3D Wireless Mouse
CODE APPENDIX

Shirley Li, Matthew Tanwanteng, and Joseph Cheng
TA: Charlie Kehoe
6.111 Introductory Digitial Systems Laboratory

Labkit Code (Verilog)

Labkit Top-Level Code

```
//////////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////
////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is
an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated
into
//    the data bus, and the byte write enables have been combined into
the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is
now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////
////////
//
// Complete change history (including bug fixes)
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb
devices
//              actually populated on the boards. (The boards support
up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Apr-29: Change history started
```

```
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb
devices
//              actually populated on the boards. (The boards support
up to
//              72Mb devices, with 21 address lines.)
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a
default
//              value. (Previous versions of this file declared this
port to
//              be an input.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
////////////////////////////////////////////////////////////////////////
////////

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

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

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

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

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

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

               clock_feedback_out, clock_feedback_in,

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

               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

               mouse_clock, mouse_data, keyboard_clock, keyboard_data,

               clock_27mhz, clock1, clock2,

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

               button0, button1, button2, button3, button_enter,
```

```
button_right,
             button_left, button_down, button_up,

             switch,

             led,

             user1, user2, user3, user4,

             daughtercard,

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

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

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

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

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

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

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

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

   input  clock_feedback_in;
   output clock_feedback_out;

   inout  [15:0] flash_data;
   output [23:0] flash_address;
```

```
   output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b,
flash_byte_b;
   input  flash_sts;

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

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

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

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

   inout [43:0] daughtercard;

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

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

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

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

   // VGA Output
   assign vga_out_red = 10'h0;
   assign vga_out_green = 10'h0;
   assign vga_out_blue = 10'h0;
   assign vga_out_sync_b = 1'b1;
   assign vga_out_blank_b = 1'b1;
   assign vga_out_pixel_clock = 1'b0;
   assign vga_out_hsync = 1'b0;
   assign vga_out_vsync = 1'b0;
```

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

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

   // SRAMs
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_clk = 1'b0;
   assign ram0_cen_b = 1'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

   // 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


   //------------------------------------------------------------
   //------------------------------------------------------------
   // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are
inputs
   // button0, button1, button2, button3, button_enter, button_right,
   // button_left, button_down, button_up, and switches are inputs

   wire reset, reset_sync, start, busy, r_wbar, cs_bar, status,
data_ready_ad, data_ready_si;
   wire init_reset, init_out;
   wire load_reg, load_init;
   wire reset_vel;
   wire load_vel, reset_pos, load_pos;
   wire load_filter, init_state;

   //x-axis
   wire[7:0] acc_x_in, acc_x_in_si, acc_x_init, acc_x_out, acc_x_reg;
   wire[15:0] vel_x;
   wire[23:0] pos_x;
   //y-axis
   wire[7:0] acc_y_in, acc_y_in_si, acc_y_init, acc_y_out, acc_y_reg;
   wire[15:0] vel_y;
   wire[23:0] pos_y;
   //z-axis
   wire[7:0] acc_z_in, acc_z_in_si, acc_z_init, acc_z_out, acc_z_reg;
   wire[15:0] vel_z;
   wire[23:0] pos_z;

   Init_Register InitReg(.clk(clock_27mhz), .init_reset(init_reset),
   .reset(reset_sync), .init_out(init_out));

   Syncrhonizer Sync(.clk(clock_27mhz), .reset(reset), .reset_sync
(reset_sync));

   MajorFSM MFSM(.clk(clock_27mhz), .reset(reset_sync), .data_ready
(data_ready),
   .init_reset(init_reset), .init_out(init_out), .load_init(load_init),
   .load_reg(load_reg), .load_filter(load_filter), .load_int_1
(load_vel),
   .load_int_2(load_pos), .init_state(init_state));

   //----------------------------------------
   //WITH RS-232 INTERFACE-------------------
   //----------------------------------------
   assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;
```

```
   wire load_si;

   Serial_Interface SI(.clk(clock_27mhz), .load(load_si), .rxd
(rs232_rxd), .acc_x(acc_z_in_si),
   .acc_y(acc_x_in_si), .acc_z(acc_y_in_si));

   FSM_Serial FSMS(.clk(clock_27mhz), .reset(reset_sync), .load_data
(load_si),
   .rxd(rs232_rxd), .data_ready(data_ready_si));


   //----------------------------------------
   //WITH AD Converter-----------------------
   //----------------------------------------
   FSM_AD FAD(.clk(clock_27mhz), .reset(reset_sync), .start(start),
   .r_wbar(r_wbar), .cs_bar(cs_bar), .status(status), .le_adc
(data_ready_ad));

   Divider Div(.clk(clock_27mhz), .reset(reset_sync), .enable(start));

   //----------------------------------------
   //CALCULATIONS----------------------------
   //----------------------------------------
   //x-axis
   Acceration_Register AccRegX(.clk(clock_27mhz), .load(data_ready),
   .load_reg(load_reg), .load_init(load_init), .acc_in(acc_x_in), .
acc_reg(acc_x_reg),
   .acc_init(acc_x_init), .acc_out(acc_x_out));

   Db_Integrator DbIntX(.clk(clock_27mhz), .acc(acc_x_out),
   .reset_vel(reset_vel), .load_vel(load_vel), .vel(vel_x),
   .reset_pos(reset_pos), .load_pos(load_pos), .pos(pos_x));

   //y-axis
   Acceration_Register AccRegY(.clk(clock_27mhz), .load(data_ready),
   .load_reg(load_reg), .load_init(load_init), .acc_in(acc_y_in), .
acc_reg(acc_y_reg),
   .acc_init(acc_y_init), .acc_out(acc_y_out));

   Db_Integrator DbIntY(.clk(clock_27mhz), .acc(acc_y_out),
   .reset_vel(reset_vel), .load_vel(load_vel), .vel(vel_y),
   .reset_pos(reset_pos), .load_pos(load_pos), .pos(pos_y));

   //z-axis
   Acceration_Register AccRegZ(.clk(clock_27mhz), .load(data_ready),
   .load_reg(load_reg), .load_init(load_init), .acc_in(acc_z_in), .
acc_reg(acc_z_reg),
   .acc_init(acc_z_init), .acc_out(acc_z_out));

   Db_Integrator DbIntZ(.clk(clock_27mhz), .acc(acc_z_out),
   .reset_vel(reset_vel), .load_vel(load_vel), .vel(vel_z),
   .reset_pos(reset_pos), .load_pos(load_pos), .pos(pos_z));

   //----------------------------------------
   //USER INTERFACE--------------------------
   //----------------------------------------
   assign reset = button0;
```

```
   assign reset_vel = button1;
   assign reset_pos = button2;

   assign   user1[0] = r_wbar;
   assign user1[1] = cs_bar;
   assign status = user1[2];

   assign acc_x_in = ~switch[0]? user1[10:3]:acc_x_in_si;
   assign acc_y_in = ~switch[0]? user1[18:11]:acc_y_in_si;
   assign acc_z_in = ~switch[0]? user3[7:0]:acc_z_in_si;
   assign data_ready = ~switch[0]? data_ready_ad:data_ready_si;

   wire[8:0] dx, dy, z;
   wire lmb, mmb, rmb, clk, data;

   // user input
      assign dy = button_up ? (button_down ? 0 : -5) :(button_down? 5:
0);
      assign dx = button_left ? (button_right? 0 : 5) : (button_right?
-5 : 0);
      assign z = 0;
      assign mmb = ~button2;
      assign lmb = ~button_enter;
      assign rmb = ~button3;

   //---------------------------------------
   //DEBUGGING------------------------------
   //---------------------------------------
   wire[639:0]dots;
   wire[7:0] led_out;
   assign led = ~led_out;

   Display_State DS(.clk(clock_27mhz), .switch(switch), .led
(led_out), .dots(dots), .init_state(init_state),
      .acc_x_reg(acc_x_reg), .acc_x_init(acc_x_init), .acc_x_out
(acc_x_out), .vel_x(vel_x), .pos_x(pos_x),
      .acc_y_reg(acc_y_reg), .acc_y_init(acc_y_init), .acc_y_out
(acc_y_out), .vel_y(vel_y), .pos_y(pos_y),
      .acc_z_reg(acc_z_reg), .acc_z_init(acc_z_init), .acc_z_out
(acc_z_out), .vel_z(vel_z), .pos_z(pos_z));

   display disp (~button0, clock_27mhz, disp_blank, disp_clock, disp_rs,
            disp_ce_b, disp_reset_b, disp_data_out, dots);


      //---------------------------------------
      //PS/2 INTERFACE-------------------------
      //---------------------------------------

      wire clkoe;
     wire dataoe;
      wire indata;
      wire inclk;
      wire outclock;          // output of clock gen
      wire odata;

      wire [3:0] state;
      wire update;
```

```
wire oupdate;
wire int_ld;
wire done;
wire restart;
wire [7:0] request;
reg [6:0] counter;
reg slowclk;
reg iclk;
wire [9:0] store;

wire [3:0] cstate;
wire enable, sw;
assign reset = button0;


assign user2[2] = outclock;
assign user2[3] = odata;
assign user2[10:4] = 7'b1111111;

assign user2[0] = clkoe ? outclock : 1'bz;  // clock
assign inclk = user2[0];

assign user2[1] = dataoe ? odata : 1'bz;
assign indata = user2[1];

assign analyzer1_data[7:0] = request;
assign analyzer3_data[7:0] = store[9:2];
assign analyzer4_data[0] = int_ld;
assign analyzer4_data[1] = user2[0];
assign analyzer4_data[2] = user2[1];
assign analyzer4_data[3]      = oupdate;
assign analyzer4_data[6:4] = {rs232_rxd, load_si, data_ready};
assign analyzer2_data[0] = int_ld;
assign analyzer2_data[1] = update;
assign analyzer2_data[2] = outclock;
assign analyzer2_data[3] = dataoe;
assign analyzer2_data[7:4] = cstate;
assign analyzer1_clock = clock_27mhz;
assign analyzer2_clock = clock_27mhz;
assign analyzer4_clock = clock_27mhz;

always @ (posedge clock_27mhz or negedge reset_sync) begin
      if(!reset_sync) begin
            counter <= 7'b0;
            slowclk <= 0;
      end
      else if(counter == 99) begin
            counter <= 7'b0;
            slowclk <= ~slowclk;
      end
      else begin
            counter <= counter + 1;
            slowclk <= slowclk;
      end

end

always @ (posedge slowclk or negedge reset_sync) begin
```

```
            if(!reset_sync) begin
                  iclk <= 0;
            end
            else begin
                  iclk <= inclk;
            end
      end

