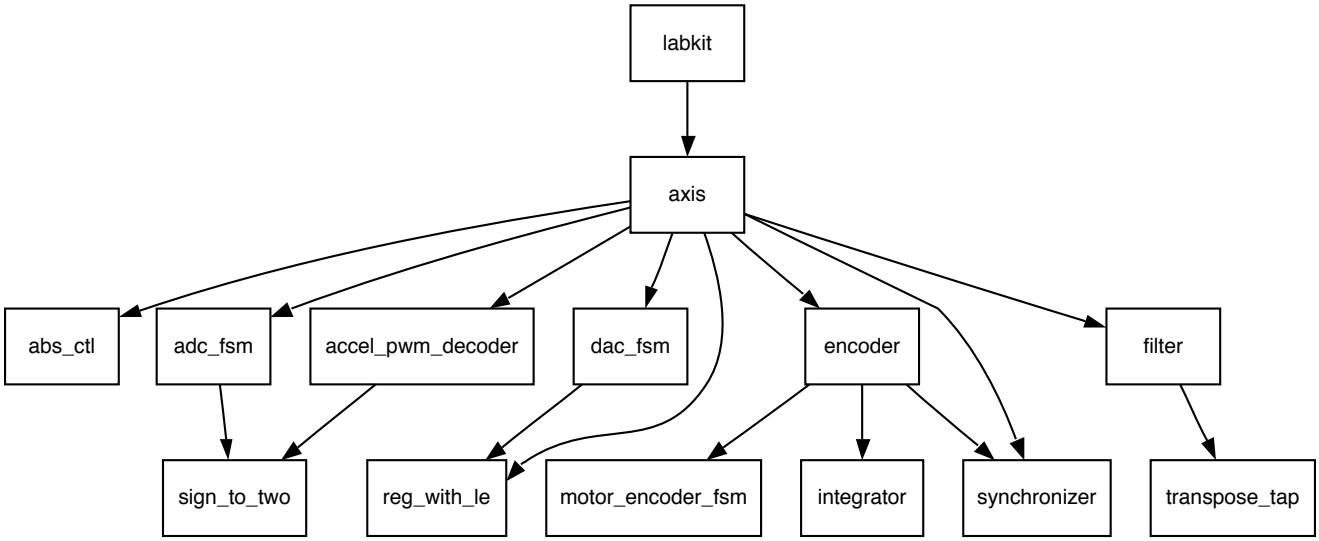
6.111 Final Project Appendix

Active Stabilization 3-Axis Sensor Platform

Scott Torborg, Kyle Vogt, Mike Scharfstein

Module Dependency Diagram



Dependencies between modules.

The source code for all Verilog modules is included in this document. The filter.v code is generated programmatically by a Perl script, so that script is included as well.

Module Code

labkit.v Verilog code

```
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;

    wire[7:0] dac3_data, pitch_dac_data, roll_dac_data;
    wire dac3_cs, pitch_dac_cs, roll_dac_cs;
    wire pitch_accel_y, pitch_accel_x, pitch_enc_b;
    wire pitch_enc_b_bar, pitch_enc_a, pitch_enc_a_bar;
    wire pitch_enc_ind, pitch_enc_ind_bar;
    wire roll_accel_y, roll_accel_x, roll_enc_b;
    wire roll_enc_b_bar, roll_enc_a, roll_enc_a_bar;
    wire roll_enc_ind, roll_enc_ind_bar;
    wire pitch_gyro_adc_sclk, pitch_gyro_adc_din, pitch_gyro_adc_cs;
    wire roll_gyro_adc_sclk, roll_gyro_adc_din, roll_gyro_adc_cs;

    wire[11:0] accel_reg, gyro_reg;
    wire global_reset;
    wire[17:0] encoder_count;
    wire[35:0] diff_amp;

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

```
// RS-232 Interface

//assign rs232_txd = 1'b1;

assign rs232_rts = 1'b1;
// rs232_rxd and rs232_cts are inputs


// PS/2 Ports

// mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs


// LED Displays

assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
// disp_data_out is an input


// Buttons, Switches, and Individual LEDs

//assign led = 8'hFF;

// button0, button1, button2, button3, button_enter, button_right,

// button_left, button_down, button_up, and switches are inputs


// Daughtercard Connectors

assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port

assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;
// systemace_irq and systemace_mpbrdy are inputs


// Logic Analyzer

assign analyzer1_data = {
   // pod A3 / A2

  encoder_count[17:2]
  //diff_amp[35:20]

};
```

```
// pod CK0

assign analyzer1_clock = clock_27mhz;

assign analyzer2_data = {
   // pod A1 / 4 bits of A0

  gyro_reg,
  // pod A0

   // roll gyro serial ADC

   roll_gyro_adc_din,
   roll_gyro_adc_sclk,
   roll_gyro_adc_cs,
   // DAC chip select lines

   roll_dac_cs
};

// pod CK1

assign analyzer2_clock = clock_27mhz;

assign analyzer3_data = {
   // pod C3

  roll_dac_data,
  // pod C2

   roll_accel_y,
   roll_accel_x,
   roll_enc_b,
   roll_enc_b_bar,
   roll_enc_a,
   roll_enc_a_bar,
   roll_enc_ind,
   roll_enc_ind_bar
};

