the baseball for the
screen, point 00 of image.
```

```
output[9:0] y;    // signals that y location of the baseball for the
screen, point 00 of image.


reg[3:0] ball_sel;    // signals the size of the baseball
reg[9:0] x;    // signals that x location of the baseball for the
screen, point 00 of image.
reg[9:0] y;    // signals that y location of the baseball for the
screen, point 00 of image.

reg[3:0] ball_sel_int;    // signals the size of the baseball
reg[9:0] x_int;    // signals that x location of the baseball for the
screen, point 00 of image.
reg[9:0] y_int;    // signals that y location of the baseball for the
screen, point 00 of image.

// internal state
reg[5:0] state, nextstate;


parameter FAST_BALL = 0;
parameter CURVE_BALL_LEFT = 1;
parameter CURVE_BALL_RIGHT = 2;
parameter KNUCKLE_BALL = 3;

// state declarations
      parameter IDLE = 0;
      // fast ball states
      parameter FB_1 = 1;
      parameter FB_2 = 2;
      parameter FB_3 = 3;
      parameter FB_4 = 4;
      parameter FB_5 = 5;
      parameter FB_6 = 6;
      parameter FB_7 = 7;
      parameter FB_8 = 8;
      parameter FB_9 = 9;
      parameter FB_10 = 10;

      // curve ball left states
      parameter CBL_1 = 11;
      parameter CBL_2 = 12;
      parameter CBL_3 = 13;
      parameter CBL_4 = 14;
      parameter CBL_5 = 15;
      parameter CBL_6 = 16;
      parameter CBL_7 = 17;
      parameter CBL_8 = 18;
      parameter CBL_9 = 19;
      parameter CBL_10 = 20;

      // knuckle ball states
      parameter KB_1 = 21;
      parameter KB_2 = 22;
      parameter KB_3 = 23;
      parameter KB_4 = 24;
      parameter KB_5 = 25;
```

```
	parameter KB_6 = 26;
	parameter KB_7 = 27;
	parameter KB_8 = 28;
	parameter KB_9 = 29;
	parameter KB_10 = 30;


	// curve ball right states
	parameter CBR_1 = 31;
	parameter CBR_2 = 32;
	parameter CBR_3 = 33;
	parameter CBR_4 = 34;
	parameter CBR_5 = 35;
	parameter CBR_6 = 36;
	parameter CBR_7 = 37;
	parameter CBR_8 = 38;
	parameter CBR_9 = 39;
	parameter CBR_10 = 40;


always @ (posedge clk)
	begin
		// register the Combination logic control signals
		ball_sel <= ball_sel_int;
		x <= x_int;
		y <= y_int;
		// update the state.
		if (reset) state <= IDLE;
			else state <= nextstate;
	end


always @ (state or pitch or update or update_size or pitch_sel)
	begin
		ball_sel_int = 4'h9;    // signals the size of the baseball
		x_int = 10'h320;    // off screen
		y_int = 10'h258;    // off screen

		case (state)
			IDLE: begin
				if (pitch)
					case (pitch_sel)
						FAST_BALL: begin nextstate = FB_1;
end
						CURVE_BALL_LEFT: begin nextstate =
CBL_1; end
						CURVE_BALL_RIGHT: begin nextstate =
CBR_1; end
						KNUCKLE_BALL: begin nextstate =
KB_1; end
						default: begin nextstate = FB_1;
end
					endcase
				else
					nextstate = IDLE;
			end
			FB_1: begin
```