      wire[8:0] dx_final, dy_final, z_final;
      assign dx_final = switch[1]? dx //: vel_x[9:1];
              :{acc_x_out[7], acc_x_out[7], acc_x_out[7], acc_x_out
[7:2]};//vel_x[9:1];
      assign dy_final = switch[1]? dy: {acc_y_out[7], acc_y_out[7],
acc_y_out[7], acc_y_out[7:2]};
      assign z_final = switch[1]? z: vel_z[9:1];

      ps2interface2 ps2i(.sysclk(slowclk), .reset(!reset_sync), .dx
(dx_final), .dy(dy_final), .z(z_final), .lmb(lmb), .mmb(mmb),
      .rmb(rmb), .iclk(inclk), .idata(indata), .oclk(outclock), .
odata_sync(odata), .clkoe(clkoe),
      .dataoe(dataoe), .state(state), .mupdate(update), .oupdate
(oupdate), .int_ld(int_ld), .done_sync(done),
      .cstate(cstate), .enable(enable), .switch(sw), .request
(request), .restart(restart), .store(store));

endmodule
```

Display Interface Code

```
//////////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- Alphanumeric Display Interface
//
//
// Created: November 5, 2003
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////
////////

module display (reset, clock_27mhz,
            disp_blank, disp_clock, disp_rs, disp_ce_b,
            disp_reset_b, disp_data_out, dots);

   input  reset, clock_27mhz;
   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
        disp_reset_b;
   input [639:0] dots;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   //////////////////////////////////////////////////////////////////////
////////
   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //
   //////////////////////////////////////////////////////////////////////
////////

   reg [4:0] count;
   reg [7:0] reset_count;
   reg clock;
   wire dreset;

   always @(posedge clock_27mhz)
     begin
      if (reset)
        begin
           count = 0;
           clock = 0;
        end
      else if (count == 26)
        begin
           clock = ~clock;
           count = 5'h00;
        end
      else
        count = count+1;
     end

   always @(posedge clock_27mhz)
     if (reset)
```

```
      reset_count <= 100;
    else
      reset_count <= (reset_count==0) ? 0 : reset_count-1;

  assign dreset = (reset_count != 0);

  assign disp_clock = ~clock;

  ////////////////////////////////////////////////////////////////////
////////
  //
  // Display State Machine
  //
  ////////////////////////////////////////////////////////////////////
////////

  reg [7:0] state;
  reg [9:0] dot_index;
  reg [31:0] control;

  assign disp_blank = 1'b0; // low <= not blanked

  always @(posedge clock)
    if (dreset)
      begin
       state <= 0;
       dot_index <= 0;
       control <= 32'h7F7F7F7F;
      end
    else
      casex (state)
      8'h00:
        begin
           // Reset displays
           disp_data_out <= 1'b0;
           disp_rs <= 1'b0; // dot register
           disp_ce_b <= 1'b1;
           disp_reset_b <= 1'b0;
           dot_index <= 0;
           state <= state+1;
        end

      8'h01:
        begin
           // End reset
           disp_reset_b <= 1'b1;
           state <= state+1;
        end

      8'h02:
        begin
           // Initialize dot register
           disp_ce_b <= 1'b0;
           disp_data_out <= 1'b0; // dot_index[0];
           if (dot_index == 639)
           state <= state+1;
           else
           dot_index <= dot_index+1;
```

```
        end

      8'h03:
        begin
           // Latch dot data
           disp_ce_b <= 1'b1;
           dot_index <= 31;
           state <= state+1;
        end

      8'h04:
        begin
           // Setup the control register
           disp_rs <= 1'b1; // Select the control register
           disp_ce_b <= 1'b0;
           disp_data_out <= control[31];
           control <= {control[30:0], 1'b0};
           if (dot_index == 0)
           state <= state+1;
           else
           dot_index <= dot_index-1;
        end

      8'h05:
        begin
           // Latch the control register data
           disp_ce_b <= 1'b1;
           dot_index <= 639;
           state <= state+1;
        end

      8'h06:
        begin
           // Load the user's dot data into the dot register
           disp_rs <= 1'b0; // Select the dot register
           disp_ce_b <= 1'b0;
           disp_data_out <= dots[dot_index];
           if (dot_index == 0)
           state <= 5;
           else
           dot_index <= dot_index-1;
        end
      endcase

endmodule
```

Accelerometer Code (Verilog)

Acceleration Registers

```
module Acceration_Register(clk, load, load_reg, load_init, acc_in,
acc_reg, acc_init, acc_out);
    //acc_in is in offset binary
    //acc_initial, acc_out is in twos complement
    //acc_initial are negative numbers

    input clk, load, load_reg, load_init;
    input[7:0] acc_in;

    output[7:0] acc_init, acc_out, acc_reg;
    reg[7:0] acc_init, acc_out, acc_reg;

    reg[7:0] acc_reg1, acc_reg2, acc_reg3, acc_reg4;
    reg[7:0] acc_reg5, acc_reg6, acc_reg7, acc_reg8;
    reg[7:0] acc_reg9, acc_reg10, acc_reg11, acc_reg12;
    reg[7:0] acc_reg13, acc_reg14, acc_reg15, acc_reg16;
    reg[11:0] acc_sum;


    always @(posedge clk)
    begin
        if(load)
         begin
            acc_reg16 <= acc_reg15;
            acc_reg15 <= acc_reg14;
            acc_reg14 <= acc_reg13;
            acc_reg13 <= acc_reg12;
            acc_reg12 <= acc_reg11;
            acc_reg11 <= acc_reg10;
            acc_reg10 <= acc_reg9;
            acc_reg9 <= acc_reg8;
            acc_reg8 <= acc_reg7;
            acc_reg7 <= acc_reg6;
            acc_reg6 <= acc_reg5;
            acc_reg5 <= acc_reg4;
            acc_reg4 <= acc_reg3;
            acc_reg3 <= acc_reg2;
            acc_reg2 <= acc_reg1;
            acc_reg1 <= acc_in;
         end
        if(load_init)
        begin
           acc_init <= {acc_reg[7],~acc_reg[6:0]}+8'b00000001;
           acc_out <= acc_out;
        end
        else if(load_reg)
        begin
            acc_init <= acc_init;
            acc_out <= {~acc_reg[7], acc_reg[6:0]} + acc_init;
        end
        else
        begin
            acc_init <= acc_init;
```

```
            acc_out <= acc_out;
        end
    end

    always @(load or load_init or load_reg or acc_reg1 or acc_reg2 or
acc_reg3 or acc_reg4)
    begin
            acc_sum = acc_reg1 + acc_reg2 + acc_reg3 + acc_reg4;
                    /*+ acc_reg5 + acc_reg6 + acc_reg7 + acc_reg8
                     + acc_reg9 + acc_reg10 + acc_reg11 + acc_reg12
                     + acc_reg13 + acc_reg14 + acc_reg15 + acc_reg16;*/
            acc_reg = acc_sum[9:2];
    end
endmodule
```

Integrator

```
//all data is in 2s complement
module Db_Integrator(clk, acc, reset_vel, load_vel, vel, reset_pos,
load_pos, pos);

    input clk;
    input reset_vel, load_vel;
    input reset_pos, load_pos;
    input[7:0] acc;

    output[15:0] vel;
    output[23:0] pos;

    reg[15:0] vel, sum_vel;
    wire[15:0] decayed_vel;
    reg[23:0] pos, sum_pos;
    wire[31:0] product;

    Multiplier M(.X(vel), .Y(16'd15), .P(product));

    always @(posedge clk)
    begin
        if(!reset_vel) vel <= 0;
        else if(load_vel) vel <= sum_vel;
        else vel <= vel;

        if(!reset_pos) pos <= 0;
        else if(load_pos) pos <= sum_pos;
        else pos <= pos;
    end

    always @(acc or reset_vel or load_vel or vel or load_pos or
reset_pos or pos)
    begin
        sum_vel = //product[19:4]
                      vel
                    + /*((acc < 8'd1)
                       &&(acc > 8'b11111110))? 16'd0:*/
                        {acc[7], acc[7], acc[7], acc[7],
                         acc[7], acc[7], acc[7], acc[7],
                         acc[7], acc[7], acc[7], acc[7:3]};
                         //acc[7], acc[7], acc[7], acc[7], acc[7], acc
[7:5]};
        sum_pos = pos
                    + /*((vel <16'b0000000000001111)
                      && (vel >16'b1111111111110001))?16'd0:  */
                        {vel[14], vel[14], vel[14], vel[14],
                       vel[14], vel[14], vel[14], vel[14], vel};
    end

endmodule
```

Input Graphical Display Code

```
module Display_Hex(clk,
num1,num2,num3,num4,num5,num6,num7,num8,num9,num10,
                          num11,num12,num13,num14,num15, num16, dots);
      input clk;
    input[3:0] num1, num2, num3, num4,
                        num5, num6, num7, num8,
                        num9, num10, num11, num12,
                        num13, num14, num15, num16;
    output[639:0] dots;
    //reg[639:0] dots;

    wire[39:0] dot1, dot2, dot3, dot4, dot5, dot6, dot7, dot8,
            dot9, dot10, dot11, dot12, dot13, dot14, dot15, dot16;


    Hex_to_Dots HD1(clk, num1, dot1);
    Hex_to_Dots HD2(clk, num2, dot2);
    Hex_to_Dots HD3(clk, num3, dot3);
    Hex_to_Dots HD4(clk, num4, dot4);
    Hex_to_Dots HD5(clk, num5, dot5);
    Hex_to_Dots HD6(clk, num6, dot6);
    Hex_to_Dots HD7(clk, num7, dot7);
    Hex_to_Dots HD8(clk, num8, dot8);
    Hex_to_Dots HD9(clk, num9, dot9);
    Hex_to_Dots HD10(clk, num10, dot10);
    Hex_to_Dots HD11(clk, num11, dot11);
    Hex_to_Dots HD12(clk, num12, dot12);
    Hex_to_Dots HD13(clk, num13, dot13);
    Hex_to_Dots HD14(clk, num14, dot14);
    Hex_to_Dots HD15(clk, num15, dot15);
    Hex_to_Dots HD16(clk, num16, dot16);

    assign dots = {dot16, dot15, dot14, dot13,
                        dot12, dot11, dot10, dot9,
                        dot8, dot7, dot6, dot5, dot5,
                        dot4, dot3, dot2, dot1};
endmodule

module Hex_to_Dots(clk, hex, dots);
      input clk;
      input[3:0] hex;
      output[39:0] dots;
      reg[39:0] dots;

      always @(posedge clk)
      begin
      case(hex)
            4'b0000:
dots=40'b00111110_01000001_01000001_01000001_00111110;
                                    //0
            4'b0001:
dots=40'b00000000_00000000_01000010_01111111_01000000; //1
            4'b0010:
dots=40'b01000110_01100001_01010001_01010001_01001110; //2
            4'b0011:
dots=40'b01100001_01001101_01010101_01010101_01100011; //3
```

