Appendix A. Modules in Alphabetical Order

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 6.111
// Engineer: Mariela Buchin
// Module Name: AD670

//////////////////////////////////////////////////////////////////////////////////
module AD670(clock_3mhz, reset_sync, sample, status, datain, read,
dataout);//, state);
      input clock_3mhz, reset_sync, sample, status;
      input [7:0] datain;     //these are the actual values being spit
out of the 670 chip
      output read;
      output [7:0] dataout;     //this will hold the data from the 670
chip and update only during the read cycle
//    output [3:0] state;    //only use for test module
      reg read;
      reg [7:0] LED, dataout;

//Internal State
reg [3:0] state;
reg [3:0] nextstate;
reg read_int;
reg load_e;
reg status_d1, status_d2;

parameter IDLE = 0;
parameter CONV0 = 1;
parameter CONV1 = 2;
parameter CONV2 = 3;
parameter WAITSTATUSHIGH = 4;
parameter WAITSTATUSLOW = 5;
parameter READDELAY0 = 6;
parameter READDELAY1 = 7;
parameter READCYCLE = 8;


always @ (posedge clock_3mhz )
begin
      if (reset_sync)
            begin
             state <= IDLE;
             dataout <= 8'b00000000;
             read <= 1;  //in idle state read is high
            end
      else
            begin
             state <= nextstate;
             dataout <= LED;
             status_d1 <= status;
             status_d2 <= status_d1;
```

```
             read <= read_int;
            end
      end


always @ (state or status_d2 or sample)
      begin
            read_int= 1;    //these are default values
      case (state)
            IDLE: begin
                        read_int = 1;
                        if(sample) nextstate= CONV0;
                        end
            CONV0:begin
                        read_int= 0;
                        nextstate= CONV1;
                        end
            CONV1:begin
                        read_int= 0;
                        nextstate= CONV2;
                        end
            CONV2:begin
                        read_int= 0;
                        nextstate= WAITSTATUSHIGH;
                        end
            WAITSTATUSHIGH:begin    //wait until the chip says it is
done converting
                        if (status_d2) nextstate= WAITSTATUSLOW;
                        else nextstate= WAITSTATUSHIGH;
                        end
            WAITSTATUSLOW:begin
                        if (!status_d2) nextstate= READDELAY0;
                        else nextstate= WAITSTATUSLOW;
                        end
            READDELAY0:begin
                        nextstate= READDELAY1;
                        end
            READDELAY1:begin
                        nextstate= READCYCLE;
                        end
            READCYCLE:begin
                        LED = datain;     //the output bits only change
during the read cycle
                        nextstate= IDLE;
                        end
            default: begin
                              nextstate= IDLE;
                              end
      endcase
      end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    14:20:49 04/21/06
// Design Name:
// Module Name:    WeightFSM
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////////
module WeightFSM( weightforce);

      // F = m g

      // F = Weight Force
      // m = mass
      // g = gravitational acceleration (approx 10)

      output [16:0] weightforce;
      wire   [16:0] weightforce;

      parameter mass = 1;
      parameter gravity = 10;       // Scaled

      assign weightforce = mass * gravity;

endmodule
`timescale 1ns / 1ps
```

```
////////////////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
//
// Create Date:   16:55:54 05/01/2006
// Design Name:   AD670
// Module Name:   AD670_Test.v
// Project Name:  ADConverter
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: AD670
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
////////

module AD670_Test_v;

	// Inputs
	reg clock_3mhz;
	reg reset_sync;
	reg sample;
	reg status;
	reg [7:0] datain;

	// Outputs
	wire read;
	wire [7:0] LEDs;
	wire [3:0] state;

	// Instantiate the Unit Under Test (UUT)
	AD670 uut (
		.clock_3mhz(clock_3mhz),
		.reset_sync(reset_sync),
		.sample(sample),
		.status(status),
		.datain(datain),
		.read(read),
		.LEDs(LEDs),
		.state(state)
	);

always #10 clock_3mhz = ~clock_3mhz;
	initial begin
		// Initialize Inputs
```

```
            clock_3mhz = 0;
            reset_sync = 1;
            sample = 0;
            status = 0;
            datain = 0;

            // Wait 100 ns for global reset to finish
            #50;
            reset_sync = 0;
            #50;
            sample = 1;
            #50;
            sample = 0;

      end

endmodule

`timescale 1ns / 1ps
```

```
////////////////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
//
// Create Date:   13:58:43 05/17/2006
// Design Name:   AD670
// Module Name:   ADtester.v
// Project Name:  finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: AD670
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
////////

module ADtester_v;

	// Inputs
	reg clock_3mhz;
	reg reset_sync;
	reg sample;
	reg status;
	reg [7:0] datain;

	// Outputs
	wire read;
	wire [7:0] dataout;
	wire [3:0] state;

	// Instantiate the Unit Under Test (UUT)
	AD670 uut (
		.clock_3mhz(clock_3mhz),
		.reset_sync(reset_sync),
		.sample(sample),
		.status(status),
		.datain(datain),
		.read(read),
		.dataout(dataout),
		.state(state)
	);
always #10 clock_3mhz = ~clock_3mhz;
	initial begin
```

```
            // Initialize Inputs
            clock_3mhz = 0;
            reset_sync = 1;
            sample = 0;
            status = 0;
            datain = 0;
            #50;
            reset_sync = 0;
            #50;
            sample = 1;
            datain = 9;
            #50;
            sample = 0;
            #50;
            status = 1;
            #50;
            status = 0;
            #200;
            sample = 1;
            datain = 3;
            #50;
            sample = 0;
            #50;
            status = 1;
            #50;
            status = 0;

      end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
//
// Create Date:    21:51:56 05/07/06
// Design Name:
// Module Name:    Alpha1
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////
module Alpha1(clk, pitch, alpha);

      input clk;
      input [9:0] pitch;

      output [7:0] alpha;
      wire   [7:0] alpha;

      rom alpha_rom(pitch, clk, alpha);

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
//
// Create Date:    14:30:12 04/27/06
// Design Name:
// Module Name:    Altitude_Vel
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////
module Altitude_Vel( velocity, del_altitude);

      // Inputs

      input signed [17:0] velocity;

      // Intermediates

            // Velocity is in mph, Altitude needs to be calculated
Feet/frame
            // MPH * Ft/M * hr/sec * sec/frames

      wire signed [17:0] fpf_velocity;          // Feet per Frame

      assign fpf_velocity = velocity[17] ? (-22 * velocity) / 1024 :
(22 * velocity) / 1024;

      // Outputs

      output signed [23:0] del_altitude;
      wire   signed [23:0] del_altitude;

      assign del_altitude[11:0]  = fpf_velocity[17:6];
      assign del_altitude[23:12] = {12{fpf_velocity[17]}};


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:
module Angle_to_360(anglein, angleout);

      input [9:0] anglein;

      wire [18:0] intermediate;

      assign intermediate = anglein * 359;

      output [8:0] angleout;
      wire   [8:0] angleout;

      assign angleout = intermediate[18:10];

endmodule
```

```
module ASCII_Values( clk, reset, newnumber, number , string);

        input        clk;

        input reset;
        input newnumber;

        input signed [15:0] number;

        reg signed   [15:0] current_number;
        wire signed  [15:0] offset;
        reg signed   [15:0] count;
        wire signed [15:0] positive_version;

        assign positive_version = number[15] ? -1*number : number;

        assign offset = positive_version - current_number;

        output wire   [63:0] string;

        reg signed [4:0] ascii_one;
        reg signed [4:0] ascii_ten;
        reg signed [4:0] ascii_hun;
        reg signed [4:0] ascii_thou;
        reg signed [4:0] ascii_tenthou;

        reg [7:0] one, ten, hun, thou, tenthou;

        always @ (posedge clk)
        begin

        if (reset)
         begin
                //Do Something
                current_number <= 0;
                ascii_one <= 0;
                ascii_ten <= 0;
                ascii_hun <= 0;
                ascii_thou <= 0;
                ascii_tenthou <= 0;
         end
        else if (newnumber)
         begin
```

```
                count <= offset;
                current_number <= positive_version;
         end
        else if (count > 0)                // Positive Value
                 begin

                  count <= count - 1;     // Decrement COUNT

                        ascii_one <= ascii_one + 1;      // Increment
ASCII_ONE

                        if (ascii_one >= 9)
                         begin

                                ascii_one <= 0;   // Set ASCII_ONE to 0

                    ascii_ten <= ascii_ten + 1;        // Increment
ASCII_TEN

                                if (ascii_ten >= 9)
                                 begin

                                   ascii_ten <= 0;  // Set ASCII_TEN to
0

                     ascii_hun <= ascii_hun + 1;        // Increment
ASCII_HUN

                         if (ascii_hun >= 9)
                        begin

                        ascii_hun <= 0;         // Set ASCII_HUN to 0

               ascii_thou <= ascii_thou + 1;    // Increment ASCII_THOU

                           if (ascii_thou >= 9)
                                       begin

                                   ascii_thou <= 0; // Set ASCII_THOU to
0

                              ascii_tenthou <= ascii_tenthou + 1;
                                                   // Increment
                                            ASCII_TENTHOU

                                                    end
                                           end
                                    end
                         end
                 end
         else if (count < 0)         // Negative Value
         begin

                        count <= count + 1;
                                    // Decrement COUNT

                        ascii_one <= ascii_one - 1;
```

```
                                    // Increment ASCII_ONE

                            if (ascii_one < 1)
                             begin

                                    ascii_one <= 9;
                                                // Set ASCII_ONE to 0

                                    ascii_ten <= ascii_ten - 1;
                                                   // Increment ASCII_TEN

                                    if (ascii_ten < 1)
                                     begin

                                             ascii_ten <= 9;
                                                         // Set ASCII_TEN
                                                   to 0

                                             ascii_hun <= ascii_hun - 1;
                                                          // Increment
                                                   ASCII_HUN

                                             if (ascii_hun < 1)
                                              begin

                                                       ascii_hun <= 9;
                                                           // Set ASCII_HUN
                                                   to 0