```
                              if (update_size)
                                    nextstate = FB_4;
                              else if (update)
                                    nextstate = FB_2;
                              else nextstate = FB_1;
                              ball_sel_int = 4'h0;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h120;
                        end
                        FB_2: begin
                              if (update_size)
                                    nextstate = FB_3;
                              else if (update)
                                    nextstate = FB_1;
                              else nextstate = FB_2;
                              ball_sel_int = 4'h1;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h120;
                        end
                        FB_3: begin
                              if (update_size)
                                    nextstate = FB_6;
                              else if (update)
                                    nextstate = FB_4;
                              else nextstate = FB_3;
                              ball_sel_int = 4'h2;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h12a;
                        end
                        FB_4: begin
                              if (update_size)
                                    nextstate = FB_5;
                              else if (update)
                                    nextstate = FB_3;
                              else nextstate = FB_4;
                              ball_sel_int = 4'h3;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h12a;
                        end
                        FB_5: begin
                              if (update_size)
                                    nextstate = FB_8;
                              else if (update)
                                    nextstate = FB_6;
                              else nextstate = FB_5;
                              ball_sel_int = 4'h4;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h13e;
                        end
                        FB_6: begin
                              if (update_size)
                                    nextstate = FB_7;
                              else if (update)
                                    nextstate = FB_5;
                              else nextstate = FB_6;
                              ball_sel_int = 4'h5;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h13e;
                        end
                        FB_7: begin
                              if (update_size)
                                    nextstate = FB_10;
                              else if (update)
                                    nextstate = FB_8;
                              else nextstate = FB_7;
                              ball_sel_int = 4'h6;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h148;
                        end
                        FB_8: begin
                              if (update_size)
                                    nextstate = FB_9;
                              else if (update)
                                    nextstate = FB_7;
                              else nextstate = FB_8;
                              ball_sel_int = 4'h7;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h148;
                        end
                        FB_9: begin
                              if (update_size)
                                    nextstate = IDLE;
                              else if (update)
                                    nextstate = FB_10;
                              else nextstate = FB_9;
                              ball_sel_int = 4'h8;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h152;
                        end
                        FB_10: begin
                              if (update_size)
                                    nextstate = IDLE;
                              else if (update)
                                    nextstate = FB_9;
                              else nextstate = FB_10;
                              ball_sel_int = 4'h9;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h152;
                        end
                        CBL_1: begin
                              if (update_size)
                                    nextstate = CBL_4;
                              else if (update)
                                    nextstate = CBL_2;
                              else nextstate = CBL_1;
                              ball_sel_int = 4'h0;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h120;
                        end
                        CBL_2: begin
                              if (update_size)
                                    nextstate = CBL_3;
                              else if (update)
                                    nextstate = CBL_1;
                              else nextstate = CBL_2;
                              ball_sel_int = 4'h1;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h120;
                        end
                        CBL_3: begin
                              if (update_size)
                                    nextstate = CBL_6;
                              else if (update)
                                    nextstate = CBL_4;
                              else nextstate = CBL_3;
                              ball_sel_int = 4'h2;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h12a;
                        end
                        CBL_4: begin
                              if (update_size)
                                    nextstate = CBL_5;
                              else if (update)
                                    nextstate = CBL_3;
                              else nextstate = CBL_4;
                              ball_sel_int = 4'h3;    // signals the size of
      the baseball
                              x_int = 10'h184;
                              y_int = 10'h12a;
                        end
                        CBL_5: begin
                              if (update_size)
                                    nextstate = CBL_8;
                              else if (update)
                                    nextstate = CBL_6;
                              else nextstate = CBL_5;
                              ball_sel_int = 4'h4;    // signals the size of
      the baseball
                              x_int = 10'h164;
                              y_int = 10'h13e;
                        end
                        CBL_6: begin
                              if (update_size)
                                    nextstate = CBL_7;
                              else if (update)
                                    nextstate = CBL_5;
                              else nextstate = CBL_6;
                              ball_sel_int = 4'h5;    // signals the size of
      the baseball
                              x_int = 10'h164;
                              y_int = 10'h13e;
                        end
                        CBL_7: begin
                              if (update_size)
                                    nextstate = CBL_10;
                              else if (update)
                                    nextstate = CBL_8;
                              else nextstate = CBL_7;
                              ball_sel_int = 4'h6;    // signals the size of
      the baseball
                              x_int = 10'h124;
                              y_int = 10'h148;
                        end
                        CBL_8: begin
                              if (update_size)
                                    nextstate = CBL_9;
                              else if (update)
                                    nextstate = CBL_7;
                              else nextstate = CBL_8;
                              ball_sel_int = 4'h7;    // signals the size of
      the baseball
                              x_int = 10'h124;
                              y_int = 10'h148;
                        end
                        CBL_9: begin
                              if (update_size)
                                    nextstate = IDLE;
                              else if (update)
                                    nextstate = CBL_10;
                              else nextstate = CBL_9;
                              ball_sel_int = 4'h8;    // signals the size of
      the baseball
                              x_int = 10'h0F4;
                              y_int = 10'h152;
                        end
                        CBL_10: begin
                              if (update_size)
                                    nextstate = IDLE;
                              else if (update)
                                    nextstate = CBL_9;
                              else nextstate = CBL_10;
                              ball_sel_int = 4'h9;    // signals the size of
      the baseball
                              x_int = 10'h0F4;
                              y_int = 10'h152;
                        end
                        CBR_1: begin
                              if (update_size)
                                    nextstate = CBR_4;
                              else if (update)
                                    nextstate = CBR_2;
                              else nextstate = CBR_1;
                              ball_sel_int = 4'h0;    // signals the size of
      the baseball
```