```
            4'b0100:
dots=40'b00001110_00001000_00001000_00001000_01111111; //4
            4'b0101:
dots=40'b01100111_01001001_01001001_01001001_01110011; //5
            4'b0110:
dots=40'b00111111_01000101_01000101_00010001_00001110; //6
            4'b0111:
dots=40'b01100011_00010001_00001001_01000101_00000011; //7
            4'b1000:
dots=40'b00111110_01000101_01000101_01000101_00111110; //8
            4'b1001:
dots=40'b00100110_01001001_01001001_01001001_00111110; //9
            4'b1010:
dots=40'b01111000_00010110_00010001_00010110_01111000; //A
            4'b1011:
dots=40'b01111111_01001001_01001001_01001001_00110110; //B
            4'b1100:
dots=40'b00111110_01000001_01000001_01000001_00100010; //C
            4'b1101:
dots=40'b01111111_01000001_01000001_00100010_00011100; //D
            4'b1110:
dots=40'b01111111_01001001_01001001_01001001_01101011; //E
            4'b1111:
dots=40'b01111111_01000101_00000101_00000001_00000011; //F
      endcase
      end
endmodule
```

State Display

```
module Display_State(clk, switch, led, dots, init_state,
      acc_x_reg, acc_x_init, acc_x_out, vel_x, pos_x,
      acc_y_reg, acc_y_init, acc_y_out, vel_y, pos_y,
      acc_z_reg, acc_z_init, acc_z_out, vel_z, pos_z);

      input clk;
      input init_state;
      input[7:0] switch;

      //data_input
      input[7:0] acc_x_reg, acc_x_init, acc_x_out;
      input[15:0] vel_x;
      input[23:0] pos_x;
      input[7:0] acc_y_reg, acc_y_init, acc_y_out;
      input[15:0] vel_y;
      input[23:0] pos_y;
      input[7:0] acc_z_reg, acc_z_init, acc_z_out;
      input[15:0] vel_z;
      input[23:0] pos_z;

      output[7:0] led;
      reg[7:0] led;
      output[639:0] dots;

      wire[39:0] dot1, dot2, dot3, dot4, dot5, dot6, dot7, dot8,
            dot9, dot10, dot11, dot12, dot13, dot14, dot15, dot16;
      reg[5:0] let1, let2, let3, let4, let5, let6, let7, let8,
                  let9, let10, let11, let12, let13, let14, let15, let16;

      reg old_wire_mode, old_tilt_mode;
      reg[23:0] counter;

      Letter_to_Dots LD1(clk, let1, dot1);
      Letter_to_Dots LD2(clk, let2, dot2);
      Letter_to_Dots LD3(clk, let3, dot3);
      Letter_to_Dots LD4(clk, let4, dot4);
      Letter_to_Dots LD5(clk, let5, dot5);
      Letter_to_Dots LD6(clk, let6, dot6);
      Letter_to_Dots LD7(clk, let7, dot7);
      Letter_to_Dots LD8(clk, let8, dot8);
      Letter_to_Dots LD9(clk, let9, dot9);
      Letter_to_Dots LD10(clk, let10, dot10);
      Letter_to_Dots LD11(clk, let11, dot11);
      Letter_to_Dots LD12(clk, let12, dot12);
      Letter_to_Dots LD13(clk, let13, dot13);
      Letter_to_Dots LD14(clk, let14, dot14);
      Letter_to_Dots LD15(clk, let15, dot15);
      Letter_to_Dots LD16(clk, let16, dot16);

      assign dots = {dot1, dot2, dot3, dot4, dot5, dot6, dot7, dot8,
                  dot9, dot10, dot11, dot12, dot13, dot14, dot15,
dot16};


      always @(posedge clk)
      begin
```

```
//default
let1=6'd0;
let2=6'd0;
let3=6'd0;
let4=6'd0;
let5=6'd0;
let6=6'd0;
let7=6'd0;
let8=6'd0;
let9=6'd0;
let10=6'd0;
let11=6'd0;
let12=6'd0;
let13=6'd0;
let14=6'd0;
let15=6'd0;
let16=6'd0;
led=8'd0;

if(init_state)
begin       //initializing
      let1=6'd9;
      let2=6'd14;
      let3=6'd9;
      let4=6'd20;
      let5=6'd9;
      let6=6'd1;
      let7=6'd12;
      let8=6'd9;
      let9=6'd26;
      let10=6'd9;
      let11=6'd14;
      let12=6'd7;
      let13=6'd27;
      let14=6'd27;
      let15=6'd27;

      old_wire_mode=switch[0];
      old_tilt_mode=switch[1];
      counter=0;
end
else
begin
      if(switch[0] != old_wire_mode)
      begin
            if(switch[0])
            begin //wireless
                  let1=6'd23;
                  let2=6'd9;
                  let3=6'd18;
                  let4=6'd5;
                  let5=6'd12;
                  let6=6'd5;
                  let7=6'd19;
                  let8=6'd19;
            end
            else
            begin //wire
```

```
            let1=6'd23;
            let2=6'd9;
            let3=6'd18;
            let4=6'd5;
      end
      if(&counter)
      begin
            counter=0;
            old_wire_mode=switch[0];
      end
      else counter=counter+1;
end
else if(switch[1] != old_tilt_mode)
begin
      if(~switch[1])
      begin //tilt mode
            let1=6'd20;
            let2=6'd9;
            let3=6'd12;
            let4=6'd20;
            let5=6'd0;
            let6=6'd13;
            let7=6'd15;
            let8=6'd4;
            let9=6'd5;
      end
      else
      begin //abs pos mode
            let1=6'd1;
            let2=6'd2;
            let3=6'd19;
            let4=6'd0;
            let5=6'd16;
            let6=6'd15;
            let7=6'd19;
            let8=6'd0;
            let9=6'd13;
            let10=6'd15;
            let11=6'd4;
            let12=6'd5;
      end
      if(&counter)
      begin
            counter=0;
            old_tilt_mode=switch[1];
      end
      else counter=counter+1;
end
else
begin
      //X, Y, or Z
      case(switch[7:6])
            2'b00: let1=6'd24;
            2'b01: let1=6'd25;
            2'b10: let1=6'd26;
            default: let1=6'd0;
      endcase
```

```
if(let1 != 6'd0)
begin
      //-AXIS<SPACE>
      let2=6'd28;
      let3=6'd1;
      let4=6'd24;
      let5=6'd9;
      let6=6'd19;
      let7=6'd0;

      case(switch[5:2])
            4'b0000:  //RAW DATA
            begin
                  let8=6'd18;
                  let9=6'd1;
                  let10=6'd23;
                  let11=6'd0;
                  let12=6'd4;
                  let13=6'd1;
                  let14=6'd20;
                  let15=6'd1;

                  case(switch[7:6])
                        2'b00: led=acc_x_reg;
                        2'b01: led=acc_y_reg;
                        2'b10: led=acc_z_reg;
                        default: led=8'd0;
                  endcase
            end
            4'b0001: //INIT
            begin
                  let8=6'd1;
                  let9=6'd3;
                  let10=6'd3;
                  let11=6'd0;
                  let12=6'd9;
                  let13=6'd14;
                  let14=6'd9;
                  let15=6'd20;

                  case(switch[7:6])
                        2'b00: led=acc_x_init;
                        2'b01: led=acc_y_init;
                        2'b10: led=acc_z_init;
                        default: led=8'd0;
                  endcase
            end
            4'b0010: //OUT
            begin
                  let8=6'd1;
                  let9=6'd3;
                  let10=6'd3;
                  let11=6'd0;
                  let12=6'd15;
                  let13=6'd21;
                  let14=6'd20;

                  case(switch[7:6])
```

```
            2'b00: led=acc_x_out;
            2'b01: led=acc_y_out;
            2'b10: led=acc_z_out;
            default: led=8'd0;
      endcase
end
4'b0011: //VEL[15:8]
begin
      let8=6'd22;
      let9=6'd5;
      let10=6'd12;
      let11=6'd0;
      let12=6'd30;
      let13=6'd34;
      let14=6'd28;
      let15=6'd37;

      case(switch[7:6])
            2'b00: led=vel_x[15:8];
            2'b01: led=vel_y[15:8];
            2'b10: led=vel_z[15:8];
            default: led=8'd0;
      endcase
end
4'b0100: //VEL[7:0]
begin
      let8=6'd22;
      let9=6'd5;
      let10=6'd12;
      let11=6'd0;
      let12=6'd0;
      let13=6'd36;
      let14=6'd28;
      let15=6'd29;

      case(switch[7:6])
            2'b00: led=vel_x[7:0];
            2'b01: led=vel_y[7:0];
            2'b10: led=vel_z[7:0];
            default: led=8'd0;
      endcase
end
4'b0101: //POS[23:16]
begin
      let8=6'd16;
      let9=6'd15;
      let10=6'd19;
      let11=6'd0;
      let12=6'd31;
      let13=6'd32;
      let14=6'd28;
      let15=6'd30;
      let16=6'd35;

      case(switch[7:6])
            2'b00: led=pos_x[23:16];
            2'b01: led=pos_y[23:16];
            2'b10: led=pos_z[23:16];
```

```
                                                default: led=8'd0;
                                            endcase
                                        end
                                        default: led=8'd0;
                                    endcase
                                end //if(let1 != 6'd0)
                            end //else
                        end
                    end
                endmodule
```

Divider

```
module Divider(clk, reset, enable);
      input clk, reset;
      output enable;
      reg[9:0] count;
      reg enable;

   parameter CLK_SPEED = 500;

      always @(posedge clk)
      begin

            if (!reset) begin
                  count <= 0;
                  enable <= 0;
            end
            else count <= count + 1;

            if (count==CLK_SPEED-1)
            begin
                  enable <= 1;
                  count <= 0;
            end
            else enable <= 0;
      end
endmodule
```

Serial FSM