// pod CK3

assign analyzer3_clock = clock_27mhz;

// user I/O bank 1: DACs

assign user1[31:24] = dac3_data;
assign user1[23] = dac3_cs;
assign user1[22:20] = 3'bZ;
assign user1[19:12] = pitch_dac_data;
assign user1[11] = pitch_dac_cs;
assign user1[10:9] = 2'bZ;
assign user1[8] = roll_dac_cs;
assign user1[7:0] = roll_dac_data;

// user I/O bank 2: pitch encoders and accelerometers
```

```
assign pitch_accel_y = user2[31];
assign pitch_accel_x = user2[30];
assign user2[29:6] = 24'bZ;
assign pitch_enc_b = user2[5];
assign pitch_enc_b_bar = user2[4];
assign pitch_enc_a = user2[3];
assign pitch_enc_a_bar = user2[2];
assign pitch_enc_ind = user2[1];
assign pitch_enc_ind_bar = user2[0];

// user I/O bank 3: gyro serial ADCs

assign user3[0] = pitch_gyro_adc_sclk;
assign pitch_gyro_adc_din = user3[1];
assign user3[2] = pitch_gyro_adc_cs;
assign user3[3] = roll_gyro_adc_sclk;
assign roll_gyro_adc_din = user3[4];
assign user3[5] = roll_gyro_adc_cs;

// user I/O bank 4: roll encoders and accelerometers

assign roll_accel_y = user4[31];
assign roll_accel_x = user4[30];
assign user4[29:6] = 24'bZ;
assign roll_enc_b = user4[5];
assign roll_enc_b_bar = user4[4];
assign roll_enc_a = user4[3];
assign roll_enc_a_bar = user4[2];
assign roll_enc_ind = user4[1];
assign roll_enc_ind_bar = user4[0];

assign global_reset = ~button0;

axis roll(
   .clk(clock_27mhz),
  .ext_reset(global_reset),
  .accel_pwm_x(roll_accel_x),
  .accel_pwm_y(roll_accel_y),
  .gyro_adc_cs(roll_gyro_adc_cs),
  .gyro_adc_din(roll_gyro_adc_din),
  .gyro_adc_sclk(roll_gyro_adc_sclk),
  .enc_a(roll_enc_a),
  .enc_b(roll_enc_b),
  .enc_ind(roll_enc_ind),
  //.uarttx(rs232_txd),

  //.uartrx(rs232_rxd),

  .dac_out(roll_dac_data),
  .dac_cs(roll_dac_cs),
  .gyro_reg(gyro_reg),
  .accel_reg(accel_reg),
  .encoder_position(encoder_count),
  .diff_amp(diff_amp)
);

axis pitch(
```

```
      .clk(clock_27mhz),
      .ext_reset(global_reset),
      .accel_pwm_x(pitch_accel_x),
      .accel_pwm_y(pitch_accel_y),
      .gyro_adc_cs(pitch_gyro_adc_cs),
      .gyro_adc_din(pitch_gyro_adc_din),
      .gyro_adc_sclk(pitch_gyro_adc_sclk),
      .enc_a(pitch_enc_a),
      .enc_b(pitch_enc_b),
      .enc_ind(pitch_enc_ind),
      //.uarttx(rs232_txd),

      //.uartrx(rs232_rxd),

      .dac_out(pitch_dac_data),
      .dac_cs(pitch_dac_cs)
   );


endmodule
```

axis.v Verilog code

```
// 6.111 Final Project
// Actively Stabilized Gimbal Sensor Platform
// Scott Torborg
// storborg@mit.edu
//
// Version 0.04
// Top-level Axis System

`include "abs_ctl.v"
`include "accel_pwm_decoder.v"
`include "adc_fsm.v"
`include "dac_fsm.v"
`include "encoder.v"
`include "filter.v"
`include "reg_with_le.v"
`include "synchronizer.v"

// include if not using Xilinx tools
//`include "MULT18X18.v"

module axis(
        // utilities
        clk,
        ext_reset,
        // accelerometer PWM inputs
        accel_pwm_y,
        accel_pwm_x,
        // gyro ADC, uses SPI interface
        gyro_adc_cs,
        gyro_adc_din,
        gyro_adc_sclk,
        // encoders
        enc_a,
        enc_b,
        enc_ind,
        // CAN transceiver I/O
        cantx,
        canrx,
        // UART I/O
        uarttx,
        uartrx,
        // DAC output lines
        dac_out,
        dac_cs,

        // debugging outputs
        sum_filter_out,
        accel_filter_out,
        accel_pwm_out,
        accel_reg,
        accel_normalized,
        gyro_filter_out,
        gyro_adc_out,
        gyro_reg,
        gyro_normalized,
```

```
);

// global utilities
input ext_reset;
input clk;
// accelerometer ADC lines
input accel_pwm_x;
input accel_pwm_y;
// gyro ADC lines
output gyro_adc_sclk;
output gyro_adc_cs;
input gyro_adc_din;
// quadrature encoders
input enc_a;
input enc_b;
input enc_ind;
// CAN transceiver
output cantx;
input canrx;
// UART I/O
output uarttx;
input uartrx;
// DAC output
output[7:0] dac_out;
output dac_cs;