                                                       ascii_thou <=
ascii_thou - 1;
                                                           // Increment
                                                   ASCII_THOU

                                                   if (ascii_thou < 1)
                                                    begin

                                                           ascii_thou <=
9;
                                                           // Set ASCII_THOU
                                                   to 0

                                               ascii_tenthou <= ascii_tenthou -
1;
                                                           // Increment
                                                    ASCII_TENTHOU

                                                           end
                                             end
                                     end
                             end
                     end
             end
always @ (ascii_one) begin
      case (ascii_one)
        0: one <= "0";
   1: one <= "1";
```

```
        2: one <= "2";
        3: one <= "3";
        4: one <= "4";
        5: one <= "5";
        6: one <= "6";
        7: one <= "7";
        8: one <= "8";
        9: one <= "9";
        default: one <= " ";
      endcase
   end

always @ (ascii_ten) begin
      case (ascii_ten)
        0: ten <= "0";
   1: ten <= "1";
        2: ten <= "2";
        3: ten <= "3";
        4: ten <= "4";
        5: ten <= "5";
        6: ten <= "6";
        7: ten <= "7";
        8: ten <= "8";
        9: ten <= "9";
        default: ten <= " ";
      endcase
   end

always @ (ascii_hun) begin
      case (ascii_hun)
        0: hun <= "0";
   1: hun <= "1";
        2: hun <= "2";
        3: hun <= "3";
        4: hun <= "4";
        5: hun <= "5";
        6: hun <= "6";
        7: hun <= "7";
        8: hun <= "8";
        9: hun <= "9";
        default: hun <= " ";
      endcase
   end

always @ (ascii_thou) begin
      case (ascii_thou)
        0: thou <= "0";
   1: thou <= "1";
        2: thou <= "2";
        3: thou <= "3";
        4: thou <= "4";
        5: thou <= "5";
        6: thou <= "6";
        7: thou <= "7";
        8: thou <= "8";
        9: thou <= "9";
        default: thou <= " ";
```

```
      endcase
   end

always @ (ascii_tenthou) begin
      case (ascii_tenthou)
        0: tenthou <= "0";
   1: tenthou <= "1";
        2: tenthou <= "2";
        3: tenthou <= "3";
        4: tenthou <= "4";
        5: tenthou <= "5";
        6: tenthou <= "6";
        7: tenthou <= "7";
        8: tenthou <= "8";
        9: tenthou <= "9";
        default: tenthou <= " ";
      endcase
   end

wire [7:0] signblank, tenblank, hundredblank, thoublank, tenthoublank;

assign tenblank  = ((number >= 0)&&(number<10  ))? " ": ((number <0
)&&(number >-10))? "-": ten;

assign hundredblank  = (number >= 10 && number<100 ) || (number >= 0 &&
      number<10) || (number <0 && number >-10) ? " ": ( number <=-10 &&
number >-100 )? "-":    hun;

assign thoublank  = ( number >= 100 && number<1000) || (number <=-10 &&
number >-100) || (number >= 10 && number<100) || (number >= 0 &&
number<10  ) || (number < 0 && number >-10 )? " ": (number <=-100 &&
number >-1000)? "-": thou;

assign tenthoublank  = ( number >= 1000 && number<10000) ||
                       ( number >= 100 && number<1000) ||
                       ( number <=-10 && number >-100)||
                       (number >= 10 && number<100) ||
                       ( number >= 0 && number<10) ||
                       (number <0 && number >-10 ) ||
                       ( number <= -100 && number>-1000) ? " ":
                       ( number <=-1000 && number >-10000)? "-":
tenthou;

assign signblank  = ((number >= 10000)&&(number<100000  ))? " ":
                    ((number <=-10000  )&&(number >-100000))? "-": " ";

assign string = {signblank, tenthoublank, thoublank, hundredblank,
tenblank,one};

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    16:18:04 05/14/06
// Design Name:
// Module Name:    compass
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////////
module compass(heading, pixel_count, line_count, pixel_r_color,
pixel_g_color, pixel_b_color);

            input [9:0] heading;
            input signed [11:0] pixel_count, line_count;
```

```
wire [9:0] actual_heading = heading + 256;

output [7:0] pixel_r_color, pixel_g_color, pixel_b_color;
wire [7:0] pixel_r_color, pixel_g_color, pixel_b_color;


parameter x_center = 505;
parameter y_center =  90;
parameter radius   =  60;

reg [23:0] pixel_color;

// HEADING LINE
wire signed [16:0] x1;
wire signed [16:0] y1;

wire signed [16:0] x2;
wire signed [16:0] y2;

// COVER UP LINE
wire signed [16:0] cx1;
wire signed [16:0] cy1;

wire signed [16:0] cx2;
wire signed [16:0] cy2;

parameter thou = 1000;

CosTest x1_val( actual_heading, thou, x1);
SinTest y1_val( actual_heading, thou, y1);

CosTest cx1_val( heading, thou, cx1);
SinTest cy1_val( heading, thou, cy1);

assign x2 = 0 - x1;
assign y2 = 0 - y1;

assign cx2 = 0 - cx1;
assign cy2 = 0 - cy1;

wire signed [16:0] b;

assign b = 2;

wire signed [16:0] b_v1_num;
wire signed [16:0] b_v1_den;
wire signed [16:0] b_v2_num;
wire signed [16:0] b_v2_den;

SinTest b_v1_num_val(actual_heading, b, b_v1_num);
SinTest b_v2_num_val(actual_heading, b, b_v2_num);
CosTest b_v1_den_val(actual_heading, b, b_v1_den);
CosTest b_v2_den_val(actual_heading, b, b_v2_den);

wire signed [16:0] current_x;
wire signed [16:0] current_y;
```

```
assign current_x = pixel_count - x_center;
assign current_y = y_center - line_count;

// TOP LINE
reg signed [16:0] value1_num1;
reg signed [16:0] value1_den1;
reg signed [16:0] value2_num1;
reg signed [16:0] value2_den1;

// BOTTOM LINE
reg signed [16:0] value1_num2;
reg signed [16:0] value1_den2;
reg signed [16:0] value2_num2;
reg signed [16:0] value2_den2;

// COVER UP LINE
reg signed [16:0] value1_num3;
reg signed [16:0] value1_den3;
reg signed [16:0] value2_num3;
reg signed [16:0] value2_den3;

// TOP LINE
reg signed [32:0] v1n_v2d1;
reg signed [32:0] v2n_v1d1;

// BOTTOM LINE
reg signed [32:0] v1n_v2d2;
reg signed [32:0] v2n_v1d2;

// COVER UP LINE
reg signed [32:0] v1n_v2d3;
reg signed [32:0] v2n_v1d3;

always @ (pixel_count or line_count)
begin

      // TOP LINE
      value1_num1 = current_x - (x1 + b_v1_num);
      value2_num1 = current_x - (x2 + b_v2_num);
      value1_den1 = current_y - (y1 - b_v1_den);
      value2_den1 = current_y - (y2 - b_v2_den);

      // BOTTOM LINE
      value1_num2 = current_x - (x1 - b_v1_num);
      value2_num2 = current_x - (x2 - b_v2_num);
      value1_den2 = current_y - (y1 + b_v1_den);
      value2_den2 = current_y - (y2 + b_v2_den);

      // COVER UP LINE
      value1_num3 = current_x - cx1;
      value2_num3 = current_x - cx2;
      value1_den3 = current_y - cy1;
      value2_den3 = current_y - cy2;

      // TOP LINE
      v1n_v2d1 = value1_num1 * value2_den1;
```

```
                v2n_v1d1 = value2_num1 * value1_den1;

                // BOTTOM LINE
                v1n_v2d2 = value1_num2 * value2_den2;
                v2n_v1d2 = value2_num2 * value1_den2;

                // COVER UP LINE
                v1n_v2d3 = value1_num3 * value2_den3;
                v2n_v1d3 = value2_num3 * value1_den3;


             if (((line_count-y_center)*(line_count-
y_center)+(pixel_count-x_center)*(pixel_count-
x_center))<=(radius*radius)          )
                       begin
                      if ((v1n_v2d1 > v2n_v1d1)||(v1n_v2d2 < v2n_v1d2)
|| (v1n_v2d3 > v2n_v1d3))
                                     pixel_color =
24'b1111_1111_1111_1111_1111_1111;
                                else
                                       pixel_color = 24'h0;
                  end
                 else
                       pixel_color =
24'b0000_0000_0000_0000_0000_0000;

            end


            assign pixel_r_color = pixel_color[23:16];
            assign pixel_g_color = pixel_color[15:8];
            assign pixel_b_color = pixel_color[7:0];




endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    14:13:21 05/01/06
// Design Name:
// Module Name:    CosTest
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////////
module CosTest( theta, value_in, value_out);

      input [9:0] theta;
      input signed [16:0] value_in;

      output signed [16:0] value_out;
      wire   signed [16:0] value_out;

      wire signed [8:0] cosout;

      cos cos12( theta, cosout);

      wire signed [25:0] intermediate;

      assign intermediate = value_in * cosout;

      assign value_out = intermediate [23:7];

endmodule
module char_string_display (vclock,hcount,vcount,pixel,cstring,cx,cy);

   parameter NCHAR = 8; // number of 8-bit characters in cstring
   parameter NCHAR_BITS = 3; // number of bits in NCHAR

   input vclock;  // 65MHz clock
   input signed [11:0] hcount;      // horizontal index of current
pixel (0..1023)
   input signed [11:0]  vcount; // vertical index of current pixel
(0..767)
   output [2:0] pixel;  // char display's pixel
   input [NCHAR*8-1:0] cstring;     // character string to display
   input signed [11:0] cx;
   input signed [11:0]  cy;

   // 1 line x 8 character display (8 x 12 pixel-sized characters)

   wire signed [11:0]   hoff = hcount-1-cx;
   wire signed [11:0]   voff = vcount-cy;
   wire [NCHAR_BITS-1:0] column = NCHAR-1-hoff[NCHAR_BITS-1+4:4];  // <
NCHAR
   wire [2:0]     h = hoff[3:1];            // 0 .. 7
   wire [3:0]     v = voff[4:1];            // 0 .. 11