```
module FSM_Serial(clk, reset, load_data, rxd, data_ready);
    input clk, reset, rxd;
    reg rxd_int; //internal register
    output load_data, data_ready;
    reg load_data, data_ready;

    reg[1:0] state, next;
    reg[4:0] data_counter; //counts the number of bits
    reg[7:0] baud_counter; //?????
    reg inc_data, reset_data;
    reg inc_baud, reset_baud;

    parameter RATE = 234;//for baud rate = 115200
    parameter LENGTH = 8;
    parameter NUM_BITS = 3;

    parameter WAIT_START = 0;
    parameter START = 1;
    parameter READ_DATA = 2;
    parameter STOP = 3;

    always @(posedge clk)
    begin
        if(!reset) state <= WAIT_START;
        else state <= next;

        if(!reset_data) data_counter <= 0;
        else if (inc_data) data_counter <= data_counter + 1;

        if(!reset_baud) baud_counter <= 0;
        else if (inc_baud) baud_counter <= baud_counter + 1;

        rxd_int <= rxd;
    end

    always @(state or rxd_int or data_counter or baud_counter)
    begin
        //default
        load_data = 0; data_ready = 0;
        inc_data = 0; reset_data = 1;
        inc_baud = 1; reset_baud = 1;

        case(state)
            WAIT_START:
            begin
                reset_data = 0; reset_baud = 0;
                inc_baud = 0;
                if(!rxd_int) next = START;
                else next = WAIT_START;
            end
            START:
            begin
                if((baud_counter == RATE/2) && rxd_int) next =
WAIT_START;
                else if(baud_counter == RATE)
                begin
```

```
                    next = READ_DATA;
                    reset_baud = 0; inc_baud = 0;
                end
                else next = START;
            end
            READ_DATA:
            begin
                if(baud_counter == RATE)
                begin
                    reset_baud = 0;
                    inc_data = 1;
                    if(&data_counter[2:0]) next = STOP;
                    else
                    begin
                        next = READ_DATA;
                    end
                end
                else
                begin
                    next = READ_DATA;
                    if(baud_counter == RATE/2) load_data = 1;
                end
                     end
            STOP:
            begin
                if((baud_counter == RATE/2)&&!rxd_int) next =
WAIT_START;
                else if(baud_counter == RATE)
                begin
                    reset_baud = 0;
                    if(data_counter[4:3] == 2'b11)
                    begin
                        data_ready = 1;
                        next = WAIT_START;
                    end
                    else next = START;
                end
                else next = STOP;
            end
            default: next = WAIT_START;
        endcase
    end
endmodule
```

Analog/Digital FSM

```
module FSM_AD(clk, reset, start, r_wbar, cs_bar, status, le_adc);
    input clk, start, reset, status;
    output r_wbar, cs_bar, le_adc;

    reg status_int, start_int;
    reg r_wbar, cs_bar, le_adc;
    reg[2:0] state, next;
    reg[3:0] counter;
    reg counter_reset, counter_enable;

    parameter IDLE = 0;
    parameter WRITE = 1;
    parameter WAITSTATUSHIGH = 2;
    parameter WAITSTATUSLOW = 3;
    parameter READ = 4;
    parameter REG = 5;

    always @(posedge clk)
    begin
        if(!reset) state <= IDLE;
        else state <= next;
        if(counter_reset) counter <= 0;
        else if(counter_enable) counter <= counter + 1;

        status_int <= status;
         start_int <= start;
    end

    always @(state or status_int or start_int or counter)
    begin
        //defaults
        r_wbar = 1; cs_bar = 0;
        le_adc = 0;
        counter_reset = 0;
        counter_enable = 0;

        case(state)
            IDLE: begin
                cs_bar = 1;
                counter_reset = 1;
                if(start_int) next = WRITE;
                else next = IDLE;
            end
            WRITE: begin
                r_wbar = 0;
                counter_enable = 1;
                if(counter == 4'b1011) next = WAITSTATUSHIGH;
                else next = WRITE;
            end
            WAITSTATUSHIGH: begin
                if(status_int) next = WAITSTATUSLOW;
                else next = WAITSTATUSHIGH;
            end
            WAITSTATUSLOW: begin
                counter_reset = 1;
                if(!status_int) next = READ;
```

```
                else next = WAITSTATUSLOW;
            end
            READ: begin
                counter_enable = 1;
                if(counter == 4'b1011) begin
                    next = IDLE;
                    le_adc = 1;
                end
                else next = READ;
            end

            default: next = IDLE;
        endcase
    end
endmodule
```

Iinitialization Register

```
module Init_Register(clk, init_reset, reset, init_out);
    //init_reset: resets the register
    //reset: "reset" initializes the system

      input clk, init_reset, reset;
      output init_out;
      reg init_out;

      always @(posedge clk or negedge init_reset)
      begin
            if(!init_reset) init_out <= 0;
            else if(!reset) init_out <= 1;
            else init_out <= init_out;
      end
endmodule
```

Letter to Dot Conversion Code

```
module Letter_to_Dots(clk, letter, dots);
      input clk;
      input[5:0] letter;
      output[39:0] dots;
      reg[39:0] dots;

      always @(posedge clk)
      begin
      case(letter)
            6'd1: dots=40'b01111000_00010110_00010001_00010110_01111000;
//A
            6'd2: dots=40'b01111111_01001001_01001001_01001001_00110110;
//B
            6'd3: dots=40'b00111110_01000001_01000001_01000001_00100010;
//C
            6'd4: dots=40'b01111111_01000001_01000001_00100010_00011100;
//D
            6'd5: dots=40'b01111111_01001001_01001001_01001001_01101011;
//E
            6'd6: dots=40'b01111111_01000101_00000101_00000001_00000011;
//F
            6'd7: dots=40'b00111110_01000001_01010001_01010001_00110110;
//G
            6'd8: dots=40'b01111111_01000101_00000100_01000001_01111111;
//H
            6'd9: dots=40'b01100011_01000001_01111111_01000001_01100011;
//I
            6'd10:
dots=40'b00111000_01001000_01000000_01000001_00111111; //J
            6'd11:
dots=40'b01111111_01001001_00010100_00100010_01000001; //K
            6'd12:
dots=40'b01111111_00000001_01000000_01000000_01100000; //L
            6'd13:
dots=40'b01111111_01000110_00001000_01000110_01111111; //M
            6'd14:
dots=40'b01111111_01000110_00001000_00110001_01111111; //N
            6'd15:
dots=40'b00111110_01000001_01000001_01000001_00111110; //O
            6'd16:
dots=40'b01111111_01001001_00001001_00001001_00000110; //P
            6'd17:
dots=40'b00111110_01001001_01010001_01100001_00111110; //Q
            6'd18:
dots=40'b01111111_01001001_00011001_00101001_01000110; //R
            6'd19:
dots=40'b00100110_01001001_01001001_01001001_00110010; //S
            6'd20:
dots=40'b00000011_00000001_01111111_00000001_00000011; //T
            6'd21:
dots=40'b00111111_01000001_01000000_01000001_00111111; //U
            6'd22:
dots=40'b00000111_00111001_01000000_00111001_00000111; //V
            6'd23:
dots=40'b01111111_00110001_00001000_00110001_01111111; //W
            6'd24:
```

```
dots=40'b01100011_01010101_00001000_01010101_01100011; //X
            6'd25:
dots=40'b00000111_00001001_01110000_00001001_00000111; //Y
            6'd26:
dots=40'b01100011_01010001_01001001_01000101_01100011; //Z
            6'd27:
dots=40'b00000000_00000000_01100000_01100000_00000000; //.
            6'd28:
dots=40'b00001000_00001000_00001000_00001000_00001000; //-
            6'd29:
dots=40'b00111110_01000001_01000001_01000001_00111110; //0
                                    //0
            6'd30:
dots=40'b00000000_00000000_01000010_01111111_01000000; //1
            6'd31:
dots=40'b01000110_01100001_01010001_01010001_01001110; //2
            6'd32:
dots=40'b01100001_01001101_01010101_01010101_01100011; //3
            6'd33:
dots=40'b00001110_00001000_00001000_00001000_01111111; //4
            6'd34:
dots=40'b01100111_01001001_01001001_01001001_01110011; //5
            6'd35:
dots=40'b00111110_01001001_01001001_01001001_00110010; //6
            6'd36:
dots=40'b01100011_00010001_00001001_00000101_00000011; //7
            6'd37:
dots=40'b00111010_01000101_01000101_01000101_00111010; //8
            6'd38:
dots=40'b00100110_01001001_01001001_01001001_00111110; //9
            default:
dots=40'b00000000_00000000_00000000_00000000_00000000; //SPACE
      endcase
      end
endmodule
```

Multiplier

```
module Multiplier(X, Y, P);
    //decay by 15/16 for a 16-bit 2's complement velocity
    input[15:0] X, Y;
    output[31:0] P;
    reg[31:0] p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13,
p14, p15, p16, P;

    always @(X or Y)
    begin

        p1 = X * Y[0];
        p1[15] = ~p1[15];

        p2 = X * Y[1];
        p2[15] = ~p2[15];
        p2 = p2 << 1;

        p3 = X * Y[2];
        p3[15] = ~p3[15];
        p3 = p3 << 2;

        p4 = X * Y[3];
        p4[15] = ~p4[15];
        p4 = p4 << 3;

        p5 = X * Y[4];
        p5[15] = ~p5[15];
        p5 = p5 << 4;

        p6 = X * Y[5];
        p6[15] = ~p6[15];
        p6 = p6 << 5;

        p7 = X * Y[6];
        p7[15] = ~p7[15];
        p7 = p7 << 6;

         p8 = X * Y[7];
        p8[15] = ~p8[15];
        p8 = p8 << 7;

         p9 = X * Y[8];
        p9[15] = ~p9[15];
        p9 = p9 << 8;

         p10 = X * Y[9];
        p10[15] = ~p10[15];
        p10 = p10 << 9;

         p11 = X * Y[10];
        p11[15] = ~p11[15];
        p11 = p11 << 10;

         p12 = X * Y[11];
        p12[15] = ~p12[15];
        p12 = p12 << 11;
```

```
         p13 = X * Y[12];
        p13[15] = ~p13[15];
        p13 = p13 << 12;

         p14 = X * Y[13];
        p14[15] = ~p14[15];
        p14 = p14 << 13;

         p15 = X * Y[14];
        p15[15] = ~p15[15];
        p15 = p15 << 14;

        p16 = X * Y[15];
         p16[14] = ~p16[14];
         p16[13] = ~p16[13];
         p16[12] = ~p16[12];
         p16[11] = ~p16[11];
         p16[10] = ~p16[10];
         p16[9] = ~p16[9];
         p16[8] = ~p16[8];
         p16[7] = ~p16[7];
        p16[6] = ~p16[6];
        p16[5] = ~p16[5];
        p16[4] = ~p16[4];
        p16[3] = ~p16[3];
        p16[2] = ~p16[2];
        p16[1] = ~p16[1];
        p16[0] = ~p16[0];
        p16 = p16 << 15;