// global synchronized reset
wire reset;

// configuration registers
reg[17:0] abs_ctl_desired;
reg[17:0] abs_ctl_gain;
reg[7:0] dac_override;
reg[7:0] dac_value;

wire[17:0] encoder_position;

output[11:0] accel_pwm_out;
output[11:0] accel_normalized;
output[11:0] accel_reg;

wire[17:0] accel_coeff_data;
wire[7:0] accel_coeff_addr;
wire accel_coeff_we;

output[31:0] accel_filter_out;

output[11:0] gyro_adc_out;
output[11:0] gyro_normalized;
output[11:0] gyro_reg;

wire[17:0] gyro_coeff_data;
wire[7:0] gyro_coeff_addr;
wire gyro_coeff_we;

output[31:0] gyro_filter_out;
```

```
output[31:0] sum_filter_out;

wire[31:0] motor_value;

// reset synchronizer
synchronizer reset_sync(
    .clk(clk),
    .in(ext_reset),
    .out(reset)
);

// accel PWM Y input block
accel_pwm_decoder accel_pwm_decoder_y(
    .reset(reset),
    // needs a 9MHz clock
    .clk(clk),
    .in(accel_pwm_y),
    .normalized_out(accel_normalized),
    .data_ready(accel_le)
);

// accel ADC input register
reg_with_le #(12) accel_pwm_reg(
    .reset(reset),
    .clk(clk),
    .in(accel_normalized),
    .le(accel_le),
    .out(accel_reg)
);

// gyro ADC input block
adc_fsm gyro_adc_fsm(
    .reset(reset),
    .clk(clk),
    .cs_b(gyro_adc_cs),
    .data_in(gyro_adc_din),
    .clk_out(gyro_adc_sclk),
    .data_ready(gyro_le),
    .data_out(gyro_normalized)
);

// gyro ADC input register
reg_with_le #(12) gyro_adc_reg(
    .reset(reset),
    .clk(clk),
    .in(gyro_normalized),
    .le(gyro_le),
    .out(gyro_reg)
);

// convolution filters
filter accel_filter(
    .clk(clk),
    .reset(reset),
    .in({accel_reg,6'b000000}),
    .coeff_data(accel_coeff_data),
    .coeff_addr(accel_coeff_addr),
```

```
        .coeff_we(accel_coeff_we),
        .out(accel_filter_out)
    );
    filter gyro_filter(
        .clk(clk),
        .reset(reset),
        .in({gyro_reg,6'b000000}),
        .coeff_data(gyro_coeff_data),
        .coeff_addr(gyro_coeff_addr),
        .coeff_we(accel_coeff_we),
        .out(gyro_filter_out)
    );

    assign sum_filter_out = gyro_filter_out + accel_filter_out;

    // encoder integration block
    encoder encoder_instance(
        .clk(clk),
        .reset(reset),
        .a(enc_a),
        .b(enc_b),
        .home(enc_ind),
        .pos(encoder_position)
    );

    // absolute control block
    abs_ctl abs_ctl_instance(
        .clk(clk),
        .reset(reset),
        .desired(abs_ctl_desired),
        .actual(encoder_position),
        .gain(abs_ctl_gain),
        .filter(sum_filter_out),
        .out(motor_value)
    );

    // DAC output block
    dac_fsm dac_fsm_instance(
        .clk(clk),
        .reset(reset),
        .enable_b(dac_cs),
        .data_in(motor_value[31:24]),
        .data_out(dac_out)
    );

endmodule
```

abs_ctl.v Verilog code

```
// 6.111 Final Project
// Actively Stabilized Gimbal Sensor Platform
// Scott Torborg
// storborg@mit.edu
//
// Version 0.02
// Absolute Control Override Block

//include if not using the xilinx tools
//`include "MULT18X18.v"

module abs_ctl(clk, reset, desired, actual, gain, filter, out, diff);
    parameter WIDTH = 32;

    input clk, reset;
    input[17:0] desired, actual, gain;
    input[WIDTH-1:0] filter;
    output[WIDTH-1:0] out;

    output[17:0] diff;
    wire[35:0] diff_amp;

    assign diff =  desired - actual;

    MULT18X18 amplifier(
        .A(gain),
        .B(diff),
        .P(diff_amp)
    );

    assign out = diff_amp[35:4] + filter;
endmodule
```

sign_to_two.v Verilog code

```
module sign_to_two(sign_mag, twos_complement);
    parameter WIDTH = 16;

    input[WIDTH-1:0] sign_mag;
    output[WIDTH-1:0] twos_complement;

    assign twos_complement = sign_mag[WIDTH-1] ? {1'b1, (~sign_mag[WIDTH-2:0])
+ 1} : sign_mag;
endmodule
```

adc_fsm.v Verilog code