   // look up character to display (from character string)
   reg [7:0]  char;
   integer  n;
   always @ (*)
      for (n=0 ; n<8 ; n = n+1 )          // 8 bits per character
(ASCII)
       char[n] <= cstring[column*8+n];

   // look up raster row from font rom
   wire reverse = char[7];
```

```
   wire [10:0] font_addr = char[6:0]*12 + v;    // 12 bytes per
character
   wire [7:0]  font_byte;
   font_rom f(font_addr,vclock,font_byte);

   // generate character pixel if we're in the right h,v area
   wire [2:0] cpixel = (font_byte[7 - h] ^ reverse) ? 7 : 0;
   wire dispflag = ((hcount > cx) & (vcount >= cy) & (hcount <=
cx+NCHAR*16)
                & (vcount < cy + 24));
   wire [2:0] pixel = dispflag ? cpixel : 0;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    12:39:04 04/27/06
// Design Name:
// Module Name:    DelVel_Force
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
////////
module DelVel_Force( F, del_vel);

      // Inputs Force

      input signed [16:0] F;

      // Outputs the change in velocity

      output signed [17:0] del_vel;
      wire   signed [17:0] del_vel;

      // Change in Velocity simulated to be the Force multiplied by a
time roughly equal to 1/1000
      // Thus a bit shift works.

      assign del_vel[9:0]   = F[16:7];
      assign del_vel[17:10] = {8{F[16]}};


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
////////
// Company:  6.111
// Engineer: Fishy
//
// Create Date:    14:50:04 03/22/06
// Design Name:
// Module Name:    display_field
// Project Name:
// Target Device:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////////
module display_field(roll, pitch, pixel_count, line_count,
pixel_r_color, pixel_g_color, pixel_b_color);

            input [9:0] roll;
            input signed [9:0] pitch;
            input signed [11:0] pixel_count, line_count;

            output [7:0] pixel_r_color, pixel_g_color, pixel_b_color;
            wire [7:0] pixel_r_color, pixel_g_color, pixel_b_color;


            parameter x_center = 320;
            parameter y_center = 310;
            parameter HEIGHT   = 300;
            parameter WIDTH    = 300;
            parameter cornerx  = 170;
            parameter cornery  = 160;

            reg [23:0] pixel_color;

            wire signed [16:0] x1;
            wire signed [16:0] y1;

            wire signed [16:0] x2;
            wire signed [16:0] y2;

            parameter thou = 1000;

            CosTest x1_val( roll, thou, x1);
            SinTest y1_val( roll, thou, y1);

            assign x2 = 0 - x1;
            assign y2 = 0 - y1;

            parameter b_scaler = 2;

            wire signed [16:0] b_v1_num;
            wire signed [16:0] b_v1_den;
            wire signed [16:0] b_v2_num;
            wire signed [16:0] b_v2_den;

            wire signed [16:0] b;

            wire signed [8:0] halfpitch;

            assign halfpitch = pitch[8:0];

            assign b = b_scaler * halfpitch;
```

```
            SinTest b_v1_num_val(roll, b, b_v1_num);
            SinTest b_v2_num_val(roll, b, b_v2_num);
            CosTest b_v1_den_val(roll, b, b_v1_den);
            CosTest b_v2_den_val(roll, b, b_v2_den);

            wire signed [16:0] current_x;
            wire signed [16:0] current_y;

            assign current_x = pixel_count - x_center;
            assign current_y = y_center - line_count;

            reg signed [16:0] value1_num;
            reg signed [16:0] value1_den;
            reg signed [16:0] value2_num;
            reg signed [16:0] value2_den;

            reg signed [32:0] v1n_v2d;
            reg signed [32:0] v2n_v1d;

            always @ (pixel_count or line_count)
            begin

                  value1_num = current_x - (x1 + b_v1_num);
                  value2_num = current_x - (x2 + b_v2_num);
                  value1_den = current_y - (y1 - b_v1_den);
                  value2_den = current_y - (y2 - b_v2_den);

                  v1n_v2d = value1_num * value2_den;
                  v2n_v1d = value2_num * value1_den;

              if ((cornerx<=pixel_count)&&(pixel_count<cornerx+WIDTH)
                  &&(cornery<=line_count)&&
(line_count<cornery+HEIGHT))
                        begin
                       if (v1n_v2d < v2n_v1d)
                              begin
                               if((pitch >= 256) || (pitch < -256))
                                    pixel_color = 24'h5C3317;
                           else
                                    pixel_color = 24'h0000FF;
                              end
                           else
                              begin
                               if((pitch >= 256) || (pitch < -256))
                                    pixel_color = 24'h0000FF;
                           else
                                    pixel_color = 24'h5C3317;
                              end
                        end
                  else
                        pixel_color =
24'b0000_0000_0000_0000_0000_0000;

            end
```

```
            assign pixel_r_color = pixel_color[23:16];
            assign pixel_g_color = pixel_color[15:8];
            assign pixel_b_color = pixel_color[7:0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    14:20:24 04/21/06
// Design Name:
```

```
// Module Name:    DragFSM
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
////////
module DragFSM(airdensity, velocity, dragforce);

        // D = Cd (r V^2 / 2) A

        // D  = Drag Force
        // Cd = Drag Coefficient
        //      .045 for typical air foil
        //         .295 for typical bullet
        // r  = air density
        // V  = Velocity
        // A  = Frontal Area of Plane

      input signed [4:0] airdensity;
      input signed [17:0] velocity;

            // Constants

      parameter signed Cd = 1;            // Cd = 2, .5 taken care of
here (inbetween bullet and airfoil)
      parameter signed A  = 1;            // Area of Plane

      output signed [16:0] dragforce;
      wire   signed [16:0] dragforce;

      wire signed [41:0] drag;
      wire signed [16:0] intermediate;

      // Airdensity Hardwired
      assign drag = Cd * 6 * velocity * velocity * A;
      assign intermediate = drag[41:25];
      assign dragforce = velocity[17] ? -1*intermediate : intermediate;

endmodule
`timescale 1ns / 1ps



//////////////////////////////////////////////////////////////////////
////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    14:20:00 04/21/06
```

```
// Design Name:
// Module Name:    FlightFSM
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
/////////
module FlightFSM(reset, pixel_clock, calc_enable, pitchangle,
rollangle, throttle, rho, alpha, alt_out, vel_out, vert_vel_out,
pitch_out, roll_out, yaw_out, endgame//);
                                 /*,totalvertforce, totalhorzforce,
totaldirvelforce, del_velocity, del_vert_velocity,
                                 del_horz_velocity, dragforce,*/,
dragforce_dir, dragforce_vert, liftforce);

      // For Debugging
      output [16:0] dragforce_dir;
      output [16:0] dragforce_vert;
      //output [16:0] thrustforce;
      output [16:0] liftforce;
      //output [16:0] weightforce;
 /*
      output [16:0] totalvertforce;
      output [16:0] totalhorzforce;
      output [16:0] totaldirvelforce;
      output [17:0] del_velocity;
      output [17:0] del_vert_velocity;
      output [17:0] del_horz_velocity;
 */

      //output signed [17:0] del_pitch;
      //output signed [17:0] del_yaw;
      //output signed [17:0] del_roll;
            // Calculations performed at every posedge of calc_enable


      input reset;
      input pixel_clock;
      input calc_enable;

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

            //    Control Inputs

      input signed [9:0] pitchangle;
      input signed [9:0] rollangle;

      input [9:0] throttle;
```

```
// STORED

            // Saved Values - Need Registers

            // Velocities

      reg signed [17:0] velocity;           // Directional Velocity
      reg signed [17:0] vert_velocity;      // Vertical Velocity
      reg signed [17:0] horz_velocity;      // Horizontal Velocity

            // Altitude

      reg signed [23:0] altitude;

            // Angles

      reg signed [17:0] pitch;
      reg signed [17:0] roll;
      reg signed [17:0] direction;

//Variables

            // Internally Kept Values / Parameters of the Plane:
            //    Mass
            //          Frontal Area
            //          Wing Area

      parameter mass = 1;                        // Mass of Plane
      parameter frontalarea = 1;          //    Frontal Area of Plane
      parameter wingarea = 1;             // Area of Wing of Plane

            // Parameters Necessary for Flight

            //          RHO
            //          rho =  6                          altitude <
5,000
            //          rho =  5           5,000 < altitude < 10,000
            //          rho =  4          10,000 < altitude < 15,000
            //          rho =  3          15,000 < altitude < 20,000
            //          rho =  2          20,000 < altitude < 25,000
            //          rho =  1          25,000 < altitude < 30,000
            //          rho =  0          30,000 < altitude

      input [4:0] rho;                    // Density of Air
      input signed [7:0] alpha;

            // Four main forces:
            // Lift
            //          Acts Perpendicular to the Horizon and
Perpendicular to Roll Angle of Wings
            //          Affects Altitude and Directional Rotation
            // Weight
            //          Acts Straight Down
            //          Affects Velocity and Altitude
            // Thrust
```

```
            //          Acts at the Angle of Attack / Angle of Pitch
            //          Affects Velocity and Altitude
            // Drag
            //          Acts at the Angle of Attack / Angle of Pitch
            //          Affects Velocity and Altitude

      wire signed [16:0] dragforce_dir;
      wire signed [16:0] dragforce_horz;
      wire signed [16:0] dragforce_vert;
      wire signed [16:0] thrustforce;
      wire signed [16:0] liftforce;
      wire signed [16:0] weightforce;

      DragFSM           drag_dir( rho, velocity, dragforce_dir);
      DragFSM           drag_horz( rho, horz_velocity, dragforce_horz);
      DragFSM           drag_vert( rho, vert_velocity, dragforce_vert);
      ThrustFSM   thrust( rho, throttle, thrustforce);
      LiftFSM           lift( rho, velocity, alpha, liftforce);
      WeightFSM   weight( weightforce);