        P = p1+p2+p3+p4+p5+p6+p7+p8+p9+p10+p11+p12+p13+p14+p15+p16+
{1'b1, 14'd0, 1'b1, 16'd0};

    end
endmodule
```

Major FSM

```
module MajorFSM(clk, reset, data_ready, init_reset, init_out,
load_init, load_reg, load_filter, load_int_1, load_int_2, init_state);
      input clk, reset, data_ready, init_out;
      reg data_ready_int, init_out_int; //internal registers
      output init_reset, load_filter, load_init, load_reg;
      output load_int_1, load_int_2, init_state;
      reg init_reset, load_filter, load_init, load_reg, init_state;
      reg load_int_1, load_int_2;

      reg[1:0] init_counter;
      reg init_counter_reset, init_counter_inc;

      reg[2:0] state, next;

      parameter WAIT = 0;
      parameter FILTER = 1;
      parameter REGISTER_DATA = 2;
      parameter INTEGRATE_1 = 3;
      parameter INTEGRATE_2 = 4;
      parameter INITIAL = 5;
      parameter INITIAL_WAIT = 6;

      always @(posedge clk)
      begin
            if(!reset) state <= WAIT;
            else state <= next;

            data_ready_int <= data_ready;
            init_out_int <= init_out;

            if(init_counter_reset) init_counter <=0;
            else if(init_counter_inc) init_counter <= init_counter + 1;
            else init_counter <= init_counter;
      end

      always @(state or data_ready_int or init_out_int or init_counter)
      begin
            //default
            load_int_1 = 0; load_int_2 = 0;
            init_reset = 1; load_init = 0; load_reg = 0;
            load_filter = 0; init_state = 0;

            init_counter_reset = 1;
            init_counter_inc = 0;

            case(state)
                  WAIT:
                  begin
                      if(init_out_int) next = INITIAL_WAIT;
                      else if(data_ready_int) next = REGISTER_DATA;
                      else next = WAIT;
                  end
                  INITIAL_WAIT:
                  begin
                      init_counter_reset = 0;
                      init_reset = 0;
```

```
                    init_state = 1;
                    if(data_ready_int) next = INITIAL;
                    else next = INITIAL_WAIT;
                end
                INITIAL:
                begin
                    init_counter_reset = 0;
                    init_counter_inc = 1;
                    load_init = 1;
                    init_state = 1;
                    if(&init_counter) next = WAIT;
                    else next = INITIAL_WAIT;
                end
                REGISTER_DATA:
                begin
                    load_reg = 1;
                    next = FILTER;
                end
                FILTER:
                begin
                      load_filter = 1;
                      next = INTEGRATE_1;
                end
                INTEGRATE_1:
                begin
                      load_int_1 = 1;
                      next = INTEGRATE_2;
                end
                INTEGRATE_2:
                begin
                      load_int_2 = 1;
                      next = WAIT;
                end
                default: next = WAIT;
            endcase
      end
endmodule
```

Synchronizer

```
module Syncrhonizer(clk, reset, reset_sync);
    input clk, reset;
    output reset_sync;

    reg reset_sync, reset_int_1, reset_int_2;

    always @(posedge clk)
    begin
            reset_sync <= reset_int_2;
            reset_int_2 <= reset_int_1;
            reset_int_1 <= reset;
    end
endmodule
```

Serial Interface

```
module Serial_Interface(clk, load, rxd, acc_x, acc_y, acc_z);
    input clk, load, rxd;
    output[7:0] acc_x, acc_y, acc_z;
    reg[7:0] acc_x, acc_y, acc_z;

    always @(posedge clk)
    begin
        if(load)
        begin
            acc_x[0] <= acc_x[1];
            acc_x[1] <= acc_x[2];
            acc_x[2] <= acc_x[3];
            acc_x[3] <= acc_x[4];
            acc_x[4] <= acc_x[5];
            acc_x[5] <= acc_x[6];
            acc_x[6] <= acc_x[7];
            acc_x[7] <= acc_y[0];

            acc_y[0] <= acc_y[1];
            acc_y[1] <= acc_y[2];
            acc_y[2] <= acc_y[3];
            acc_y[3] <= acc_y[4];
            acc_y[4] <= acc_y[5];
            acc_y[5] <= acc_y[6];
            acc_y[6] <= acc_y[7];
            acc_y[7] <= acc_z[0];

            acc_z[0] <= acc_z[1];
            acc_z[1] <= acc_z[2];
            acc_z[2] <= acc_z[3];
            acc_z[3] <= acc_z[4];
            acc_z[4] <= acc_z[5];
            acc_z[5] <= acc_z[6];
            acc_z[6] <= acc_z[7];
            acc_z[7] <= rxd;
        end
        else
        begin
            acc_x <= acc_x;
            acc_y <= acc_y;
            acc_z <= acc_z;
        end
    end
endmodule
```

PS/2 Code (Verilog)

Clock Generator FSM

```
module clockgen(sysclk, reset, istart, ostart, abort, done, respond,
clk, state, enable, switch, counter);//, counter, inccounter,
resetcounter, switch, next, select);
      input sysclk, reset;
      input istart;                        // signals to start clock for
reading data
      input ostart;                        // signals to start clock for
writing data
      input abort;                         // stop clock generation, host
interrupt

      output done;                         // done with a clock
generation for an input/output
      output respond;                      // signals an host_request has
been processed, an acknowledgement bit must be sent
      output clk;
      // debug
      output [5:0] counter;
//    output inccounter;
//    output resetcounter;
//    output switch;
//    output [2:0] next;
//    output select;

      output [3:0] state;
      output enable;
      output switch;
      //
      wire enable;
      reg clk;
      reg done;
      reg respond;
      reg clk_sync;

      counter count(sysclk, reset, enable);

      reg [3:0] state;
      reg [3:0] next;

      parameter IDLE = 4'b0000;
      parameter SREAD = 4'b0010;
      parameter READ = 4'b0011;
      parameter RESPOND = 4'b0111;
      parameter RESPOND2 = 4'b0110;
      parameter BUFF = 4'b0100;
      parameter SWRITE = 4'b1100;
      parameter WRITE = 4'b1000;

      reg [5:0] counter;
      reg inccounter;
      reg resetcounter;
      reg switch;
      wire areset;
```

```
assign areset = abort || reset;

always @ (posedge inccounter or posedge resetcounter) begin
      if(resetcounter)
            counter <= 6'b0;
      else
            counter <= counter + 1;
end

always @ (posedge enable or posedge reset) begin
      if(reset) begin
            switch <= 0;
      end
      else switch <= ~switch;
end

always @ (posedge sysclk or posedge areset) begin
      if(areset) begin
            clk_sync <= 0;
            clk <= 1;
            inccounter <= 0;
            done <= 0;
            respond <= 0;
            resetcounter <= 1;
            next <= IDLE;
      end
      else begin
            clk_sync <= clk;
            state <= next;
      case (next)
            IDLE: begin
                  if(istart) next <= SREAD;
                  else if(ostart) next <= SWRITE;
                  else next <= IDLE;
                  resetcounter <= 1;
                  inccounter <= 0;
                  respond <= 0;
                  clk <= 1;
                  done <= 0;
            end
            SREAD: begin
                  respond <= 0;
                  if(enable && switch) begin
                        clk <= switch;
                        inccounter <= 0;
                        resetcounter <= 0;
                        next <= READ;
                        done <= 0;
                  end
                  else begin
                        next <= SREAD;
                        resetcounter <= 1;
                        inccounter <= 0;
                        done <= 0;
                        clk <= clk;
                  end
            end
            READ: begin
```

```
		if(enable) begin
			clk <= switch;
			if(counter == 19) begin
				next <= RESPOND;
				respond <= 1;
				resetcounter <= 0;
				inccounter <= 1;
				done <= 1;
			end
			else begin
				next <= READ;
				inccounter <= 1;
				respond <= 0;
				resetcounter <= 0;
				done <= 0;
			end
		end
		else begin
			next <= READ;
			respond <= 0;
			done <= 0;
			resetcounter <= 0;
			inccounter <= 0;
			clk <= clk;
		end
	end
	RESPOND: begin
		if(enable) begin
			next <= RESPOND2;
			respond <= 1;
			resetcounter <= 1;
			inccounter <= 0;
			clk <= switch;
			done <= 0;
		end
		else begin
			next <= RESPOND;
			respond <= 1;
			done <= 0;
			resetcounter <= 0;
			clk <= clk;
			inccounter <= inccounter;
		end
	end
	RESPOND2: begin
		if(enable) begin
			next <= IDLE;
			respond <= 1;
			inccounter <= 0;
			resetcounter <= 1;
			clk <= switch;
			done <= 1;
		end
		else begin
			next <= RESPOND2;
			respond <= 1;
			done <= 0;
			resetcounter <= resetcounter;
```

```
                        clk <= clk;
                        inccounter <= inccounter;
                  end
            end
            SWRITE: begin
                  respond <= 0;
                  if(enable && switch) begin
                        clk <= switch;
                        next <= WRITE;
                        inccounter <= 0;
                        resetcounter <= 1;
                        done <= 0;
                  end
                  else begin
                        next <= SWRITE;
                        clk <= 1;
                        inccounter <= 0;
                        resetcounter <= 0;
                        done <= 0;
                  end
            end
            WRITE: begin
                  clk <= switch;
                  respond <= 0;
                  if(enable) begin
                        if(counter == 21) begin
//                      if(counter >= 4 && switch) begin   //test
                              next <= IDLE;
                              resetcounter <= 0;
                              inccounter <= 1;
                              done <= 1;
                        end
/*                      else if(counter == 22) begin
                              next <= IDLE;
                              resetcounter <= 0;
                              inccounter <= 0;
                              done <= 0;
                        end
*/                      else begin
                              next <= WRITE;
                              inccounter <= 1;
                              done <= 0;
                              resetcounter <= 0;
                        end
                  end
                  else begin
                        next <= WRITE;
                        done <= 0;
                        inccounter <= 0;
                        resetcounter <= 0;
                  end
            end
            default: begin
                  next <= next;
                  clk <= clk;
                  respond <= respond;
                  inccounter <= counter;
                  resetcounter <= resetcounter;
```

```
                        done <= done;
                        respond <= respond;
                    end
                endcase
                end
        end
endmodule
```

Counter

```
module counter(clk, reset_synch, enable);

      input clk, reset_synch;
      output enable;
      reg [20:0] Q;
      reg enable;

      //  parameter total_cycles = 13;          7.4 us
      parameter total_cycles = 5;
            // 1600 ~ a little < 50 ms per short pulse