```
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Mike Scharfstein
//
// Create Date:    13:23:44 05/09/05
// Module Name:    adc_fsm
// Revision 0.01 - File Created
// Additional Comments: 10MHz clk assumed. SPI interface
//      data_out in two's complement
//////////////////////////////////////////////////////////////////////////////////
module adc_fsm(clk, reset, cs_b, data_in, clk_out, data_ready, data_out);
    input clk;
    input reset;

    input data_in;
    output cs_b;
    output clk_out;
    reg cs_b, clk_out;

    output data_ready;
    reg data_ready;
    output [11:0] data_out;
    reg [11:0] data;

    reg[3:0] state, next_state;
    reg[4:0] bit_counter;
    reg[2:0] clk_out_counter;

    parameter IDLE = 0;
    parameter WAIT_FOR_START = 1;
    parameter READ_DATA = 2;
    parameter RESET = 3;

    parameter CLK_OUT_PERIOD = 8; //1.25MHz clk_out

    parameter CLK_OFF_DUTY = 4;

    sign_to_two #(12) sign_to_two_instance(
        .sign_mag({~data[11], data[10:0]}),
        .twos_complement(data_out)
    );

    always @ (posedge clk) begin
        if(reset) begin
            state <= IDLE;
            clk_out_counter <= 0;
        end else begin
            state <= next_state;

            if(state == IDLE) begin
                clk_out_counter <= 0;
            end else begin
                clk_out_counter <= clk_out_counter + 1;
            end

            if(cs_b) begin
```

```
                clk_out <= 0;
            end else if(clk_out_counter < CLK_OFF_DUTY) begin
                clk_out <= 0;
            end else begin
                clk_out <= 1;
            end
        end
    end

    always @ (posedge clk) begin
        data_ready <= 0;

        case(state)
            IDLE: begin
                cs_b <= 1;
                bit_counter <= 0;
                data_ready <= 1;

                next_state <= WAIT_FOR_START;
            end WAIT_FOR_START: begin
                cs_b <= 0;
                data_ready <= 1;

                if(clk_out_counter == CLK_OFF_DUTY) begin //on posedge clk_out

                    bit_counter <= bit_counter + 1;
                    if(bit_counter == 2) begin //at third clk_out rising edge,
change state

                        bit_counter <= 0;
                        next_state <= READ_DATA;
                    end
                end
            end READ_DATA: begin
                cs_b <= 0;
                data_ready <= 0;

                if(clk_out_counter == CLK_OFF_DUTY) begin //on posedge clk_out

                    bit_counter <= bit_counter + 1;

                    if(bit_counter == 0) begin
                        data <= data_in;
                    end else begin
                        data <= (data << 1) | data_in;
                    end

                    if(bit_counter == 11) begin//at third clk_out rising edge,
change state

                        bit_counter <= 0;
                        next_state <= RESET;
                    end
                end
            end RESET: begin
                cs_b <= 1;
                data_ready <= 1;
```

```
                bit_counter <= bit_counter + 1;

                if(bit_counter == 6) begin
                    next_state <= IDLE;
                end
            end default: begin
                next_state <= IDLE;
            end
        endcase
    end
endmodule
```

accel_pwm_decoder.v Verilog code

```
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Mike Scharfstein
//
// Create Date:    17:24:24 05/09/05
// Module Name:    accel_pwm_decoder
// Revision 0.01 - File Created
// Additional Comments: assume 9MHz clk Rset should be about 125kOhm so get
12-bit resolution.
//      in should be registered.
//      frequency is 1kHz, 222ns per bit (should be 244ns)
//      output: sign-magnitude value (12 bits) representing a value b/w +/-
1.2g
//////////////////////////////////////////////////////////////////////////////////
module accel_pwm_decoder(clk, reset, in, normalized_out, data_ready);
    input clk;
    input reset;
    input in;

    output [11:0] normalized_out;
    output data_ready;
    reg data_ready;
    reg [32:0] high;

    reg [2:0] state, next_state;
    reg enable;

    parameter IDLE = 0;
    parameter WAIT_FOR_RISE = 1;
    parameter WAIT_FOR_FALL = 2;

    sign_to_two #(12) sign_to_two_instance(
        .sign_mag({~high[11], high[10:0]}),
        .twos_complement(normalized_out)
    );

    always @ (posedge clk) begin
        if(reset) begin
            state <= IDLE;
            enable <= 0;
        end else begin
            state <= next_state;
            enable <= enable + 1;
        end
    end

    always @ (posedge enable) begin
        case(state)
            IDLE: begin
                high <= 0;
                data_ready <= 0;

                next_state <= WAIT_FOR_RISE;
            end WAIT_FOR_RISE: begin
                if(in) begin
                    high <= 0;
```

```
                    data_ready <= 0;

                    next_state <= WAIT_FOR_FALL;
                end else begin
                    data_ready <= 1;
                    next_state <= WAIT_FOR_RISE;
                end
            end WAIT_FOR_FALL: begin
                if(!in) begin
                    data_ready <= 1;
                    next_state <= WAIT_FOR_RISE;
                end else begin
                    high <= high + 1;
                end
            end IDLE: begin
                 next_state <= IDLE;
            end
        endcase
    end
endmodule
```

dac_fsm.v Verilog code

```
module dac_fsm(clk, reset, enable_b, data_in, data_out);
    input clk, reset;
    input [7:0] data_in;
    output enable_b;
    output [7:0] data_out;
    reg enable_b;

    reg[1:0] state, next_state;
    reg[2:0] cycle_count; // cycle = 100ns


    reg load_enable;

    wire[7:0] sign_mag_data_in;
    two_to_sign two_to_sign_instance(
        .twos_complement(data_in),
        .sign_magnitude(sign_mag_data_in)
    );
    reg_with_le #(8) REG(
        .clk(clk),
        .reset(reset),
        .le(load_enable),
        .in({~sign_mag_data_in[7], sign_mag_data_in[6:0]}), //convert to
binary offset from two's complement