```
                  x_int = 10'h184;
                  y_int = 10'h120;
            end
            CBR_2: begin
                  if (update_size)
                        nextstate = CBR_3;
                  else if (update)
                        nextstate = CBR_1;
                  else nextstate = CBR_2;
                  ball_sel_int = 4'h1;    // signals the size of
the baseball
                  x_int = 10'h184;
                  y_int = 10'h120;
            end
            CBR_3: begin
                  if (update_size)
                        nextstate = CBR_6;
                  else if (update)
                        nextstate = CBR_4;
                  else nextstate = CBR_3;
                  ball_sel_int = 4'h2;    // signals the size of
the baseball
                  x_int = 10'h184;
                  y_int = 10'h12a;
            end
            CBR_4: begin
                  if (update_size)
                        nextstate = CBR_5;
                  else if (update)
                        nextstate = CBR_3;
                  else nextstate = CBR_4;
                  ball_sel_int = 4'h3;    // signals the size of
the baseball
                  x_int = 10'h184;
                  y_int = 10'h12a;
            end
            CBR_5: begin
                  if (update_size)
                        nextstate = CBR_8;
                  else if (update)
                        nextstate = CBR_6;
                  else nextstate = CBR_5;
                  ball_sel_int = 4'h4;    // signals the size of
the baseball
                  x_int = 10'h1A4;
                  y_int = 10'h13e;
            end
            CBR_6: begin
                  if (update_size)
                        nextstate = CBR_7;
                  else if (update)
                        nextstate = CBR_5;
                  else nextstate = CBR_6;
                  ball_sel_int = 4'h5;    // signals the size of
the baseball
                  x_int = 10'h1A4;
                  y_int = 10'h13e;
                  end
                  CBR_7: begin
                        if (update_size)
                              nextstate = CBR_10;
                        else if (update)
                              nextstate = CBR_8;
                        else nextstate = CBR_7;
                        ball_sel_int = 4'h6;    // signals the size of
the baseball
                        x_int = 10'h1C4;
                        y_int = 10'h148;
                  end
                  CBR_8: begin
                        if (update_size)
                              nextstate = CBR_9;
                        else if (update)
                              nextstate = CBR_7;
                        else nextstate = CBR_8;
                        ball_sel_int = 4'h7;    // signals the size of
the baseball
                        x_int = 10'h1C4;
                        y_int = 10'h148;
                  end
                  CBR_9: begin
                        if (update_size)
                              nextstate = IDLE;
                        else if (update)
                              nextstate = CBR_10;
                        else nextstate = CBR_9;
                        ball_sel_int = 4'h8;    // signals the size of
the baseball
                        x_int = 10'h1F4;
                        y_int = 10'h152;
                  end
                  CBR_10: begin
                        if (update_size)
                              nextstate = IDLE;
                        else if (update)
                              nextstate = CBR_9;
                        else nextstate = CBR_10;
                        ball_sel_int = 4'h9;    // signals the size of
the baseball
                        x_int = 10'h1F4;
                        y_int = 10'h152;
                  end
                  KB_1: begin
                        if (update_size)
                              nextstate = KB_4;
                        else if (update)
                              nextstate = KB_2;
                        else nextstate = KB_1;
                        ball_sel_int = 4'h0;    // signals the size of
the baseball
                        x_int = 10'h184;
                        y_int = 10'h120;
                  end
                  KB_2: begin
```