      // commented out parameters used for debugging the wave form

      always @ (posedge clk) begin
            enable <= 0;
            if(reset_synch) begin
                  Q <= 21'b0;
                  enable <= 1'b0;
            end
            else if(Q == total_cycles) begin
                  Q <= 21'b0;
                  enable <= 1'b1;
            end
            else begin
           Q <= Q + 1;
             enable <= 1'b0;
      end
   end

endmodule
```

Serial to Parallel Converter

```
module deserializer(clk, reset, data, start, obyte, busy, done,
updated, temp);
      input clk;
      input reset;
      input data;                                  // data bus input
      input start;                                 // don't read data being
output
      output done;                                 // signals we're done
      output [7:0] obyte;                          // here's where we
output the serialized data as a data byte
      output busy;                                 // Tells whoever that
we're still processing data

      // debug
      output updated;
      output [9:0] temp;
      //
      wire [7:0] byte;
      reg [7:0] obyte;
      reg [9:0] temp;                       // stores the 11-bit message
(only need 10, ignore start bit)
      reg busy, done;
      reg updated;                                 // variable to make sure
we start over fresh
      reg go;
/*
      always @ (posedge start or posedge done)
            if(start) begin
                  if(!done) go <= 1;
                  else go <= 0;
                  end
            else go <= 0;
*/
      always @ (posedge clk or posedge reset) begin
            if(reset) begin
            temp <= 10'h3FF;
            busy <= 0;
            done <= 0;
            updated <= 0;
             obyte <= 8'b0;
            end
            else if(start) begin
                  if(~temp[9]) begin            // finished, read last
data bit
                        temp <= temp * 2;
                        temp [0] <= data;
                        busy <= 0;
                        done <= 1;
                        updated <= 0;
                        obyte <= {temp[1], temp[2],temp[3],temp[4],temp
[5],temp[6],temp[7],temp[8]};
                  end
                  else begin
                        temp <= temp * 2;
                        temp[0] <= data;
                        busy <= 1;
```

```
                        done <= 0;
                        updated <= 1;
                        obyte <= obyte;
                  end
            end
            else begin
                  temp <= 10'b1111111110;
                  busy <= 1;
                  done <= 1;
                  updated <= 1;
                  obyte <= obyte;
            end

      end

      assign byte = {temp[2], temp[3],temp[4],temp[5],temp[6],temp[7],
temp[8],temp[9]};
endmodule
```

PS/2 Major FSM

```
module ps2fsm(sysclk, reset, inclk, indata, host_request, mupdate,
      inbusy, indone, outbusy, outdone, int_ld, response_sel, out_sel,
      m_sel, oupdate, beginread, clkoe, dataoe, clk, respond, t3done,
      state, done_sync, cstate, enable, switch, request, restart);

      // NOTE:  Need to add state checking data value when host releases
clock, if data is low, then read, else go idle again

      input sysclk;                 // 27 Mhz internal clock
      input reset;
      input inclk;                  // reading the clock bus
      input indata;                 // reading data bus
      input outbusy;                // signals last transmission in
progress
      input inbusy;                 // signals input data still being
read
      input indone;
      input outdone;                // signals completion of last
transmission

      input [7:0] host_request;     // incoming data byte

      input mupdate;                // interpreter sends update to
signal new mouse data needs to be sent


      output clkoe;                 // signal to control clock inout
      output dataoe;                // signal to control data inout
      output int_ld;                // load signal for interpreter
      output oupdate;               // indicator to serializer that some
output must be sent
      output beginread;       // tells deserialization to begin reading
      output out_sel;
      output [1:0] response_sel;
      output [1:0] m_sel;

      output clk;

      output respond;
      output t3done;
      // debug
      output done_sync;
      output [3:0] state;
      output [3:0] cstate;
      output enable;
      output switch;
      output [7:0] request;
      output restart;
      //

      wire enable;
      wire clkoe;
      reg dataoe;
      reg out_sel;
      reg [1:0] response_sel;
      reg [1:0] m_sel;
```

```
      reg oupdate;
      reg beginread;
      reg int_ld;                   // latch load signal
      reg resethelp;                // helper toggle to indicate two
output data bytes need to be sent (on host request ffh)

      reg [7:0] request;      // stored incoming data byte

      reg [11:0] hicount;           // counter for how long clock is
high  each count ~ 32ns
      reg [11:0] lowcount;          // counter for how long clock is low
      reg [14:0] sample;            // counter for sampling

//    counter(sysclk, reset, enable);  // 10 ms per enable pulse
                                          // 1200 ~ 5ms
      parameter SAMPLE_RATE = 2400;    //10 ms sample rate
//    parameter SAMPLE_RATE = 27;              // test sample rate

      reg [3:0] state;
      reg [3:0] next;

      parameter IDLE = 4'b0000;
      parameter C_HIGH = 4'b0001;
      parameter TRANSMIT = 4'b0011;
      parameter TRANSMIT2 = 4'b0111;
      parameter TRANSMIT3 = 4'b0110;
      parameter C_LOW = 4'b0100;
      parameter READ = 4'b1100;
      parameter RESPOND = 4'b1110;
      parameter RESPOND2 = 4'b1010;
      parameter RBUFF = 4'b1101;
      parameter RESPOND3 = 4'b1000;
      parameter RBUFF2 = 4'b0101;
      parameter RESPOND4 = 4'b1011;
      parameter RESETSTATE = 4'b1111;
      reg t3done;
      reg updated;
      wire cdone;

      reg done_sync;
      reg abort;
      reg restart;
      reg go;
      wire [3:0] cstate;
      // debug
      wire switch;
      wire [5:0] counter;
      clockgen cgen(sysclk, reset, beginread, oupdate, abort, cdone,
respond, clk, cstate, enable, switch, counter);

      assign clkoe = clk ? 0 : 1;

      parameter HIWAIT = 6;   // ~1700 for 50 ms
      parameter LOWAIT = 12;  // ~3400 for 100 ms

//    always @ (state or reset or inclk or mupdate or outdone or outbusy
or lowcount or hicount or inbusy or host_request or done_sync or
request or restart)
```

```
      always @ (posedge sysclk or posedge reset)
            if(reset) begin
                  response_sel <= 0;
                  oupdate <= 0;
                  dataoe <= 0;
                  next <= RESETSTATE;
                  request <= 8'b0;
                  m_sel <= 0;
                  out_sel <= 0;
                  abort <= 0;
                  restart <= 1;
                  hicount <= 12'b0;
                  lowcount <= 12'b0;
                  sample <= 15'h0;
                  state <= RESETSTATE;
                  done_sync <= 0;
                  go <= 0;
                  t3done <= 0;
                  end
            else begin
                  done_sync <= cdone;
                  state <= next;
                  if(inclk) begin
                        if(hicount < HIWAIT)
                              hicount <= hicount + 1;
                        else
                              hicount <= 12'd1700;    // HIWAIT, can we
set that?
                        lowcount <= 12'b0;
                  end
                  else begin
                        hicount <= 12'b0;
                        if(lowcount < LOWAIT)
                              lowcount <= lowcount + 1;
                        else
                              lowcount <= 12'd3400;
                  end
                  if(sample >= SAMPLE_RATE) begin
                        sample <= 15'h0;
                        int_ld <= 1;
                  end
                  else begin
                        sample <= sample + 1;
                        int_ld <= 0;
                  end
            response_sel <= response_sel;
            beginread <= 0;
            case (next)
                  RESETSTATE:
                        begin
                              next <= RESPOND2;
                              request <= 8'b11111111;
                              dataoe <= 0;
                              oupdate <= 1;
                              beginread <= 0;
                              restart <= 1;
                              abort <= 0;
                              t3done <= 0;
```

```
                        end
                  IDLE: begin
                        if(!inclk)
                              next <= C_LOW;
                        else if(mupdate) begin
                              next <= C_HIGH;
                        end
//                      else if(!indata && hicount >= HIWAIT/2) begin
//                            next <= READ;
//                      end
                        else
                              next <= IDLE;
                        oupdate <= 0;
                        m_sel <= 0;
                        dataoe <= 0;
                        out_sel <= 1;
                        request <= 8'b0;
                        restart <= 0;
                        abort <= 0;
                        go <= 0;
                        t3done <= t3done;
                        end
                  C_LOW: begin
                        if(lowcount >= LOWAIT && inclk && !indata) begin
                              next <= READ;
                              beginread <= 0;
                       end
                        else if(inclk) begin
                              next <= IDLE;
                              beginread <= 0;
                        end
                        else begin
                              next <= C_LOW;
                              beginread <= 0;
                        end
                        dataoe <= 0;
                        out_sel <= 1;
                        request <= 8'b0;
                        restart <= restart;
                        abort <= 0;
                        t3done <= t3done;
                        end
                  C_HIGH: begin
                        if(!inclk) begin
                              next <= IDLE;
                              oupdate <= 0;
                              dataoe <= 0;
                        end
                        else if(hicount >= HIWAIT && indata) begin
                              next <= TRANSMIT;
                              oupdate <= 1;
                              m_sel <= 0;
                              dataoe <= 1;
                        end
                        else begin
                              next <= C_HIGH;
                              oupdate <= 0;
                              dataoe <= 0;
```

```
                        end
                        request <= 8'b0;
                        restart <= restart;
                        out_sel <= 1;
                        abort <= 0;
                        t3done <= t3done;
                        end
                    READ: begin
/*                   if(lowcount >= LOWAIT - 1 && !beginread) begin
                       next <= C_LOW;
                       oupdate <= 0;
                        abort <= 1;
                    end
*/                        if (done_sync) begin
                                next <= RESPOND;
                                dataoe <= 1;
                                oupdate <= 0;
                                beginread <= 0;
                                end
                        else begin
                                next <= READ;
                                oupdate <= 0;
                                dataoe <= 0;
                                beginread <= 1;
                                go <= 0;
                        end
                        resethelp <= 0;
                        out_sel <= 0;
                        request <= 8'b0;
                        abort <= 0;
                        t3done <= t3done;
                        end
                    RESPOND: begin                          // state just
waits and loads request data
                        out_sel <= 0;
                        response_sel <= 0;
                        t3done <= t3done;
                        abort <= 0;
                        restart <= restart;
                        if(lowcount >= LOWAIT - 1) begin
                                next <= C_LOW;
                                oupdate <= 0;
                                abort <= 1;
                                dataoe <= 0;
                                request <= request;
                        end
                        else if(!go && (hicount < 5 || !done_sync))
begin
                                if(done_sync)
                                      go <= 1;
                                else go <= go;
                                request <= request;
                                next <= RESPOND;
                                oupdate <= 0;
                                dataoe <= 0;
                                abort <= 0;
                        end
                        else if(indata) begin
```