        .out(data_out)
    );

    parameter IDLE = 0;
    parameter DATA_CONV = 1;

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

    always @ (posedge clk) begin
        case(state)
            IDLE: begin
                enable_b <= 1;
                cycle_count <= 0;
                load_enable <= 1;

                next_state <= DATA_CONV;
            end DATA_CONV: begin
                load_enable <= 0;
                enable_b <= 0;

                cycle_count <= cycle_count + 1;
                if(cycle_count == 3) begin
                   next_state <= IDLE;
                end else begin
```

```
                    next_state <= DATA_CONV;
                end
            end default: next_state <= IDLE;
        endcase
    end
endmodule
```

encoder.v Verilog code

```
// 6.111 Final Project
// Actively Stabilized Gimbal Sensor Platform
// Scott Torborg
// storborg@mit.edu
//
// Version 0.01
// Motor Encoder Wrapper

module encoder(clk, reset, a, b, home, pos);
    input clk, reset;
    input a, b, home;

    output[17:0] pos;

    wire count_pulse;
    wire direction;

    wire count_pulse_sync;
    wire direction_sync;

    synchronizer #(2) int_sync(
        .clk(clk),
        .in({count_pulse, direction}),
        .out({count_pulse_sync, direction_sync})
    );

    motor_encoder_fsm motor_encoder_fsm_instance(
        .A(a),
        .B(b),
        .count_pulse(count_pulse),
        .direction(direction)
    );

    integrator integrator_instance(
        .clk(clk),
        .reset(reset),
        .count_pulse(count_pulse_sync),
        .direction(direction_sync),
        .counts(pos)
    );
endmodule
```

motor_encoder_fsm.v Verilog code

```
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Mike Scharfstein
//
// Create Date:    22:04:47 05/09/05
// Module Name:    motor_encoder_fsm
// Revision 0.01 - File Created
// Additional Comments: A, B should be registered
//      outputs:
//          count_pulse: 1 encoder cound on posedge
//          direction: CW if 0, CCW if 1. determines polarity of count_pulse
//
//////////////////////////////////////////////////////////////////////////////////
module motor_encoder_fsm(A, B, count_pulse, direction);
    input A;
    input B;

    output count_pulse, direction;
    reg count_pulse, direction;

    reg [1:0] last_AB_state;
    wire [1:0] AB_state;

    assign AB_state = {A, B};

    always @ (A or B) begin
        case(AB_state)
            2'b00: begin
                count_pulse <= 1;

                if(last_AB_state == 2'b10) begin
                    direction <= 0;
                end else begin
                    direction <= 1;
                end

                last_AB_state <= AB_state;
            end default: begin
                count_pulse <= 0;

                last_AB_state <= AB_state;
            end
        endcase
    end
endmodule
```

integrator.v Verilog code

```
//////////////////////////////////////////////////////////////////////
// Engineer: Mike Scharfstein
//
// Create Date:    23:14:03 05/09/05
// Module Name:    integrator
// Revision 0.01 - File Created
//////////////////////////////////////////////////////////////////////
module integrator(clk, reset, count_pulse, direction, counts);
    input clk;
    input reset;
    input count_pulse;
    input direction;

    output [17:0] counts;
    reg [17:0] counts;

    reg reset_reg;

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

    parameter COUNTS_PER_REVOLUTION = 37000;

    always @(posedge count_pulse) begin
        if(reset_reg) begin
            counts <= 0;
            reset_reg <= 0;
        end

        if(direction) begin //CCW

            if(counts == 0) begin
                counts <= COUNTS_PER_REVOLUTION;
            end else begin
                counts <= counts - 1;
            end
        end else begin //CW

            if(counts == COUNTS_PER_REVOLUTION) begin
                counts <= 0;
            end else begin
                counts <= counts + 1;
            end
        end
    end
endmodule
```

filter.v Verilog code

```
// 6.111 Final Project
// Actively Stabilized Gimbal Sensor Platform
// Scott Torborg
// storborg@mit.edu
//
// Version 0.XX
// Convolution Filter

// Generated by "./gen_filter.pl 16"

`include "transpose_tap.v"

module filter(clk, reset, in, coeff_data, coeff_addr, coeff_we, out);
    parameter WIDTH = 32;
    parameter ADDR_WIDTH = 4;
    parameter TAPS = 16;

    input clk, reset;

    // interface to coefficient register bank
    input coeff_we;
    input[17:0] coeff_data;
    input[ADDR_WIDTH-1:0] coeff_addr;

    input[WIDTH-1:0] in;
    output[WIDTH-1:0] out;

    // register bank to store coefficients
    reg[17:0] coeffs[TAPS-1:0];