            // Two Main Torques:

            // Roll           Omega is the Angular Velocity
            // Pitch          Omega is the Angular Velocity

            //    Yaw         Omega is the Angular Velocity
            //          Yaw is solely for calculating the Direction

      wire signed [16:0] pitch_omega;
      wire signed [16:0] yaw_omega;

      wire signed [16:0] roll_omega;

      PitchOmega mod_pitch_yaw_omega( pitchangle, roll, pitch_omega,
yaw_omega);
      RollOmega  mod_roll_omega( rollangle, roll_omega);

            // Resulting Vertical Force, Horizontal Force, and Velocity
Direction Force

      wire signed [16:0] totalvertforce;        // Vertical
      wire signed [16:0] totalhorzforce;        // Horizontal in  Yaw
direction
      wire signed [16:0] totaldirvelforce;      // Direction of
Velocity

      VertHorzForces    forces(dragforce_dir, dragforce_horz,
dragforce_vert, thrustforce, liftforce, weightforce, pitch, roll,
totalvertforce, totalhorzforce, totaldirvelforce);


// DELTA

            // Delta Velocities

      wire signed [17:0] del_velocity;
      wire signed [17:0] del_vert_velocity;
```

```
wire signed [17:0] del_horz_velocity;

      // Direction Velocity
DelVel_Force vel( totaldirvelforce, del_velocity);
      // Vertical Velocity
DelVel_Force vertvel( totalvertforce, del_vert_velocity);
      // Horizontal Velocity
DelVel_Force horzvel( totalhorzforce, del_horz_velocity);

      // Delta Altitude

wire signed [23:0] del_altitude;

Altitude_Vel del_alt( vert_velocity, del_altitude);

      // Delta Angles

wire signed [17:0] del_pitch;
wire signed [17:0] del_yaw;
wire signed [17:0] del_roll;

DelVel_Force mod_del_pitch( pitch_omega, del_pitch);
DelVel_Force mod_del_yaw( yaw_omega, del_yaw);
DelVel_Force mod_del_roll( roll_omega, del_roll);


always @ (posedge pixel_clock)
begin

      if (reset)
            begin

            // Restore Velocities
            velocity <= 200 * 256;
            vert_velocity <= 0;
            horz_velocity <= 0;

            // Restore Rotations
            pitch <= 0;
            roll <= 0;
            direction <= 30 * 256;

            // Restore Altitude
            altitude <= 15000 * 256;

            // Game Reset
            endgame <= 0;

            end
            else if (altitude <= 50)
            begin
            endgame[0] <= 1;
            end
            else if (altitude >= 31000 * 256)
            begin
            endgame[1] <= 1;
            end
```

```
                else if (calc_enable)
                begin


                // Increment Velocities

                velocity <= velocity + 32*del_velocity;
                vert_velocity <= vert_velocity +
32*del_vert_velocity;
                horz_velocity <= horz_velocity +
32*del_horz_velocity;

                // Increment Altitude
                if (vert_velocity < 0)
                altitude <= altitude - 64*del_altitude;
                else
                altitude <= altitude + 64*del_altitude;

                // Increment Angles

                pitch <= pitch + 6*del_pitch;
                roll <= roll + 6*del_roll;
                direction <= direction + 3*del_yaw;

                end

      end

      // Altitude
      output [15:0] alt_out;
      // Velocity
      output [9:0] vel_out;
      output [9:0] vert_vel_out;
      // Angles
      output [9:0] pitch_out;
      output [9:0] roll_out;
      output [9:0] yaw_out;
      // Wires
      wire [15:0] alt_out;
      wire [9:0] vel_out;
      wire [9:0] vert_vel_out;
      wire [9:0] pitch_out;
      wire [9:0] roll_out;
      wire [9:0] yaw_out;

      // Assignments
      assign alt_out = altitude[23:8];
      assign vel_out = velocity[17:8];
      assign vert_vel_out = vert_velocity[17:8];
      assign pitch_out = pitch[17:8];
      assign roll_out = roll[17:8];
      assign yaw_out = direction[17:8];

endmodule
/*****************************************************************
*********
```

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

```
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity
synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file font_rom.v when simulating
// the core, font_rom. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module font_rom(
      addr,
      clk,
      dout);


input [10 : 0] addr;
input clk;
output [7 : 0] dout;

// synopsys translate_off

      BLKMEMSP_V6_1 #(
            11,   // c_addr_width
            "0",  // c_default_data
            1536, // c_depth
            0,    // c_enable_rlocs
            0,    // c_has_default_data
            0,    // c_has_din
            0,    // c_has_en
            0,    // c_has_limit_data_pitch
            0,    // c_has_nd
            0,    // c_has_rdy
            0,    // c_has_rfd
            0,    // c_has_sinit
            0,    // c_has_we
            18,   // c_limit_data_pitch
            "font_rom.mif",   // c_mem_init_file
            0,    // c_pipe_stages
            0,    // c_reg_inputs
            "0",  // c_sinit_value
            8,    // c_width
            0,    // c_write_mode
            "0",  // c_ybottom_addr
            1,    // c_yclk_is_rising
            1,    // c_yen_is_high
            "hierarchy1",     // c_yhierarchy
            0,    // c_ymake_bmm
            "16kx1",    // c_yprimitive_type
            1,    // c_ysinit_is_high
            "1024",     // c_ytop_addr
            0,    // c_yuse_single_primitive
            1,    // c_ywe_is_high
            1)    // c_yydisable_warnings
      inst (
            .ADDR(addr),
```

```
            .CLK(clk),
            .DOUT(dout),
            .DIN(),
            .EN(),
            .ND(),
            .RFD(),
            .RDY(),
            .SINIT(),
            .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of font_rom is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of font_rom is "black_box"

endmodule

//`include "display_16hex.v"
//`include "cstringdisp.v"
//`include "font_rom.v"

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




module labkit   (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in,
ac97_synch,
             ac97_bit_clock,
           //beep
             vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
             vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
             vga_out_vsync,

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

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

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

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

              clock_feedback_out, clock_feedback_in,

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

              rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

              mouse_clock, mouse_data, keyboard_clock, keyboard_data,

              clock_27mhz, clock1, clock2,

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

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

              switch,

              led,

              user1, user2, user3, user4,

              daughtercard,

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

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

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

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

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

   input  [19:0] tv_in_ycrcb;
```

```
   input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
tv_in_aef,
         tv_in_hff, tv_in_aff;
   output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock,
tv_in_iso,
         tv_in_reset_b, tv_in_clock;
   inout  tv_in_i2c_data;

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

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

   input  clock_feedback_in;
   output clock_feedback_out;

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

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

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

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

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

   inout [43:0] daughtercard;

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

   output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
             analyzer4_data;
```

```
   output analyzer1_clock, analyzer2_clock, analyzer3_clock,
analyzer4_clock;


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

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

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

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

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

   // SRAMs
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_clk = 1'b0;
   assign ram0_cen_b = 1'b1;
   assign ram0_ce_b = 1'b1;
   assign ram0_oe_b = 1'b1;
   assign ram0_we_b = 1'b1;
   assign ram0_bwe_b = 4'hF;
   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
```

```
   assign ram1_clk = 1'b0;
   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b1;
   assign ram1_oe_b = 1'b1;
   assign ram1_we_b = 1'b1;
   assign ram1_bwe_b = 4'hF;
   assign clock_feedback_out = 1'b0;
   // clock_feedback_in is an input

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

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

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

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

   // disp_data_in is an input

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

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

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
   assign systemace_data = 16'hZ;
   assign systemace_address = 7'h0;
   assign systemace_ce_b = 1'b1;
   assign systemace_we_b = 1'b1;
```

```
   assign systemace_oe_b = 1'b1;
   // systemace_irq and systemace_mpbrdy are inputs

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


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

   // use FPGA's digital clock manager to produce a
   // 65MHz clock (actually 64.8MHz)

      wire pclk, pixel_clock;
   DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 29
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 27
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   // synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

      wire [7:0] attitude_red, attitude_green, attitude_blue;

  wire power_on_reset;    // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(pixel_clock), .Q(power_on_reset),
              .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;

      wire reset, user_reset;
      debounce resetdebounce(power_on_reset, pixel_clock, ~button0,
user_reset);
      assign reset = user_reset | power_on_reset;


      wire upbutton, downbutton, rightbutton, leftbutton;
      wire [3:0] speed;
      assign speed = switch[3:0];

      debounce upbuttondebounce(1'b1, pixel_clock, ~button_up,
upbutton);
      debounce downbuttondebounce(1'b1, pixel_clock, ~button_down,
downbutton);
      debounce rightbuttondebounce(1'b1, pixel_clock, ~button_right,
rightbutton);
      debounce leftbuttondebounce(1'b1, pixel_clock, ~button_left,
leftbutton);
      debounce obuttondebounce(1'b1, pixel_clock, ~button1, button_1);
      debounce tbuttondebounce(1'b1, pixel_clock, ~button2, button_2);
   // generate basic XVGA video signals
```

```
      wire signed [11:0] hcount;
   wire signed [11:0] vcount;
   wire hsync,vsync,blank, hblank, vblank;

   vga vga1(reset, pixel_clock,hcount,vcount,hsync,vsync,blank, vblank,
hblank);


      wire newframe;
      wire [63:0] stringalt;
      wire [63:0] stringvel;
      wire [63:0] stringhead;
      wire [63:0] stringvert;

      // Inputs
      wire [9:0] yoke_roll;
      wire [9:0] yoke_pitch;
      wire [9:0] throttle_input;
      wire [7:0] throt;

      assign throttle_input[9:2] = throt;
      assign throttle_input[1:0] = 0;

      // Output
      wire [9:0] direction;
      wire [8:0] heading;