```
                              if(restart) begin
                                    request <= request;
                                    next <= RESPOND2;
                                    oupdate <= 1;
                                    dataoe <= 1;
                                    abort <= 0;
                              end
                              else if(host_request == 8'b0) begin
                                    next <= IDLE;   // invalid... what
should we do?
                                    oupdate <= 0;
                                    dataoe <= 0;
                                    request <= 8'b0;
                              end
                              else begin
                                    next <= RESPOND2;
                                    oupdate <= 1;
                                    dataoe <= 1;
                                   request <= host_request;
                              end
                        end
                        else begin
                              request <= request;
                              next <= RESPOND;
                              oupdate <= 0;
                              dataoe <= 0;
                              abort <= 0;
                        end
                  end
                  RESPOND2: begin
                        restart <= restart;
                        out_sel <= 0;
                        t3done <= t3done;
                        dataoe <= 1;
                        abort <= 0;
                        go <= 0;

                        if(restart) response_sel <= 1;   // responds
with AAh
                        else response_sel <= 0;

                        if(!done_sync) begin
                              request <= request;
                              next <= RESPOND2;
                              oupdate <= 0;
                        end

                        else if(restart) begin
                              next <= RBUFF2;     // goes to respond
with 00h
                              oupdate <= 0;
                              request <= request;
                        end
                        else if(request == 8'hF2) begin
                              next <= RBUFF2;
                              oupdate <= 0;
                              request <= request;
                        end
```

```
      else if(request == 8'hFF) begin
            next <= RBUFF;
            request <= 8'b0;
            oupdate <= 0;
      end
      else begin
            next <= IDLE;
            request <= 8'b0;
            oupdate <= 0;
      end
end
RBUFF: begin
      response_sel <= 2'b1;
      restart <= restart;
      t3done <= t3done;
      request <= request;
      if(hicount >= 5 && indata) begin
            request <= request;
            next <= RESPOND3;
            oupdate <= 1;
            response_sel <= 2;
            dataoe <= 1;
            abort <= 0;
      end
      else begin
            request <= request;
            next <= RBUFF;
            oupdate <= 0;
            response_sel <= 2;
            dataoe <= 0;
            abort <= 0;
      end
end
RESPOND3: begin
      response_sel <= 2'b1;   // corresponds to AAh
      restart <= restart;
            abort <= 0;
      t3done <= t3done;
            out_sel <= 0;
            request <= request;
            if(done_sync) begin
                  next <= RBUFF2;
                  request <= 8'b0;
            end
            else begin
                  next <= RESPOND3;
                  request <= request;
            end
            oupdate <= 0;
            dataoe <= 1;
end
RBUFF2: begin                    // 00
      response_sel <= 2;
      t3done <= t3done;
      restart <= restart;
      request <= request;
      if(hicount >= 5 && indata) begin
            request <= request;
```

```
                next <= RESPOND4;
                oupdate <= 1;
                response_sel <= 2;
                dataoe <= 1;
                abort <= 0;
        end
        else begin
                request <= request;
                next <= RBUFF2;
                oupdate <= 0;
                response_sel <= 2;
                dataoe <= 0;
                abort <= 0;
        end
end
RESPOND4: begin
        response_sel <= 2;      // corresponds to 00h
        restart <= restart;
        t3done <= t3done;
                abort <= 0;
                out_sel <= 0;
                request <= request;
                if(done_sync) begin
                        next <= IDLE;
                        request <= 8'b0;
                end
                else begin
                        next <= RESPOND4;
                        request <= request;
                end
                oupdate <= 0;
                dataoe <= 1;
end

TRANSMIT: begin
        /*if(lowcount >= LOWAIT - 1) begin
                next <= C_LOW;
                oupdate <= 0;
                abort <= 1;
        end
        else*/
        if(go && hicount >= HIWAIT) begin
                next <= TRANSMIT2;
                oupdate <= 1;
                m_sel <= 1;
                abort <= 0;
                go <= 0;
        end
        else begin
                if(done_sync)
                        go <= 1;
                else
                        go <= go;
                oupdate <= 0;
                next <= TRANSMIT;
                m_sel <= 0; // start from first data bit
                abort <= 0;
                end
```

```
                        dataoe <= 1;
                        out_sel <= 1;
                        request <= request;
                        t3done <= t3done;
                        end
                  TRANSMIT2: begin
                        /*if(lowcount >= LOWAIT - 1 && inclk && !indata)
begin
                              next <= C_LOW;
                              oupdate <= 0;
                              abort <= 1;
                        end
                        else */if(go && hicount >= HIWAIT) begin
                              next <= TRANSMIT3;
                              oupdate <= 1;
                              m_sel <= 2;
                              abort <= 0;
                        end
                        else begin
                              if(done_sync)
                                    go <= 1;
                              else
                                    go <= go;
                           next <= TRANSMIT2;
                              oupdate <= 0;
                              abort <= 0;
                              m_sel <= 1;
                        end
                        dataoe <= 1;
                        out_sel <= 1;     // output from interpreter
                        t3done <= t3done;
                        request <= request;
                        end
                  TRANSMIT3: begin
/*                      if(lowcount >= LOWAIT - 1 && inclk && !indata)
begin
                              next <= C_LOW;
                              oupdate <= 0;
                            abort <= 1;
                        end
                        else*/ if(done_sync) begin
                              next <= IDLE;
                              oupdate <= 0;
                              abort <= 0;
                              m_sel <= m_sel;
                              t3done <= 1;
                        end
                        else begin
                              oupdate <= 0;
                              next <= TRANSMIT3;
                              abort <= 0;
                              m_sel <= 2;             // start from
first data bit
                              t3done <= t3done;
                           end
                        dataoe <= 1;
                        out_sel <= 1;           // output from
interpreter
```

```
                        request <= request;
                    end
                    default: begin
                        request <= request;
                        oupdate <= 0;
                        next <= RESETSTATE;
                        t3done <= t3done;
                        abort <= 0;
                    end
                endcase
        end
endmodule
```

PS/2 Interface

```
module ps2interface2(sysclk, reset, dx, dy, z, lmb, mmb, rmb, iclk,
      idata, oclk, odata_sync, clkoe, dataoe, state, mupdate, oupdate,
      int_ld, done_sync, cstate, enable, switch, request, restart,
      store);

      input sysclk;                // sysclk is the system clock
(independent clock signal), controls operations
      input [8:0] dx;         // change in x and change in y since last
update (not sure how many bits yet
      input [8:0] dy;
      input [8:0] z;          // absolute z position
      input lmb, mmb, rmb;    // Is the corresponding mouse button being
pressed?
      input reset;                 // tell accelerometer interface to
reset x and y
      input iclk;                  // clk is bidirectional open
interface bus controlling ouput and input on data bus
      output oclk;
      input idata;                        // data is bidirectional open
interface bus containing serialized data packets in either direction
      output odata_sync;
      output clkoe;
      output dataoe;
      output [3:0] state;
      output mupdate;
      output oupdate;
      output int_ld;
      //debug for fsm
      output done_sync;
      output [3:0] cstate;
      output enable;
      output switch;
      output [7:0] request;
      output restart;
      output [9:0] store;
//    output dataoe;

      wire enable;
      wire switch;

      wire clkoe;
     wire dataoe;
      wire outclock;          // output of clock gen
      wire odata;
      wire dtpdone, dtpbusy;
      wire [7:0] dtpbyte;

      wire mupdate;
      wire ptddone, ptdbusy;

      wire [7:0] pdata;
      wire [3:0] state;

      // deserializer debug variables
      wire debug_updated;
      wire [9:0] debug_temp;
```

```
      //
      assign db = pdata;
     assign oclk = outclock;
      assign store = debug_temp;
      deserializer dtp(outclock, reset, idata, beginread, dtpbyte,
dtpbusy, dtpdone, debug_updated, debug_temp);

      // serializer debug vars
      wire [9:0] debug_storedata;
      //

     reg odata_sync;
      reg odata_temp;
     always @ (posedge sysclk) begin
            odata_sync <= odata_temp;
            odata_temp <= odata;
      end
      serializer ptd(outclock, reset, oupdate, pdata, outdata, ptdbusy,
ptddone, debug_storedata);

      wire [1:0] r_sel;
      wire [1:0] m_sel;
      wire t3done;

      // debug for fsm
      wire done_sync;
      wire [3:0] cstate;
      wire [7:0] counter;
      wire restart;

      ps2fsm fsm(sysclk, reset, iclk, idata, dtpbyte, mupdate, dtpbusy,
dtpdone, ptdbusy, ptddone, int_ld, r_sel, out_sel, m_sel, oupdate,
beginread, clkoe, dataoe, outclock, respond, t3done, state, done_sync,
cstate, enable, switch, request, restart);

      // interpreter debug vars
      wire [7:0] debug_d1;
      wire [7:0] debug_d2;
      wire [7:0] debug_d3;
      //

      wire [7:0] db1;                     // corresponds to a 3byte
message
      wire [7:0] db2;
      wire [7:0] db3;

      interpreter int(sysclk, reset, int_ld, t3done, dx, dy, z, lmb,
mmb, rmb, db1, db2, db3, mupdate, debug_d1, debug_d2, debug_d3);

   wire [7:0] mouse_data;
   wire [7:0] response_data;

      assign odata = respond ? 1'b1 : outdata;
      assign pdata = out_sel ? mouse_data : response_data;
      assign mouse_data = m_sel[1] ? db3 : m_sel[0] ? db2 : db1;
      assign response_data = r_sel[1] ? 8'h00 : r_sel[0] ? 8'hAA :
8'hFA;
```

```
endmodule
```

Parallel to Serial Converter

```
module serializer(clk, reset, oupdate, indata, outdata, busy, done,
                  storedata);
      input clk;              // clock signal generated by clockgen
      input reset;
      input oupdate;          // signals need to serialize some data

      input [7:0] indata;           // data from interpreter or response
      output outdata;               // serialized data
      output busy;                  // signals in process of sending
data
      output done;                  // sends signal indicating
completion of some transmission