```
                        if (update_size)
                              nextstate = KB_3;
                        else if (update)
                              nextstate = KB_1;
                        else nextstate = KB_2;
                        ball_sel_int = 4'h1;    // signals the size of
the baseball
                        x_int = 10'h184;
                        y_int = 10'h120;
                  end
                  KB_3: begin
                        if (update_size)
                              nextstate = KB_6;
                        else if (update)
                              nextstate = KB_4;
                        else nextstate = KB_3;
                        ball_sel_int = 4'h2;    // signals the size of
the baseball
                        x_int = 10'h174;
                        y_int = 10'h13a;
                  end
                  KB_4: begin
                        if (update_size)
                              nextstate = KB_5;
                        else if (update)
                              nextstate = KB_3;
                        else nextstate = KB_4;
                        ball_sel_int = 4'h3;    // signals the size of
the baseball
                        x_int = 10'h174;
                        y_int = 10'h13a;
                  end
                  KB_5: begin
                        if (update_size)
                              nextstate = KB_8;
                        else if (update)
                              nextstate = KB_6;
                        else nextstate = KB_5;
                        ball_sel_int = 4'h4;    // signals the size of
the baseball
                        x_int = 10'h184;
                        y_int = 10'h13A;
                  end
                  KB_6: begin
                        if (update_size)
                              nextstate = KB_7;
                        else if (update)
                              nextstate = KB_5;
                        else nextstate = KB_6;
                        ball_sel_int = 4'h5;    // signals the size of
the baseball
                        x_int = 10'h184;
                        y_int = 10'h13A;
                  end
                  KB_7: begin
                        if (update_size)
                              nextstate = KB_10;
                        else if (update)
                              nextstate = KB_8;
                        else nextstate = KB_7;
                        ball_sel_int = 4'h6;    // signals the size of
the baseball
                        x_int = 10'h19A;
                        y_int = 10'h158;
                  end
                  KB_8: begin
                        if (update_size)
                              nextstate = KB_9;
                        else if (update)
                              nextstate = KB_7;
                        else nextstate = KB_8;
                        ball_sel_int = 4'h7;    // signals the size of
the baseball
                        x_int = 10'h19A;
                        y_int = 10'h158;
                  end
                  KB_9: begin
                        if (update_size)
                              nextstate = IDLE;
                        else if (update)
                              nextstate = KB_10;
                        else nextstate = KB_9;
                        ball_sel_int = 4'h8;    // signals the size of
the baseball
                        x_int = 10'h190;
                        y_int = 10'h142;
                  end
                  KB_10: begin
                        if (update_size)
                              nextstate = IDLE;
                        else if (update)
                              nextstate = KB_9;
                        else nextstate = KB_10;
                        ball_sel_int = 4'h9;    // signals the size of
the baseball
                        x_int = 10'h190;
                        y_int = 10'h142;
                  end


                  default: nextstate = IDLE;
            endcase // case state
      end // aways @ (state or status_d2 or sample)


endmodule


// This module is used to determine if a point and some border is
visable.  Currently
// generates a square around the point.

module point_track(pixel_count, line_count, clk, visable, x, y);
```

```
input [10:0] line_count, pixel_count, x, y;
input clk;

output visable;

parameter BORDER = 3;  // The number of pixels to color around a x, y
point;
parameter X_RES = 800; // The maximum number of pixels in a line
parameter Y_RES = 600; // The maximum number of lines.

reg visable;

always @ (posedge clk)
begin
   visable <= 0;  // default not visable.
   // this is only visible if the current count is in the box defined
by the x, y point.

   if (line_count > (y - BORDER))
    if (line_count < (y + BORDER))
         if (pixel_count > (x - BORDER))
            if (pixel_count < (x + BORDER))
               visable <= 1;

end



endmodule

module synchronizer(bat1,bat2,bat3,pitch,bat1_s,bat2_s,bat3_s,pitch_s,
clk);
// contains a number of signal synchronizers.
    input [19:0] bat1;
    input [19:0] bat2;
    input [19:0] bat3;
    input pitch;
    input clk;
    output [19:0] bat1_s;
    output [19:0] bat2_s;
    output [19:0] bat3_s;
    output [19:0] pitch_s;

    reg [19:0] bat1_s;
    reg [19:0] bat2_s;
    reg [19:0] bat3_s;
    reg pitch_s;