      Angle_to_360 heading_convert( direction, heading);

      wire [15:0] altitude;
      wire  [9:0] velocity;
      wire  [9:0] vertical_velocity;
      wire  [9:0] pitch;
      wire  [9:0] roll;

      wire [15:0] big_vel;
      assign big_vel[15:10] = {6{velocity[9]}};        // Sign
Extension
      assign big_vel[9:0] = velocity;

      wire [15:0] large_head;
      assign large_head[15:9] = 0;
      assign large_head[8:0]  = heading;

      wire [15:0] big_vert_vel;
      assign big_vert_vel[15:10] = {6{vertical_velocity[9]}};     //
Sign Extension
      assign big_vert_vel[9:0] = vertical_velocity;

      // ENDGAME
      wire [1:0] endgame;
```

```
      // Debugging
      wire [16:0] del_pitch;
      wire [16:0] del_yaw;
      wire [16:0] del_roll;

      PlanePhysics physics( reset, pixel_clock, newframe,
throttle_input, yoke_pitch, yoke_roll,
                                                altitude, velocity,
vertical_velocity, pitch, roll, direction, endgame, del_pitch, del_yaw,
del_roll);

      //assign led = vertical_velocity[9:2];

  ASCII_Values alt_ascii(pixel_clock, reset, newframe, altitude,
stringalt);
  ASCII_Values vel_ascii(pixel_clock, reset, newframe, big_vel,
stringvel);
  ASCII_Values head_ascii(pixel_clock, reset, newframe, large_head,
stringhead);
  ASCII_Values vert_ascii(pixel_clock, reset, newframe, big_vert_vel,
stringvert);

      // Strings for forces for debugging
      wire [63:0] liftstring;
      wire [63:0] thruststring;
      wire [63:0] weightstring;
      ASCII_Values lift_ascii(pixel_clock, reset, newframe,
del_pitch[16:1], thruststring);
      ASCII_Values thrust_ascii(pixel_clock, reset, newframe,
del_yaw[16:1], liftstring);
      ASCII_Values weight_ascii(pixel_clock, reset, newframe,
del_roll[16:1], weightstring);


   // character display module: sample string in middle of screen
   wire [63:0] cstring = stringalt;
   wire [2:0]  cdpixel;
   char_string_display cd(pixel_clock,hcount,vcount,
                   cdpixel,cstring,12'd220,12'd30);

      wire [63:0] cstring2 = stringvel;
   wire [2:0]  cdpixel2;
   char_string_display cd2(pixel_clock,hcount,vcount,
                   cdpixel2,cstring2,12'd220,12'd50);


      wire [63:0] cstring3 = stringhead;
   wire [2:0]  cdpixel3;
   char_string_display cd3(pixel_clock,hcount,vcount,
                   cdpixel3,cstring3,12'd220,12'd70);

      wire [63:0] cstring4 = stringvert;
   wire [2:0]  cdpixel4;
   char_string_display cd4(pixel_clock,hcount,vcount,
                   cdpixel4,cstring4,12'd220,12'd90);

      wire [63:0] cstring5 = "ALTITUDE";
```

```
   wire [2:0]  cdpixel5;
   char_string_display cd5(pixel_clock,hcount,vcount,
                   cdpixel5,cstring5,12'd70,12'd30);


      wire [63:0] cstring6 = "SPEED   ";
   wire [2:0]  cdpixel6;
   char_string_display cd6(pixel_clock,hcount,vcount,
                   cdpixel6,cstring6,12'd70,12'd50);


      wire [63:0] cstring7 = "HEADING ";
   wire [2:0]  cdpixel7;
   char_string_display cd7(pixel_clock,hcount,vcount,
                   cdpixel7,cstring7, 12'd70,12'd70);

      wire [63:0] cstring8 = "A. RATE ";
   wire [2:0]  cdpixel8;
   char_string_display cd8(pixel_clock,hcount,vcount,
                   cdpixel8,cstring8, 12'd70,12'd90);


//    // Display Forces for Debugging:
//    wire [63:0] cstring14 = "VERTICAL";
//   wire [2:0]  cdpixel14;
//   char_string_display cd14(pixel_clock,hcount,vcount,
//                   cdpixel14,cstring14, 12'd70,12'd110);
//
//    wire [63:0] cstring17 = liftstring;
//   wire [2:0]  cdpixel17;
//   char_string_display cd17(pixel_clock,hcount,vcount,
//                   cdpixel17,cstring17, 12'd220,12'd110);
//
//    wire [63:0] cstring15 = "DIRECT  ";
//   wire [2:0]  cdpixel15;
//   char_string_display cd15(pixel_clock,hcount,vcount,
//                   cdpixel15,cstring15, 12'd70,12'd130);
//
//    wire [63:0] cstring18 = thruststring;
//   wire [2:0]  cdpixel18;
//   char_string_display cd18(pixel_clock,hcount,vcount,
//                   cdpixel18,cstring18, 12'd220,12'd130);
//
//    wire [63:0] cstring16 = "LIFT    ";
//   wire [2:0]  cdpixel16;
//   char_string_display cd16(pixel_clock,hcount,vcount,
//                   cdpixel16,cstring16, 12'd70,12'd150);
//
//    wire [63:0] cstring19 = weightstring;
//   wire [2:0]  cdpixel19;
//   char_string_display cd19(pixel_clock,hcount,vcount,
//                   cdpixel19,cstring19, 12'd220,12'd150);

      // Compass Points
      wire [63:0] cstring9 = "N       ";
      wire [2:0] cdpixel9;
      char_string_display cd9(pixel_clock, hcount, vcount,
```

```
                    cdpixel9, cstring9, 12'd495, 12'd3);

    wire [63:0] cstring10 = "S       ";
    wire [2:0] cdpixel10;
    char_string_display cd10(pixel_clock, hcount, vcount,
                    cdpixel10, cstring10, 12'd495, 12'd155);

    wire [63:0] cstring11 = "W       ";
    wire [2:0] cdpixel11;
    char_string_display cd11(pixel_clock, hcount, vcount,
                    cdpixel11, cstring11, 12'd420, 12'd80);

    wire [63:0] cstring12 = "E       ";
    wire [2:0] cdpixel12;
    char_string_display cd12(pixel_clock, hcount, vcount,
                    cdpixel12, cstring12, 12'd570, 12'd80);




    //GAME OVER 1
    wire [63:0] cstring13 = "GAME";
    wire [2:0] cdpixel13;
    char_string_display cd13(pixel_clock, hcount, vcount,
                    cdpixel13, cstring13, 12'd260, 12'd200);

wire [63:0] cstring14 = "OVER!";
    wire [2:0] cdpixel14;
    char_string_display cd14(pixel_clock, hcount, vcount,
                    cdpixel14, cstring14, 12'd358, 12'd200);

    wire [63:0] cstring15 = "A LITTLE";
    wire [2:0] cdpixel15;
    char_string_display cd15(pixel_clock, hcount, vcount,
                    cdpixel15, cstring15, 12'd40, 12'd150);

    wire [63:0] cstring16 = "LOW";
    wire [2:0] cdpixel16;
    char_string_display cd16(pixel_clock, hcount, vcount,
                    cdpixel16, cstring16, 12'd100, 12'd150);

    wire [63:0] cstring17 = "THERE, ";
    wire [2:0] cdpixel17;
    char_string_display cd17(pixel_clock, hcount, vcount,
                    cdpixel17, cstring17, 12'd220, 12'd150);


    wire [63:0] cstring18 = "BUDDY!";
    wire [2:0] cdpixel18;
    char_string_display cd18(pixel_clock, hcount, vcount,
                    cdpixel18, cstring18, 12'd320, 12'd150);


    //GAME OVER 2

    wire [63:0] cstring24 = "GAME";
    wire [2:0] cdpixel24;
```

```
      char_string_display cd24(pixel_clock, hcount, vcount,
                        cdpixel24, cstring24, 12'd260, 12'd200);

   wire [63:0] cstring19 = "OVER!";
      wire [2:0] cdpixel19;
      char_string_display cd19(pixel_clock, hcount, vcount,
                        cdpixel19, cstring19, 12'd358, 12'd200);

      wire [63:0] cstring20 = "A LITTLE";
      wire [2:0] cdpixel20;
      char_string_display cd20(pixel_clock, hcount, vcount,
                        cdpixel20, cstring20, 12'd40, 12'd150);

      wire [63:0] cstring21 = "HIGH";
      wire [2:0] cdpixel21;
      char_string_display cd21(pixel_clock, hcount, vcount,
                        cdpixel21, cstring21, 12'd115, 12'd150);

      wire [63:0] cstring22 = "THERE, ";
      wire [2:0] cdpixel22;
      char_string_display cd22(pixel_clock, hcount, vcount,
                        cdpixel22, cstring22, 12'd235, 12'd150);


      wire [63:0] cstring23 = "BUDDY!";
      wire [2:0] cdpixel23;
      char_string_display cd23(pixel_clock, hcount, vcount,
                        cdpixel23, cstring23, 12'd335, 12'd150);




   wire [23:0] throttleout1;
      wire [23:0] throttleout2;

   reg [7:0] red, green, blue;
   reg b,hs,vs;
      wire [7:0] compassred, compassgreen, compassblue;



//GIANT MUX


   always @(posedge pixel_clock) begin
      hs <= hsync;
      vs <= vsync;
      b <= blank;
            if (endgame[0]) begin

//END GAME HI

            red <= {8{cdpixel24[2]}}
|{8{cdpixel23[2]}}|{8{cdpixel22[2]}}|{8{cdpixel21[2]}}|{8{cdpixel20[2]}
}|{8{cdpixel19[2]}};
            green <=    {8{0}};
            blue <= {8{0}};
```

```
            end
            else if (endgame[1])    begin

//END GAME LOW

            red <= {8{cdpixel18[2]}}
|{8{cdpixel17[2]}}|{8{cdpixel16[2]}}|{8{cdpixel15[2]}}|{8{cdpixel14[2]}
}|{8{cdpixel13[2]}};
            green <=    {8{0}};
            blue <= {8{0}};
            end
            else
            begin