    // input register
    reg[17:0] in_reg;

    wire[WIDTH-1:0] tap1in;
    wire[WIDTH-1:0] tap2in;
    wire[WIDTH-1:0] tap3in;
    wire[WIDTH-1:0] tap4in;
    wire[WIDTH-1:0] tap5in;
    wire[WIDTH-1:0] tap6in;
    wire[WIDTH-1:0] tap7in;
    wire[WIDTH-1:0] tap8in;
    wire[WIDTH-1:0] tap9in;
    wire[WIDTH-1:0] tap10in;
    wire[WIDTH-1:0] tap11in;
    wire[WIDTH-1:0] tap12in;
    wire[WIDTH-1:0] tap13in;
    wire[WIDTH-1:0] tap14in;
    wire[WIDTH-1:0] tap15in;

    // reset to default coefficients
    always @ (posedge clk)
    begin
        if(reset)
        begin
```

```
            // should be a low pass
            coeffs[0] = 18'h00100;
            coeffs[1] = 18'h000d7;
            coeffs[2] = 18'h00074;
            coeffs[3] = 18'h0000c;
            coeffs[4] = - 18'h00030;
            coeffs[5] = - 18'h00031;
            coeffs[6] = - 18'h0000c;
            coeffs[7] = 18'h00018;
            coeffs[8] = 18'h00020;
            coeffs[9] = 18'h0000c;
            coeffs[10] = - 18'h0000e;
            coeffs[11] = - 18'h00018;
            coeffs[12] = - 18'h0000c;
            coeffs[13] = 18'h00008;
            coeffs[14] = 18'h00012;
            // most recent value
            coeffs[15] = 18'h0000b;


            /*
            coeffs[0] = 18'h00000;
            coeffs[1] = 18'h00000;
            coeffs[2] = 18'h00000;
            coeffs[3] = 18'h00000;
            coeffs[4] = 18'h00000;
            coeffs[5] = 18'h00000;
            coeffs[6] = 18'h00000;
            coeffs[7] = 18'h00000;
            coeffs[8] = 18'h00000;
            coeffs[9] = 18'h00000;
            coeffs[10] = 18'h00000;
            coeffs[11] = 18'h00000;
            coeffs[12] = 18'h00000;
            coeffs[13] = 18'h00000;
            coeffs[14] = 18'h00000;
            // most recent value
            coeffs[15] = 18'h00100;
            */
        end
    end

    // to write to coefficients
    always @ (posedge clk)
    begin
        if(coeff_we)
        begin
            coeffs[coeff_addr] <= coeff_data;
        end
    end

    // input register
    always @ (posedge clk)
    begin
        in_reg <= in;
    end
```

```
// transpose filter structure
transpose_tap #(WIDTH) tap0(
    .clk(clk),
    .reset(reset),
    .filter_in(in_reg),
    .coeff_in(coeffs[0]),
    .cascade_in(0),
    .cascade_out(tap1in)
);

transpose_tap #(WIDTH) tap1(
    .clk(clk),
    .reset(reset),
    .filter_in(in_reg),
    .coeff_in(coeffs[1]),
    .cascade_in(tap1in),
    .cascade_out(tap2in)
);

transpose_tap #(WIDTH) tap2(
    .clk(clk),
    .reset(reset),
    .filter_in(in_reg),
    .coeff_in(coeffs[2]),
    .cascade_in(tap2in),
    .cascade_out(tap3in)
);

transpose_tap #(WIDTH) tap3(
    .clk(clk),
    .reset(reset),
    .filter_in(in_reg),
    .coeff_in(coeffs[3]),
    .cascade_in(tap3in),
    .cascade_out(tap4in)
);

transpose_tap #(WIDTH) tap4(
    .clk(clk),
    .reset(reset),
    .filter_in(in_reg),
    .coeff_in(coeffs[4]),
    .cascade_in(tap4in),
    .cascade_out(tap5in)
);

transpose_tap #(WIDTH) tap5(
    .clk(clk),
    .reset(reset),
    .filter_in(in_reg),
    .coeff_in(coeffs[5]),
    .cascade_in(tap5in),
    .cascade_out(tap6in)
);

transpose_tap #(WIDTH) tap6(
    .clk(clk),
```

```
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[6]),
        .cascade_in(tap6in),
        .cascade_out(tap7in)
    );

    transpose_tap #(WIDTH) tap7(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[7]),
        .cascade_in(tap7in),
        .cascade_out(tap8in)
    );

    transpose_tap #(WIDTH) tap8(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[8]),
        .cascade_in(tap8in),
        .cascade_out(tap9in)
    );

    transpose_tap #(WIDTH) tap9(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[9]),
        .cascade_in(tap9in),
        .cascade_out(tap10in)
    );

    transpose_tap #(WIDTH) tap10(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[10]),
        .cascade_in(tap10in),
        .cascade_out(tap11in)
    );

    transpose_tap #(WIDTH) tap11(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[11]),
        .cascade_in(tap11in),
        .cascade_out(tap12in)
    );

    transpose_tap #(WIDTH) tap12(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[12]),
```

```
        .cascade_in(tap12in),
        .cascade_out(tap13in)
    );

    transpose_tap #(WIDTH) tap13(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[13]),
        .cascade_in(tap13in),
        .cascade_out(tap14in)
    );

    transpose_tap #(WIDTH) tap14(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[14]),
        .cascade_in(tap14in),
        .cascade_out(tap15in)
    );

    transpose_tap #(WIDTH) tap15(
        .clk(clk),
        .reset(reset),
        .filter_in(in_reg),
        .coeff_in(coeffs[15]),
        .cascade_in(tap15in),
        .cascade_out(out)
    );

endmodule
```

gen_filter.pl Perl code, used to generate filter.v

```
#!/usr/bin/perl -w

# 6.111 Final Project FIR Filter Generator

# Scott Torborg

$taps = $ARGV[0] || die "usage: ./gen_filter.pl <taps>";