      // internals
    reg [19:0] bat1_d1, bat1_d2, bat2_d1, bat2_d2, bat3_d1, bat3_d2,
pitch_d1, pitch_d2;


always @ (posedge clk)
begin
      bat1_d1 <= bat1;
      bat2_d1 <= bat2;
      bat3_d1 <= bat3;
      pitch_d1 <= pitch;

      bat1_d2 <= bat1_d1;
      bat2_d2 <= bat2_d1;
      bat3_d2 <= bat3_d1;
      pitch_d2 <= pitch_d1;

      bat1_s <= bat1_d2;
      bat2_s <= bat2_d2;
      bat3_s <= bat3_d2;
      pitch_s <= pitch_d2;

end

endmodule

// VGA module taken from Nate's test vga output.  modified for
resolution
// and image generation.

module vga (reset, clock_27mhz, vga_out_red, vga_out_green,
vga_out_blue,
          vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
          vga_out_hsync, vga_out_vsync, pixel_clock, bat1, bat2, bat3,
pitch, pitch_sel, ball_vis,
          in_window, rom_address, move, bd, pixel_reset);

   input reset; // Active high reset, synchronous with 27MHz clock
   input clock_27mhz; // 27MHz input clock

   output [7:0] vga_out_red, vga_out_green, vga_out_blue; // Outputs to
DAC
   output vga_out_sync_b, vga_out_blank_b; // Composite sync/blank
outputs to DAC
   output vga_out_pixel_clock; // Pixel clock for DAC
   output vga_out_hsync, vga_out_vsync; // Sync outputs to VGA
connector

   // main "memory" interface
   input [19:0] bat1, bat2, bat3;
   input pitch;
   input[1:0] pitch_sel;
   output ball_vis;

      output in_window;
      output[12:0] rom_address;
      output[3:0] move;
      output bd; // bat done
      output pixel_reset;


   output pixel_clock;

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

```
   //
   // Internal signals
   //

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

   reg prst, pixel_reset; // Active high reset, synchronous with pixel
clock
   wire pixel_clock;  // 49.5 MHz pixel clock, different from
vga_output_clk CLF

//   reg [7:0] vga_out_red, vga_out_blue, vga_out_green;
   wire vga_out_sync_b, vga_out_blank_b, vga_out_hsync, vga_out_vsync;

   wire [10:0] pixel_count; // Counts pixels in each line
   wire [10:0] line_count; // Counts lines in each frame


//////////////////////////////////////////////////////////////////////
/////
   //
   // Generate the pixel clock (49.5 MHz)  == 27 * 11 / 6
   // and bring the reset signal into sync with the pixel clock
   //

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

   // synthesis attribute period of clock_27mhz is 37ns;

   DCM vga_dcm (.CLKIN(clock_27mhz),
                  .RST(1'b0),
                  .CLKFX(pixel_clock));
   // synthesis attribute DLL_FREQUENCY_MODE of vga_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of vga_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of vga_dcm is "TRUE"
   // synthesis attribute DFS_FREQUENCY_MODE of vga_dcm is "LOW"
   // synthesis attribute CLKFX_DIVIDE of vga_dcm is 6
   // synthesis attribute CLKFX_MULTIPLY of vga_dcm is 11
   // synthesis attribute CLK_FEEDBACK of vga_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of vga_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of vga_dcm is 0
   // synthesis attribute clkin_period of vga_dcm is "37.04ns"

   always @(posedge pixel_clock)
     begin
            prst <= reset;
            pixel_reset <= prst;
     end


//////////////////////////////////////////////////////////////////////
/////
   //
   // wire the sync and count module
   //
```

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

   vga_sync sync_gen (.reset(pixel_reset), .clk(pixel_clock),

        .vga_out_sync_b(vga_out_sync_b), .vga_out_blank_b(vga_out_blank
_b),
        .vga_out_pixel_clock(vga_out_pixel_clock), .vga_out_hsync(vga_o
ut_hsync),
        .vga_out_vsync(vga_out_vsync), .pixel_count(pixel_count), .line
_count(line_count));


///////////////////////////////////////////////////////////////////////
/////
   //
   // Generate cache refresh signals based on vsync.
   //