//GAME MODE

      red <= {8{cdpixel[2]}} | {8{cdpixel2[2]}} | {8{cdpixel3[2]}} |
//                       {8{cdpixel14[2]}} | {8{cdpixel15[2]}} |
{8{cdpixel16[2]}} | {8{cdpixel17[2]}} | {8{cdpixel18[2]}} |
{8{cdpixel19[2]}} |
                   {8{cdpixel4[2]}} | {8{cdpixel5[2]}} |
                         {8{cdpixel6[2]}} | {8{cdpixel7[2]}} |
{8{cdpixel8[2]}} |{8{cdpixel9[2]}} |{8{cdpixel10[2]}} |
      {8{cdpixel11[2]}} | {7{cdpixel12[2]}} |
                         attitude_red | throttleout1[23:16] |
throttleout2[23:16] | compassred;

      green <= {8{cdpixel[1]}} | {8{cdpixel2[1]}} | {8{cdpixel3[1]}} |
//                    {8{cdpixel14[1]}} | {8{cdpixel15[1]}} |
{8{cdpixel16[1]}} | {8{cdpixel17[1]}} | {8{cdpixel18[1]}} |
{8{cdpixel19[1]}} |
                     {8{cdpixel4[1]}} | {8{cdpixel5[1]}} |
                              {8{cdpixel6[1]}} | {8{cdpixel7[1]}} |
{8{0}} |{8{cdpixel8[1]}} |{8{cdpixel10[1]}} |    {8{cdpixel11[1]}}
|{8{cdpixel12[1]}} |
                              attitude_green |throttleout1[15:8] |
throttleout2[15:8] | compassgreen;

      blue <= {8{cdpixel[0]}} | {8{cdpixel2[0]}} | {8{cdpixel3[0]}} |
//                   {8{cdpixel14[0]}} | {8{cdpixel15[0]}} |
{8{cdpixel16[0]}} | {8{cdpixel17[0]}} | {8{cdpixel18[0]}} |
{8{cdpixel19[0]}} |
                     {8{cdpixel4[0]}} | {8{cdpixel5[0]}} |
                           {8{cdpixel6[0]}} | {8{cdpixel7[0]}} | {8{0}}
| {8{cdpixel8[0]}} |{8{cdpixel10[0]}} |   {8{cdpixel11[0]}}
|{8{cdpixel12[0]}} |
                           attitude_blue | throttleout1[7:0] |
throttleout2[7:0] |compassblue;
            end
   end

   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~pixel_clock.
   assign vga_out_red = red;
   assign vga_out_green = green;
   assign vga_out_blue = blue;
   assign vga_out_sync_b = 1'b1;    // not used
```

```
   assign vga_out_blank_b = ~b;
   assign vga_out_pixel_clock = ~pixel_clock;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;



   //Hardware
Section_____________________________________________________________
wire [7:0] leds;
assign led = leds;

 sensors needmorepower(clock_27mhz, pixel_clock, newframe, button0,
switch[0], user1, user3, user4, leds, throt, yoke_pitch, yoke_roll);

//_____________________________________________________________________________
______


//GENERATES NEW FRAME SIGNAL

      reg vsyncdelayed;

      always @ (posedge pixel_clock) begin
   vsyncdelayed <= vsync;
   end

      assign newframe = vsyncdelayed & ~vsync;







      reg signed [9:0] number_reg;
      wire signed [9:0] number;
      assign number [9] = throt[7];
      assign number [8] = throt[7];
      assign number [7:0] = throt[7:0];


      always @(posedge pixel_clock)
      number_reg <= number;



   //For debugging elements of the display
      //divider divspeed (pixel_clock, reset, speed,
speedenableangles);

      //roll roll_val(pixel_clock, speedenableangles, reset,
rightbutton, leftbutton, roll);

      //pitch pitch_val(pixel_clock, speedenableangles, reset,
upbutton, downbutton, pitch);
```

```
//    throttle_value throttle(pixel_clock, speedenableangles, reset,
button_2, button_1, number);



      display_field display(roll, pitch, hcount, vcount, attitude_red,
attitude_green, attitude_blue);
      //throttlet t(.pixelclock(pixel_clock), .reset(reset),
      //                      .up(button2),. down(button1),
.throttle(throttletest));
      throttle throttle1 (.pixelclock(pixel_clock), .number({0,0,
throt[7:0]}),//(throttletest),
                                          .linecount(vcount),
.pixelcount(hcount), .RGB(throttleout1));
      defparam throttle1.cornerx = 600;
      defparam throttle1.cornery = 0;

      throttle throttle2 (.pixelclock(pixel_clock), .number({0,0,
throt[7:0]}),//(throttletest),
                                          .linecount(vcount),
.pixelcount(hcount), .RGB(throttleout2));
      defparam throttle2.cornerx = 0;
      defparam throttle2.cornery = 0;


      compass compass1(-1*direction, hcount, vcount, compassred,
compassgreen, compassblue);

//assign led  = 8'b00000000;//leds[7:0];


endmodule
```

```
//VGA Module

module vga(reset, pixclock,hcount,vcount,hsync,vsync,blank, vblank,
hblank);
   input pixclock, reset;
   output signed [11:0] hcount;
   output signed[11:0] vcount;
   output         vsync;
   output         hsync;
   output         blank;
      output                  vblank;
      output                  hblank;

   reg           hsync,vsync,hblank,vblank,blank;
   reg signed [11:0]          hcount;
   reg signed [11:0] vcount;


   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = (hcount == 639);
   assign    hsyncon = (hcount == 655);
   assign    hsyncoff = (hcount == 751);
   assign    hreset = (hcount == 799);

   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 479);
   assign    vsyncon = hreset & (vcount == 490);
   assign    vsyncoff = hreset & (vcount == 492);


   assign    vreset = hreset & (vcount == 523);

   wire      next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge pixclock) begin
```

```
                        if (reset) begin
                                    hcount <= 12'b0;
                                              vcount <= 12'b0;
                                              hsync <=  12'b1;
                                              vsync <=  12'b1;
                                              blank <=  1'b1;
                         end
                         else begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;

      blank <= next_vblank | (next_hblank & ~hreset);
                end
   end
endmodule


//Debouncer

module debounce (reset, pixel_clock, noisy, clean);
   input reset, pixel_clock, noisy;
   output clean;

   reg [19:0] count;
   reg new, clean;

   always @(posedge pixel_clock)
     if (reset) begin new <= noisy; clean <= noisy; count <= 0; end
     else if (noisy != new) begin new <= noisy; count <= 0; end
     else if (count == 650000) clean <= new;
     else count <= count+1;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    14:20:14 04/21/06
// Design Name:
// Module Name:    LiftFSM
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////
module LiftFSM(rho, velocity, alpha, liftforce);

      // L = .5 * Cl * r * V^2 * A

      // L  = Lift Force
      // Cl = Lift Coefficient
      // r  = air density
      // V  = Velocity of Plane
      // A  = Area of Wing

      // Cl is dependent on Angle of Attack

      // Cl = 2 * pi    * angle (in radians)
```

```
      // angle = Angle of Attack or Pitch

      // Cl increases with increasing Angle of Attack until a Stall
Angle
      // at which point the lift decreases dramatically.

      input signed [4:0] rho;
      input signed [17:0] velocity;
      input signed [7:0] alpha;

      output signed [16:0] liftforce;
      wire   signed [16:0] liftforce;

      wire signed [51:0] intermediate;

      parameter A = 1;

      assign intermediate = alpha * rho * velocity * velocity * A * 3 /
8;    //calculate equation

      assign liftforce = 3*intermediate[49:33];       // Divide out to
compensate for alpha

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
//
// Create Date:    15:53:06 05/09/06
// Design Name:
// Module Name:    pitch
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
////////
module pitch(pixelclock, speedenable, reset, upbutton, downbutton,
pitch);

      input speedenable;
      input pixelclock;

      input reset;

      input upbutton;
      input downbutton;
```

```
      output [9:0] pitch;
      reg    [9:0] pitch;

      always @ (posedge pixelclock)
      begin
            if (reset)  //reset position
                  pitch <= 0;
            else if (speedenable)
                        begin
                        if (upbutton)      //increment pitch
                              pitch <= pitch + 1;
                        else if (downbutton)     //decrement pitch
                              pitch <= pitch - 1;
                  end
      end



endmodule




`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    21:19:27 04/24/06
// Design Name:
// Module Name:    PitchOmega
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////
module PitchOmega(yokepitch, roll, vertical_omega, horizontal_omega);

            // Inputs:

            //          Yoke Pitch Angle
            //          Roll
```

```
            // Intermediates:

            //    Pitch Omega
            //          Yoke_Pitch_Angle * Constant

            // Outputs:

            //    Vertical Omega
            //          Pitch_Omega Cos[Roll]

            // Horizontal Omega
            //          Pitch_Omega Sin[Roll]

      input signed [9:0] yokepitch;
      input signed [17:0] roll;

      parameter yoke_pitch_constant = 32;

      wire signed [16:0] pitch_omega;

      assign pitch_omega = yokepitch * yoke_pitch_constant;

      output signed [16:0] vertical_omega;
      wire   signed [16:0] vertical_omega;
      output signed [16:0] horizontal_omega;
      wire   signed [16:0] horizontal_omega;
      SinTest sin_vert_omega( roll[17:8], pitch_omega,
horizontal_omega);
      CosTest cos_horz_omega( roll[17:8], pitch_omega, vertical_omega);

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
//
// Create Date:    19:15:09 05/08/06
// Design Name:
// Module Name:    PlanePhysics
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////
module PlanePhysics( reset, pixel_clock, frame_enable, throttle,
yoke_pitch, yoke_roll,
```

```
                                            altitude, velocity,
vertical_velocity, pitch, roll, direction, endgame//);
                                            //,rho, alpha,
totalvertforce, totalhorzforce, totaldirvelforce, del_velocity,
                                            // del_vert_velocity,
del_horz_velocity,
                                            /* dragforce,*/, thrustforce,
liftforce, weightforce);

      input reset;
      input pixel_clock;
      input frame_enable;

      output [1:0] endgame;
      wire   [1:0] endgame;