$addr_width = log($taps)/log(2);

print <<EOF;

// 6.111 Final Project
// Actively Stabilized Gimbal Sensor Platform
// Scott Torborg
// storborg\@mit.edu
//
// Version 0.XX
// Convolution Filter

// Generated by \"./gen_filter.pl $taps\"

`include "transpose_tap.v"

module filter (clk, reset, in, coeff_data, coeff_addr, coeff_we, out);
    parameter WIDTH = 32;
    parameter ADDR_WIDTH = $addr_width;
    parameter TAPS = $taps;

    input clk, reset;

    // interface to coefficient register bank
    input coeff_we;
    input[17:0] coeff_data;
    input[ADDR_WIDTH-1:0] coeff_addr;

    input[17:0] in;
    output[WIDTH-1:0] out;

    // register bank to store coefficients
    reg[17:0] coeffs[TAPS-1:0];

    // input register
    reg[17:0] in_reg;

EOF

for($k=1; $k < $taps; $k++) {
    print "\twire[WIDTH-1:0] tap".$k."in;\n";
}

print <<EOF;

    // to write to coefficients
    always \@ (posedge clk)
```

```
    begin
        if(coeff_we)
        begin
            coeffs[coeff_addr] <= coeff_data;
        end
    end

    // input register
    always \@ (posedge clk)
    begin
        in_reg <= in;
    end

    // transpose filter structure
EOF

for($k=0; $k < $taps; $k++) {
    print "\ttranspose_tap #(WIDTH) tap".$k." (\n";
    print "\t\t.clk(clk),\n";
    print "\t\t.reset(reset),\n";
    print "\t\t.filter_in(in_reg),\n";
    print "\t\t.coeff_in(coeffs[$k]),\n";
    if($k == 0) {
        print "\t\t.cascade_in(0),\n";
    } else {
        print "\t\t.cascade_in(tap".$k."in),\n";
    }
    if($k == $taps - 1) {
        print "\t\t.cascade_out(out)\n";
    } else {
        print "\t\t.cascade_out(tap".($k+1)."in)\n";
    }
    print "\t);\n\t\n";
}

print "endmodule\n";
```

transpose_tap.v Verilog code

```
// 6.111 Final Project
// Actively Stabilized Gimbal Sensor Platform
// Scott Torborg
// storborg@mit.edu
//
// Version 0.02
// Transpose Filter Segment

// include if testing without the xilinx tools
`include "MULT18X18.v"

module transpose_tap(clk, reset, filter_in, coeff_in, cascade_in,
cascade_out);
    parameter WIDTH = 32;

    input clk, reset;
    input[WIDTH-1:0] cascade_in;

    // multiplicand width fixed to 18 bits,
    // because HW multipliers are 18 bits
    input[17:0] filter_in, coeff_in;

    output[WIDTH-1:0] cascade_out;
    reg[WIDTH-1:0] cascade_out;

    wire[35:0] product;

    MULT18X18 coeff_mult(
        .A(coeff_in),
        .B(filter_in),
        .P(product)
    );

    always @ (posedge clk)
    begin
        cascade_out <= cascade_in + {{4{product[35]}},product[35:8]};
    end

endmodule
```

reg_with_le.v Verilog code

```
// 6.111 Final Project
// Actively Stabilized Gimbal Sensor Platform
// Scott Torborg
// storborg@mit.edu
//
// Version 0.02
// Register with Load Enable

module reg_with_le(clk, reset, in, le, out);
    parameter WIDTH = 32;

    input clk, reset, le;
    input[WIDTH-1:0] in;

    output[WIDTH-1:0] out;
    reg[WIDTH-1:0] out;

    always @ (posedge clk)
    begin
        if(reset)
        begin
            out <= 0;
        end
        else if(le)
        begin
            out <= in;
        end
        else
        begin
            out <= out;
        end
    end
endmodule
```

uart_rx.v Verilog code

```
module uart_rx(clk, reset, tick8, rx, rx_data, rx_flag);

    input clk, reset, tick8, rx, rx_flag;
    output [7:0] rx_data;

    parameter start =   0;
    parameter bit0  =   1;
    parameter bit1 =    2;
    parameter bit2 =    3;
    parameter bit3 =    4;
    parameter bit4  =   5;
    parameter bit5 =    6;
    parameter bit6 =    7;
    parameter bit7 =    8;
    parameter stop =    9;

    reg [7:0] rx_data;
    reg [3:0] state;
    reg [1:0] rx_sync;
    reg [1:0] rx_cont;
    reg rx_bit;
    reg [2:0] spacing;
    reg next_bit;
    reg data_time;

    // Determine if we are looking at a real data bit

    always @ (posedge clk) begin
        if((state != start) && (state != stop)) data_time <= 1'b1;
        else data_time <= 1'b0;
    end

    // Syncronize incoming rx signal

    always @ (posedge clk)
        if(tick8) rx_sync <= {rx_sync[0], rx};