///////////////////////////////////////////////////////////////////////
/////
   wire [19:0] bat1_s, bat2_s, bat3_s, pitch_s;
   //wire [19:0] bs1, bs2, bs3, bs4;
   //assign bs1 = {13'h0, 9'h12C, 10'h1C0};
   //assign bs2 = {13'h0, 9'h12C, 10'h1A0};
   //assign bs3 = {13'h0, 9'h11C, 10'h190};
   //assign bs4 = {13'h0, 9'h13C, 10'h170};


   synchronizer sync
(.bat1(bat1), .bat2(bat2), .bat3(bat3), .pitch(pitch),

       .bat1_s(bat1_s), .bat2_s(bat2_s), .bat3_s(bat3_s), .pitch_s(pitc
h_s), .clk(pixel_clock));

   wire cache_count_enable;
   assign cache_count_enable = ~vga_out_vsync;

//   video_ca_addr addr
(.clk(pixel_clock), .reset(pixel_reset), .enc(cache_count_enable),
//
         .count(address), .p1(p1), .p2(p2), .p3(p3), .data(mm_data));
//   video_reg   v_reg
(.clk(pixel_clock), .refresh(), .addr(read_address), .data(c_data));

///////////////////////////////////////////////////////////////////////
/////
   //
   // Generate a pretty picture
   //   swap the bottom and top point

///////////////////////////////////////////////////////////////////////
/////
   pic_gen picture
(.reset(pixel_reset), .clk(pixel_clock), .line_count(line_count),
                      .pixel_count(pixel_count), .vga_out_red(vga_out_r
ed),

        .vga_out_green(vga_out_green), .vga_out_blue(vga_out_blue),

        .p1(bat2_s), .p2(bat1_s), .p3(bat3_s), .update(cache_count_enab
le),

        .pitch(pitch_s), .pitch_sel(pitch_sel), .v4(ball_vis), .in_wind
ow(in_window),

        .rom_address(rom_address), .move(move), .bat_done(bd));

//   wire write_enable;
//   wire[4:0] address;
//   wire[31:0] c_data;  // cached data.


endmodule

module vga_sync(reset, clk,  vga_out_sync_b, vga_out_blank_b,
vga_out_pixel_clock,
          vga_out_hsync, vga_out_vsync, pixel_count, line_count);

   input reset; // Active high reset, synchronous with pixel clock
   input clk; // 49.5 MHz pixel clock

   output vga_out_sync_b, vga_out_blank_b; // Composite sync/blank
outputs to DAC
   output vga_out_pixel_clock; // Pixel clock for DAC
   output vga_out_hsync, vga_out_vsync; // Sync outputs to VGA
connector
   output [10:0] pixel_count;  // the current pixel of the line.
   output [10:0] line_count;   // the current line of the frame.


///////////////////////////////////////////////////////////////////////
/////
   //
   // Timing values                                         800x600,
75Hz 49.500 800 16 80 160 600 1 2 21
   //

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

   // 800 X 600 @ 75Hz with a 49.5 MHz pixel clock
`define H_ACTIVE  800 // pixels
`define H_FRONT_PORCH     16 // pixels
`define H_SYNCH           80 // pixels
`define H_BACK_PORCH     160 // pixels
`define H_TOTAL         1056 // pixels

`define V_ACTIVE   600 // lines
`define V_FRONT_PORCH      1 // lines
`define V_SYNCH            2 // lines
`define V_BACK_PORCH      21 // lines
```

```
`define V_TOTAL          624 // lines


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

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

   reg hsync1, hsync2, vga_out_hsync, vsync1, vsync2, vga_out_vsync;
   reg [10:0] pixel_count; // Counts pixels in each line
   reg [10:0] line_count; // Counts lines in each frame



//////////////////////////////////////////////////////////////////////
/////
   //
   // Pixel and Line Counters
   //

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

   always @(posedge clk)
     if (reset)
       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;


//////////////////////////////////////////////////////////////////////
/////
   //
   // Sync and Blank Signals
   //