      // THROTTLE
      input [9:0] throttle;

      // YOKE
      input [9:0] yoke_pitch;
      input [9:0] yoke_roll;

      // VALUES
      output [15:0] altitude;
      output [9:0] velocity;
      output [9:0] vertical_velocity;
      output [9:0] pitch;
      output [9:0] roll;
      output [9:0] direction;

      wire [15:0] altitude;
      wire [9:0] velocity;
      wire [9:0] vertical_velocity;
      wire [9:0] pitch;
      wire [9:0] roll;
      wire [9:0] direction;

//    output [4:0] rho;
//    output [7:0] alpha;

      wire [4:0] rho;
      wire [7:0] alpha;

      // CALCULATING RHO
      Rho mod_rho( frame_enable, altitude, rho);

      // CALCULATING ALPHA
      Alpha1 mod_alpha( frame_enable, pitch, alpha);
/*
      output [16:0] totalvertforce;
      output [16:0] totalhorzforce;
      output [16:0] totaldirvelforce;
      output [17:0] del_velocity;
      output [17:0] del_vert_velocity;
      output [17:0] del_horz_velocity;
```

```
      wire [16:0] totalvertforce;
      wire [16:0] totalhorzforce;
      wire [16:0] totaldirvelforce;
      wire [17:0] del_velocity;
      wire [17:0] del_vert_velocity;
      wire [17:0] del_horz_velocity;
*/
      //output [16:0] dragforce;
      output [16:0] thrustforce;
      output [16:0] liftforce;
      output [16:0] weightforce;

      //wire [16:0] dragforce;
      wire [16:0] thrustforce;
      wire [16:0] liftforce;
      wire [16:0] weightforce;

      // PHYSICS
      FlightFSM physics( reset, pixel_clock, frame_enable, yoke_pitch,
yoke_roll, throttle, rho, alpha, altitude,
                                          velocity, vertical_velocity,
pitch, roll, direction, endgame//);
                                          /*,totalvertforce,
totalhorzforce, totaldirvelforce, del_velocity,
                                          del_vert_velocity,
del_horz_velocity,
                                          dragforce, */,thrustforce,
liftforce, weightforce);

endmodule




`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////
////////
// Engineer: Mariela Buchin
// Modified Last by Scott Fisher
// Module Name:    ratetoangle
/*
 AngleRate = K * (ADCVoltage-ZeroVoltage)
 Chose K = 1 for counterclockwise motion
 Chose K = 17/16 for clockwise motion
 K is some constant (Degs/sec/volt)
 Angle = Angle + AngleRate*deltaT
 Calibration for ZeroVoltage done on reset
*/
//////////////////////////////////////////////////////////////////////////
////////
module ratetoangle(clock, reset, enable, rate, angle);
input clock, reset, enable;
input [7:0] rate;
output signed [9:0] angle;
wire   signed [9:0] angle;
```

```
reg [13:0] intermediate;
reg signed [9:0] zero_rate;

wire signed [9:0] rate2;
assign rate2[9:8] = 0;    //converting to 9 bit rate
assign rate2[7:2] = rate[7:2];
assign rate2[1:0] = 0;    //gets rid of noise from the sensors

wire signed [13:0] del_angle;
assign del_angle = rate2 - zero_rate;

assign angle = intermediate[13:4]; //makes the increments look slower

always @ (posedge clock)
begin
      if (reset)
      begin
      intermediate <= 0;         //zeros the angle
      zero_rate <= rate2;           //saves the digital value for zero
movement
      end
      else if (enable) begin
            if ((del_angle < 0) && (angle > -505 - del_angle))
      //counterclockwise guard against going over max amount of bits
            intermediate <= intermediate + del_angle;
            else if ((del_angle > 0) && (angle < 505 - del_angle))
//clockwise
            intermediate <= intermediate + ((17 * del_angle) / 16);
      end
end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
//
// Create Date:    14:47:10 04/27/06
// Design Name:
// Module Name:    Rho
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////////
module Rho( clk, altitude, rho);

      // Input

      input clk;
      input [15:0] altitude;

      //wire [15:0] altitude;

      //assign altitude = alt[23:8];

      // Output

      output [4:0] rho;
      reg    [4:0] rho;


      always @ (posedge clk)
      begin
            if (altitude > 0 && altitude < 5000)
                  rho = 5;
            else if (altitude < 10000)
                  rho = 4;
            else if (altitude < 15000)
                  rho = 3;
            else if (altitude < 20000)
                  rho = 2;
            else if (altitude < 25000)
                  rho = 1;
            else if (altitude < 30000)
                  rho = 0;
            else
                  rho = 0;
      end
endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:    15:27:51 05/04/06
// Design Name:
// Module Name:    roll
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module roll(pixelclock, speedenable, reset, rightbutton, leftbutton,
roll);

      input speedenable;
      input pixelclock;

      input reset;

      input rightbutton;
      input leftbutton;

      output [9:0] roll;
      reg    [9:0] roll;

      always @ (posedge pixelclock)
      begin
            if (reset)  //reset position
                  roll <= 0;
            else if (speedenable)
                        begin
                        if (rightbutton)   //increment roll
                              roll <= roll + 1;
                        else if (leftbutton)     //decrement roll
                              roll <= roll - 1;
                  end
      end



endmodule







`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    21:32:27 04/24/06
// Design Name:
// Module Name:    RollOmega
// Project Name:
// Target Device:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////////
module RollOmega(yokeroll, roll_omega);

            // Inputs:

            //          Yoke Roll Angle

            // Outputs:

            //    Roll Omega
            //          Yoke_Roll_Angle * Constant

      input signed [9:0] yokeroll;

      parameter yoke_roll_constant = 32;

      output signed [16:0] roll_omega;
      wire   signed [16:0] roll_omega;

      assign roll_omega = yokeroll * yoke_roll_constant;

endmodule
/*********************************************************************
*********
*     This file is owned and controlled by Xilinx and must be used
*
*     solely for design, simulation, implementation and creation of
*
*     design files limited to Xilinx devices or technologies. Use
*
*     with non-Xilinx devices or technologies is expressly prohibited
*
*     and immediately terminates your license.
*
*
*
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
*
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR
*
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION
*
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION
*
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS
*
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
*
```

```
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
*
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY
*
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
*
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
*
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
*
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*
*     FOR A PARTICULAR PURPOSE.
*
*
*
*     Xilinx products are not intended for use in life support
*
*     appliances, devices, or systems. Use in such applications are
*
*     expressly prohibited.
*
*
*
*     (c) Copyright 1995-2004 Xilinx, Inc.
*
*     All rights reserved.
*
**********************************************************************
********/
// The synopsys directives "translate_off/translate_on" specified below
are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity
synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file rom.v when simulating
// the core, rom. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module rom(
      addr,
      clk,
      dout);


input [9 : 0] addr;
input clk;
output [7 : 0] dout;

// synopsys translate_off

      BLKMEMSP_V6_1 #(
            10,   // c_addr_width
```

```
            "0",  // c_default_data
            1024, // c_depth
            0,    // c_enable_rlocs
            0,    // c_has_default_data
            0,    // c_has_din
            0,    // c_has_en
            0,    // c_has_limit_data_pitch
            0,    // c_has_nd
            0,    // c_has_rdy
            0,    // c_has_rfd
            0,    // c_has_sinit
            0,    // c_has_we
            18,   // c_limit_data_pitch
            "rom.mif",  // c_mem_init_file
            0,    // c_pipe_stages
            0,    // c_reg_inputs
            "0",  // c_sinit_value
            8,    // c_width
            0,    // c_write_mode
            "0",  // c_ybottom_addr
            1,    // c_yclk_is_rising
            1,    // c_yen_is_high
            "hierarchy1",     // c_yhierarchy
            0,    // c_ymake_bmm
            "16kx1",    // c_yprimitive_type
            1,    // c_ysinit_is_high
            "1024",     // c_ytop_addr
            0,    // c_yuse_single_primitive
            1,    // c_ywe_is_high
            1)    // c_yydisable_warnings
      inst (
            .ADDR(addr),
            .CLK(clk),
            .DOUT(dout),
            .DIN(),
            .EN(),
            .ND(),
            .RFD(),
            .RDY(),
            .SINIT(),
            .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of rom is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of rom is "black_box"

endmodule

`timescale 1ns / 1ps
```

```
//////////////////////////////////////////////////////////////////////
////////
// Company: 6.111
// Engineer:Mariela Buchin
// Module Name:    sensors
//////////////////////////////////////////////////////////////////////
////////
module sensors(clock_27mhz, pixel_clock, newframe, button0, button1,
user1, user3, user4, led, throttle_sync2, pitch, roll);
input newframe;
input clock_27mhz, pixel_clock, button0, button1;
inout [31:0] user1, user3, user4;
output [9:0] throttle_sync2;
output [9:0] pitch, roll;
output [7:0] led;
wire [7:0] led;
//Interfacing the Analog Input to the Labkit through the AD670
Converter
//---------------------------------------------------------------------
----
      //Inputs
      wire reset_sync;
      wire [7:0] prateanalog, pitchrate;
      wire [9:0] roll, pitch;
      wire [7:0] rrateanalog, rollrate;
      wire [7:0] tanalog, throttle;
      reg [7:0] pitchrate_sync, pitchrate_sync2;
      reg [7:0] rollrate_sync, rollrate_sync2;
      reg [7:0] throttle_sync, throttle_sync2;
      wire state;   //for logic analyzer

      //Outputs
      wire read1, read3, read4, status1, status3, status4;

      assign user1[7:0] = 8'hz;    //assign to high z
      assign user3[7:0] = 8'hz;    //assign to high z
      assign user4[7:0] = 8'hz;    //assign to high z
```