      // debug
      output [9:0] storedata;
      //
//    reg outdatas;
      reg done, busy, outdata;

      reg update, updated;

      reg [9:0] storedata;
      wire parity;

      assign parity = indata[0] + indata[1] + indata[2] + indata[3] +
                        indata[4] + indata[5] + indata[6] + indata[7];

      always @ (oupdate or updated) begin
            if(oupdate && !updated) update <= 1;
            else if(updated) update <= 0;
            else update <= update;
      end

      always @ (posedge clk or posedge reset) begin
            if(reset) begin
                  storedata <= 10'b0;
                  outdata <= 0;
                  busy <= 1;
                  done <= 0;
                  updated <= 0;
//                outdatas <= 0;
            end
            else begin
            if(update) begin
                  storedata <= {2'b1, ~parity, indata[7], indata[6],
                        indata[5], indata[4], indata[3], indata[2],
                        indata[1]};
                  updated <= 1;
                  outdata <= indata[0];
                  busy <= 1;
                  done <= 0;
            end
/*          else if(!updated) begin
                  outdata <= 0;
                  storedata <= storedata;
                  busy <= 0;
```

```
                  updated <= 0;
                  done <= 0;
            end
*/
            else begin
                  outdata <= storedata[0];
                  if(storedata == 10'b1) begin
                        done <= 1;
                        busy <= 0;
                        updated <= 0;
                        storedata <= 10'b0;
                  end
                  else if(storedata == 10'b0) begin
                        done <= 1;
                        busy <= 0;
                        updated <= 0;
                        storedata <= 10'b0;
                  end
                  else begin
                        done <= 0;
                        busy <= 1;
                        updated <= 1;
                        storedata <= {1'b0, storedata[9:1]};
                  end
            end
//          outdatas <= outdata;
            end
      end
endmodule
```

Wireless Code (C)

```
/*
/  Shirley Li
/
/  3dMouse source
*/

//SET TO 1 FOR TRANSMITTER MODULE
//SET TO 0 FOR RECEIVER MODULE
#define transmitter 0

#include <reg1010.h>
#include <cc1010eb.h>
#include <hal.h>
#include <stdio.h>

// Protocol constants

#define PREAMBLE_BYTE_COUNT 7
#define PREAMBLE_BITS_SENSE 16
#define RF_RX_BUF_SIZE 50

// Test packet
#define TEST_STRING_LENGTH 3

byte adc_data[TEST_STRING_LENGTH];

rf_rx_display = FALSE;
byte rf_rx_string[TEST_STRING_LENGTH];
byte rf_rx_buf[RF_RX_BUF_SIZE];
byte rf_rx_index = 0;

void RFSetupReceive (void);

#define MAIN_MONITOR_TIMEOUT  0x00FF
int main_monitor = 0;

int i = 0;
int j = 0;
int packet_error_cnt = 0;
byte packet_error = FALSE;


void main(void) {
```

```
#ifdef FREQ868
// X-tal frequency: 14.745600 MHz
// RF frequency A: 868.277200 MHz          Rx
// RF frequency B: 868.277200 MHz          Tx
// RX Mode: Low side LO
// Frequency separation: 64 kHz
// Data rate: 2.4 kBaud
// Data Format: Manchester
// RF output power: 4 dBm
// IF/RSSI: RSSI Enabled

RF_RXTXPAIR_SETTINGS code RF_SETTINGS = {
   0x4B, 0x2F, 0x15,    // Modem 0, 1 and 2: Manchester, 2.4 kBaud
   //0x43, 0x2F, 0x15,    // Modem 0, 1 and 2: NRZ, 2.4 kBaud
   //0xA1, 0x2F, 0x29,    // Modem 0, 1 and 2: NRZ, 38.4 kBaud
   //0xA0, 0x2F, 0x52,    // Modem 0, 1 and 2: NRZ, 76.8 kBaud
   0x75, 0xA0, 0x00,    // Freq A
   0x58, 0x32, 0x8D,    // Freq B
   0x01, 0xAB,          // FSEP 1 and 0
   0x40,              // PLL_RX
   0x30,              // PLL_TX
   0x6C,               // CURRENT_RX
   0xF3,              // CURRENT_TX
   0x32,              // FREND
   0xFF,              // PA_POW
   0x00,              // MATCH
   0x00,              // PRESCALER
   };

#endif
        // Calibration data
   RF_RXTXPAIR_CALDATA xdata RF_CALDATA;

        // Disable watchdog timer
        WDT_ENABLE(FALSE);


   // Set optimum settings for speed and low power consumption
   MEM_NO_WAIT_STATES();
   FLASH_SET_POWER_MODE(FLASH_STANDBY_BETWEEN_READS);

        // Calibrate
   halRFCalib(&RF_SETTINGS, &RF_CALDATA);
```

```
        // Setup UART0 with polled I/O
        //UART0_SETUP(57600, CC1010EB_CLKFREQ, UART_NO_PARITY |
UART_RX_TX | UART_POLLED);
        UART0_SETUP(115200, CC1010EB_CLKFREQ, UART_NO_PARITY |
UART_RX_TX | UART_POLLED);    //CC1010EB_CLKFREQ = 14746

        WDT_ENABLE(FALSE);
        RLED_OE(TRUE);
        YLED_OE(TRUE);
        GLED_OE(TRUE);
        BLED_OE(TRUE);

        RLED = LED_OFF;
        YLED = LED_OFF;
        GLED = LED_ON;
        BLED = LED_OFF;


   if (transmitter==1)
        {
                //FOR TRANSMITTER

                // Setup ADC, turn it on
                halConfigADC(ADC_MODE_SINGLE | ADC_REFERENCE_VDD,
CC1010EB_CLKFREQ, 0); // 0: Threshold value (not used in this program)
                ADC_POWER(TRUE); // Power up ADC from sleep mode

                // Turn on RF, send packet
                //halRFOverrideBaudRate(RF_19200B);
                halRFSetRxTxOff(RF_TX, &RF_SETTINGS, &RF_CALDATA);
        }
        else
        {
                //FOR RECEIVER
                ADC_POWER(FALSE);

                //rf_rx_string[0]=0;
                //rf_rx_string[1]=1;
                //rf_rx_string[2]=2;

                // Set to RX
                halRFSetRxTxOff(RF_RX, &RF_SETTINGS, &RF_CALDATA);
```

```
                RFSetupReceive();

                // Reset main loop monitor
                 main_monitor = MAIN_MONITOR_TIMEOUT;
        }


while (TRUE)
        {
                if (transmitter==1)
                {
                YLED=~YLED;

                ADC_SELECT_INPUT(ADC_INPUT_AD0);
                adc_data[0] = ADC_GET_SAMPLE_8BIT();
                ADC_SAMPLE_SINGLE();
                printf("building %d from ADC 0\n",adc_data[0]);

                ADC_SELECT_INPUT(ADC_INPUT_AD1);
                adc_data[1] = ADC_GET_SAMPLE_8BIT();
                ADC_SAMPLE_SINGLE();
                printf("building %d from ADC 1\n",adc_data[1]);

                ADC_SELECT_INPUT(ADC_INPUT_AD2);
                adc_data[2] = ADC_GET_SAMPLE_8BIT();
                ADC_SAMPLE_SINGLE();
                printf("building %d from ADC 2\n",adc_data[2]);

                /*if(j%2 == 0){
                adc_data[0] = 0;
                adc_data[1] = 255;
                adc_data[2] = 0;
                }
                else{
                adc_data[0] = 255;
                adc_data[1] = 0;
                adc_data[2] = 255;
                }*/
                //j = j+1;

                halRFSendPacket(PREAMBLE_BYTE_COUNT, &adc_data[0], 3);


                /*printf("data_0 %d \n",(int)adc_data[0]);
                printf("data_1 %d \n",(int)adc_data[1]);
```

```
                printf("data_2 %d \n",(int)adc_data[2]);*/
                }

                else{

          // Detect RF packet error and display RF packet data:
          if(rf_rx_display == TRUE){
           rf_rx_display = FALSE;
                for(i = 2; i < TEST_STRING_LENGTH+2; i++){
                 //if(rf_rx_buf[i] != i-2){
             //    packet_error = TRUE;
             //    RLED = LED_ON;
            //     }

                //printf("%d \n",(int)rf_rx_buf[i]);

                        putchar (rf_rx_buf[i]);
                }

          if(packet_error == TRUE){
             packet_error_cnt++;
          }
          packet_error = FALSE;
          }else{
          }

        // Indicate main loop is running ok:
        if (main_monitor-- < 0x0000) {
          GLED=LED_ON;
                        BLED=LED_OFF;
          main_monitor = MAIN_MONITOR_TIMEOUT;
        }

    }


          }
}



// RF interrupt service routine:
void RF_ISR (void) interrupt INUM_RF {

    INT_ENABLE(INUM_RF, INT_OFF);
```

```
        INT_SETFLAG (INUM_RF, INT_CLR);

        // Get RF receive data
    rf_rx_buf[rf_rx_index] = RF_RECEIVE_BYTE();

        RLED = LED_OFF;

    switch(rf_rx_index){
       case 0:
          RF_LOCK_AVERAGE_FILTER(TRUE);
          if(rf_rx_buf[rf_rx_index] != RF_SUITABLE_SYNC_BYTE){
                                    RLED = LED_ON;
          }else{
          }
          rf_rx_index++;
          break;

       case 1:
          if(rf_rx_buf[rf_rx_index] != TEST_STRING_LENGTH){
            RLED = LED_ON;
          }else{
          }
          rf_rx_index++;
          break;

       case TEST_STRING_LENGTH+3:
          rf_rx_index = 0;
                        rf_rx_display = TRUE;

                        BLED=LED_ON;

          PDET &= ~0x80;
          PDET |= 0x80;
          break;

       default:
          if((rf_rx_buf[rf_rx_index] != rf_rx_string[rf_rx_index-2]) && (rf_rx_index <
TEST_STRING_LENGTH+2)){
           YLED = LED_ON;
          }else{
          }
          rf_rx_index++;
          break;
    }
```

```
    // Indicate packet reception

    INT_ENABLE(INUM_RF, INT_ON);

        return;
}

// Setup RF for RX
void RFSetupReceive (void) {

    // Disable global interrupt
    INT_GLOBAL_ENABLE (INT_OFF);

    // Setup RF interrupt
    INT_SETFLAG (INUM_RF, INT_CLR);
    INT_PRIORITY (INUM_RF, INT_HIGH);
    INT_ENABLE (INUM_RF, INT_ON);

    // Enable RF interrupt based on bytemode
    RF_SET_BYTEMODE();

    // Setup preamble configuration
    RF_SET_PREAMBLE_COUNT(16);
    RF_SET_SYNC_BYTE(RF_SUITABLE_SYNC_BYTE);

    // Make sure avg filter is free-running + 22 baud settling time
    MODEM1=(MODEM1&0x03)|0x24;

    // Reset preamble detection
    PDET &= ~0x80;
    PDET |= 0x80;

    // Start RX
    RF_START_RX();

    // Enable global interrupt
    INT_GLOBAL_ENABLE (INT_ON);

}
```