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

   assign vga_out_pixel_clock = ~clk;  // by spec.
   // pulse to tell the memory to refresh (signaled at the start of the
vsync.
   wire refresh;

   always @ (posedge clk)
     begin
      if (reset)
        begin
           hsync1 <= 1;
           hsync2 <= 1;
           vga_out_hsync <= 1;
           vsync1 <= 1;
           vsync2 <= 1;
           vga_out_vsync <= 1;
        end
      else
        begin
           // Horizontal sync
           if (pixel_count == (`H_ACTIVE+`H_FRONT_PORCH))
             hsync1 <= 0; // start of h_sync
           else if (pixel_count == (`H_ACTIVE+`H_FRONT_PORCH+`H_SYNCH))
             hsync1 <= 1; // end of h_sync
           // Vertical sync
           if (pixel_count == (`H_TOTAL-1))
             begin
              if (line_count == (`V_ACTIVE+`V_FRONT_PORCH))
              begin
                vsync1 <= 0; // start of v_sync
            end
              else if (line_count ==
(`V_ACTIVE+`V_FRONT_PORCH+`V_SYNCH))
                vsync1 <= 1; // end of v_sync
             end
        end

      // Delay hsync and vsync by two cycles to compensate for 2 cycles
of
      // pipeline delay in the DAC.
      hsync2 <= hsync1;
      vga_out_hsync <= hsync2;
      vsync2 <= vsync1;
      vga_out_vsync <= vsync2;
     end

   // Blanking
   assign vga_out_blank_b = ((pixel_count<`H_ACTIVE) &
(line_count<`V_ACTIVE));

   // Composite sync
   assign vga_out_sync_b = hsync1 ^ vsync1;

endmodule

module video_ca_addr (clk, reset, enc, count, data, p1, p2, p3, state);
// this module will count from 0 to all 1's after
// which the done signal will be asserted
```

```
input clk;   // pixel Clk
input reset;
input enc;   // assert high to enable count if not done
input [31:0] data;

// As these counts will be used for addressing, they need to be
registered.
output [4:0] count;  // determines the address being read from.
reg [4:0] count, count_d;

output [31:0] p1, p2, p3;
reg [31:0] p1, p2, p3;

output[2:0] state;

parameter IDLE = 0;          // power up and chip reset
parameter POINT_1 = 1;  // First pass write wait state
parameter POINT_2 = 2;
parameter POINT_3 = 3;
parameter POINT_4 = 4;
parameter WAIT_ENB_LOW = 5;

reg [2:0] state, next;

always @ (posedge clk or posedge reset)
begin
      if (reset)
      begin
            state <= IDLE;
            count <= 0;
      end
      else
      begin
            state <= next;
            count <= count_d;
      end

end


always @ (state or enc)
begin
  // defaults
  count_d = 5'h00;
  next = state;
  case (state)
     IDLE:
       begin
               if (enc)
                 next = POINT_1;
         end
     POINT_1:
       begin
              count_d = 5'h00;
              next = POINT_2;
```

```
          end
        POINT_2:
          begin
                 count_d = 5'h01;
                 p1 = data;
                 next = POINT_3;
          end
        POINT_3:
          begin
                 count_d = 5'h02;
                 p2 = data;
                 next = POINT_4;
          end
        POINT_4:
          begin
                 count_d = 5'h0;
                 p3 = data;
                 next = WAIT_ENB_LOW;
          end
           WAIT_ENB_LOW:
               begin
                  if (!enc)
                    next = IDLE;
               end
        default: next = IDLE;
  endcase
end
endmodule

module video_reg(clk, refresh, addr, data, p1, p2, p3);
// this module loads the values from the video cache into registers for
use with point track

input clk;   // pixel Clk
input refresh;   // assert high to enable count if not done
input [31:0] data;

output [31:0] p1, p2, p3;
output [4:0] addr;

reg [4:0] addr;
reg [1:0] state, nextstate;

parameter IDLE = 0;          // power up and chip reset
parameter POINT_1 = 1;  // First pass write wait state
parameter POINT_2 = 2;
parameter POINT_3 = 3;

always @ (posedge clk)
begin
      state <= nextstate;

end
always @ (state or refresh)  // or fsm_input
begin
  nextstate = state;
  p1 = p1;
  p2 = p2;
  p3 = p3;  // registers hold values
  addr = 0;
  case (state)
     IDLE:
       begin
               if (refresh)
                 nextstate = POINT_1;
        end
     POINT_1:
       begin
              addr = 5'h01;
              p1 = data;
              nextstate = POINT_2;

       end
     POINT_2:
       begin
              addr = 5'h02;
              p2 = data;
              nextstate = POINT_3;
       end
     POINT_3:
       begin
              addr = 5'h03;
              p3 = data;
              nextstate = IDLE;
       end
     default: nextstate = IDLE;
  endcase
end

endmodule
```