```
      assign user1[9] = 1'hz;       //ditto
      assign user3[9] = 1'hz;       //ditto
      assign user4[9] = 1'hz;       //ditto

      assign prateanalog = user1[7:0];
      assign rrateanalog = user3[7:0];
      assign tanalog = user4[7:0];

      assign status1 = user1[9];      //Purple wire     pitch
      assign status3 = user3[9];      //Purple wire     roll
      assign status4 = user4[9];      //Purple wire  throttle

      assign user1[8] = read1;   //Blue wire     pitch
      assign user3[8] = read3;   //Blue wire     roll
      assign user4[8] = read4;   //Blue wire   throttle

      wire button_press, button_press2;
      assign user4[10] = 1'hz;
      assign button_press = user4[10];
      assign user1[31:10] = 22'b0;  //assign the rest of user1
      assign user3[31:10] = 22'b0;  //assign the rest of user3
      assign user4[31:11] = 22'b0;  //assign the rest of user4

      assign led = ~pitchrate;  //for debugging purposes

//Reset and Button_press Debounce====================================
      debounce reset(1'b1, clock_27mhz, ~button0, reset_sync);
//Negating the Button Here
      debounce sampling(1'b1, clock_27mhz, ~button_press ,
button_press2);       //Negating the Button Here
//Button_press is the button on the vest that is used to zero the angle
and reset the zero voltage

//=================================================================
//  Making a 3.375 MHz clock=======================================
//=================================================================
reg [2:0] count;
wire conclk, convert_clock;

BUFG convert_clock_buf (.I(conclk), .O(convert_clock));
always @ (posedge clock_27mhz)
 begin
      if(reset_sync) count <= 0;
      else count <= count + 1;
 end

assign conclk = count [2];     //this is the 3.375 MHz clock

////////////////////////////////////////////////////////////////
//Instantiate Convertor for each analog input
////////////////////////////////////////////////////////////////
AD670 digPitch(convert_clock, reset_sync, newframe, status1,
prateanalog, read1, pitchrate);
AD670 digRoll(convert_clock, reset_sync, newframe, status3,
rrateanalog, read3, rollrate);
```

```
AD670 digThrot(convert_clock, reset_sync, newframe, status4, tanalog,
read4, throttle);
//=================================================================
==================
//Syncing the output of the Converter with the 27mhz clock
//=================================================================
==================
always @ (posedge pixel_clock)
begin
      pitchrate_sync <= pitchrate;
      pitchrate_sync2 <= pitchrate_sync;
end
//-----------------------------------------------------------------
-----------------
always @ (posedge pixel_clock)
begin
      rollrate_sync <= rollrate;
      rollrate_sync2 <= rollrate_sync;
end
//-----------------------------------------------------------------
-----------------
always @ (posedge pixel_clock)
begin
      throttle_sync <= throttle;
      throttle_sync2 <= throttle_sync;
end
//-----------------------------------------------------------------
-----------------
//Converting Angular Rates to Angles
ratetoangle rolly(pixel_clock, (reset_sync ||  button_press2 ||
button1), newframe, rollrate_sync2, roll);
ratetoangle pitcher(pixel_clock, (reset_sync || button_press2 ||
button1), newframe, pitchrate_sync2, pitch);
//-----------------------------------------------------------------
-----------------
//Used for debugging the AD convertor input/output
assign analyzer4_data = {8'h0, convert_clock, newframe, status4, state,
read4};
assign analyzer4_clock = 1'b1;

endmodule
`timescale 1ns / 1ps
```

```
module SinTest(theta, value_in, value_out);

      input [9:0] theta;
      input signed [16:0] value_in;

      output signed [16:0] value_out;
      wire   signed [16:0] value_out;

      wire signed [8:0] cosout;

      sin sin12( theta, cosout);

      wire signed [25:0] intermediate;

      assign intermediate = value_in * cosout;

      assign value_out = intermediate [23:7];

endmodule
`timescale 1ns / 1ps
```

```
module throttle(pixelclock, number, linecount, pixelcount, RGB);
input [9:0] number;
input pixelclock;
input [11:0] linecount, pixelcount;
output reg [23:0] RGB;

wire [16:0] intermediate;
wire [8:0] final_num;

assign intermediate = number[7:0] * 480;
assign final_num = intermediate[16:8];

parameter cornerx = 600;
parameter cornery = 0;
parameter width   =  40;
parameter height  = 480;

always @ (pixelcount or linecount)
begin
 if ((cornerx<=pixelcount)&&(pixelcount<cornerx+width)
&&(cornery<=linecount)&& (linecount<cornery+height))
                  begin
                //if (linecount <= )
                   if (linecount <= (cornery + (height - final_num)))
                             RGB = 24'hD10000;
                   else RGB = 24'h05EA00;
                  end
 else RGB = 24'h000000;
end

endmodule
```

```
`timescale 1ns / 1ps



module throttle_value(pixelclock, speedenable, reset, upbutton,
downbutton, throttle);

      input speedenable;
      input pixelclock;

      input reset;

      input upbutton;
      input downbutton;

      output [9:0] throttle;
      reg    [9:0] throttle;

      always @ (posedge pixelclock)
      begin
            if (reset)  //reset position
                  throttle <= 0;
            else if (speedenable)
                        begin
                        if (upbutton)      //increment pitch
                              throttle <= throttle + 1;
                        else if (downbutton)     //decrement pitch
                              throttle <= throttle - 1;
                  end
      end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    14:20:37 04/21/06
// Design Name:
module ThrustFSM(rho, throttle, thrustforce);

            // F = m * a ( * r)

            // F = Thrust Force
            // m = Mass of Fuel per Impulse
            // a = Acceleration provided by Engine
            // r = Air Density

            // Inputs

            input [4:0] rho;
            input [9:0] throttle;     // Throttle between 0 and 100 to
denote percentage of thrust

            // Constants

            parameter mass = 1;           // Mass of Fuel
(Preliminarily Equal to Plane)
            parameter a = 1;        // Constant to multiply the value
of Throttle

            output [16:0] thrustforce;
```

```
            wire   [16:0] thrustforce;

            //
            assign thrustforce = mass * (6 - rho) * throttle * a / 8;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////
// Company:  Bon Voyage
// Engineer: Fishy
//
// Create Date:    20:28:07 04/24/06
// Design Name:
// Module Name:    VertHorzForces
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////
module VertHorzForces(dragforce_dir, dragforce_horz, dragforce_vert,
thrustforce, liftforce, weightforce, pitch, roll, tvf, thf, tdv);

            // Inputs:

            // Drag     (dir, horz, vert)
            // Thrust
            // Lift
```

```
            // Weight

            // Pitch
            // Roll

            // Intermediate Values:

            // Force in Velocity Direction
            //          TD: Thrust - Drag
            // Vertical TD Force
            //          VTD: TD Sin(Pitch)

            // Components of Lift Force
            // Vertical Lift Force
            //          VLF: Lift Cos(Pitch) Cos(Roll)
            // Horizontal Lift Force
            //          HLF: Lift Sin(Roll)

            // Outputs:

            // Total Vertical Force
            //          VLF + VTD - Weight

            // Total Horizontal Yaw Force
            //          HLF

            // Total Force in Direction of Velocity
            //          TD

            input signed [16:0] dragforce_dir;
            input signed [16:0] dragforce_horz;
            input signed [16:0] dragforce_vert;
            input signed [16:0] thrustforce;
            input signed [16:0] liftforce;
            input [16:0] weightforce;

            input signed [17:0] pitch;
            input signed [17:0] roll;

            wire signed [16:0] td_dir;          // Thrust Drag
            wire signed [16:0] td_vert;   // Thrust Drag
            wire signed [16:0] vtd;
            wire signed [16:0] vlf;
            wire signed [16:0] hlf;

            // Temporary
            wire signed [16:0] thrustforce_dir;
            wire signed [16:0] thrustforce_vert;

            // Vertical Thrust Component = Sin(pitch) * thrustforce
            SinTest sin_vtc(pitch[17:8], thrustforce,
thrustforce_vert);
            // Horzintal Thrust Component = Cos(pitch) * thrustforce
            CosTest cos_htc(pitch[17:8], thrustforce, thrustforce_dir);

            // TD (Thrust/Drag) Direction
            assign td_dir = thrustforce_dir - dragforce_dir;
```

```
            assign td_vert = thrustforce_vert - dragforce_vert;

            // VTD = Sin(Pitch) * TD
                        // No Longer Needed
            //SinTest sin_vtd(pitch[17:8], td, vtd);
       // No Longer Needed

            // VLF = Liftforce * Cos(Pitch) * Cos(Roll)
                  // VLF_tmp = Liftforce * Cos(Pitch)         // Pitch
Already Taken Care of
            //CosTest cos_vlftmp(pitch[17:8], liftforce, vlf_tmp);
                  // VLF = VLF_tmp * Cos(Roll)
            CosTest cos_vlf(roll[17:8], liftforce, vlf);

            // HLF = Sin(Roll) * Liftforce
            SinTest sin_hlf(roll[17:8], liftforce, hlf);

            output signed [16:0] tvf;           // TOTAL VERTICAL FORCE
            wire   signed [16:0] tvf;
            output signed [16:0] thf;           // TOTAL HORIZONTAL
FORCE
            wire   signed [16:0] thf;
            output signed [16:0] tdv;           // TOTAL DIRECTIONAL
FORCE
            wire   signed [16:0] tdv;

            assign tvf = vlf + td_vert - weightforce;
            assign thf = hlf - dragforce_horz;
            assign tdv = td_dir;

endmodule
`timescale 1ns / 1ps

module vga_test_v;

//Inputs
reg clk;
reg reset;


//Outputs
wire hsync;
wire vsync;
wire [11:0] hcount, vcount;
wire hblank;
wire vblank;
wire blank;

vga uut (
       .reset(reset),
    .pixclock(clk),
    .hcount(hcount),
    .vcount(vcount),
       .hsync(hsync),
    .vsync(vsync),
       .blank (blank),
    .vblank(vblank),
```

```
    .hblank(hblank));
always #0.065 clk = ~clk;

initial begin
clk=0;
reset=0;
#500;
reset=1;
#500;
reset=0;
end
endmodule
```