    // Filter incoming rx signal for short spikes

    always @(posedge clk) begin
        if(tick8) begin
            if(rx_sync[1] && rx_cont != 2'b11) rx_cont <= rx_cont + 1;
            else if(~rx_sync[1] && rx_cont != 2'b00) rx_cont <= rx_cont - 1;

            if(rx_cont == 2'b00) rx_bit <= 0;
            else if(rx_cont == 2'b11) rx_bit <= 1;
        end
    end

    // State transition logic:

    //   Go to next state if we've oversampled 8 times and the bit is good

    //   This is signaled by next_bit going high
```

```
    always @(posedge clk) begin
    if(tick8) begin
    case(state)
        start: if(~rx_bit) state <= bit0;
        bit0: if(next_bit) state <= bit1;
        bit1: if(next_bit) state <= bit2;
        bit2: if(next_bit) state <= bit3;
        bit3: if(next_bit) state <= bit4;
        bit4: if(next_bit) state <= bit5;
        bit5: if(next_bit) state <= bit6;
        bit6: if(next_bit) state <= bit7;
        bit7: if(next_bit) state <= stop;
        stop: if(next_bit) state <= start;
        default: state <= start;
    endcase
    end
    end

    wire next_bit = (spacing == 7);

    // Next bit comes now.  We've sampled 8 times.

    always @(posedge clk) begin
        if(state == 0) spacing <= 0;
        else if(tick8) spacing <= spacing + 1;
    end

    // Shift in the new bit if it's time

    always @(posedge clk) begin
        if(tick8 && next_bit && data_time) rx_data <= {rx_bit, rx_data[7:1]};
    end

endmodule
```

uart_tx.v Verilog code

```
module uart_tx(clk, reset, tick, tx, tx_data, tx_flag);

    input clk, reset, tick, tx_start;
    input [7:0] tx_data;
    output tx;

    parameter idle  =   0;      // We wait here until start signal received

    parameter start =   1;      // Each data byte is preceeded by a start bit

    parameter bit0  =   2;
    parameter bit1  =   3;
    parameter bit2  =   4;
    parameter bit3  =   5;
    parameter bit4  =   6;
    parameter bit5  =   7;
    parameter bit6  =   8;
    parameter bit7  =   9;
    parameter stop  =   10;     // And followed by a stop bit


    reg [3:0] state;
    reg txbit;

    // State transition logic:

    //   Only advances if this clock cycle is the special one containing

    //   the tick signal.  Not every clock cycle is that special!!!


    always @(posedge clk) begin
    if(reset) state <= idle;
    else begin
        case(state)
            idle: if(tx_flag) state <= start;   // Move on if start signal
present

            start: if(tick) state <= bit0;
            bit0: if(tick) state <= bit1;
            bit1: if(tick) state <= bit2;
            bit2: if(tick) state <= bit3;
            bit3: if(tick) state <= bit4;
            bit4: if(tick) state <= bit5;
            bit5: if(tick) state <= bit6;
            bit6: if(tick) state <= bit7;
            bit7: if(tick) state <= stop;
            stop: if(tick) state <= idle;       // We're done, wait for next
tx flag

            default: if(tick) state <= idle;
        endcase
    end
    end
```

```
    // State logic:

    //   Basically assign txbit as high unless the current data bit says
otherwise


    always @(state)
    case(state)
        0: txbit <= 1'b1;
        1: txbit <= 1'b1;
        2: txbit <= tx_data[0];
        3: txbit <= tx_data[1];
        4: txbit <= tx_data[2];
        5: txbit <= tx_data[3];
        6: txbit <= tx_data[4];
        7: txbit <= tx_data[5];
        8: txbit <= tx_data[6];
        9: txbit <= tx_data[7];
        10: txbit <= 1'b1;
        default: txbit <= 1'b1;
    endcase

    // Update the state of the tx pin


    assign tx = txbit;

endmodule
```

baudtick.v Verilog code

```
module baudtick(clk, tick);

input clk;
output tick;
reg [18:0] acc;

always @ (posedge clk) acc <= acc[17:0] + 1118;

assign tick = acc[18];

endmodule
```

baudtick8.v Verilog code

```
module baudtick8(clk, tick8);

input clk;
output tick8;

reg [18:0] acc;

always @ (posedge clk) acc <= acc[17:0] + 140;

assign tick8 = acc[18];

endmodule
```

command.v Verilog code

```
module command(clk, reset, rx_ready, tx_ready, rx_reg, tx_reg, out1, out2,
out3, out4, out5, out6, out7, out8, out9, out10);

    input clk, reset, rx_ready;
    input [7:0] rx_reg;
    output tx_ready;
    output [7:0] tx_reg;
    output [7:0] out1;
    output [7:0] out2;
    output [7:0] out3;
    output [7:0] out4;
    output [7:0] out5;
    output [7:0] out6;
    output [7:0] out7;
    output [7:0] out8;
    output [7:0] out9;
    output [7:0] out10;

    reg [7:0] combyte;
    reg [7:0] databyte;

    parameter ackbyte =     6;  // Standard ack value

    parameter errorbyte =   255;    // We got a bad command

    parameter com1 =    101;    // Values for regs out1...outn

    parameter com2 =    102;
    parameter com3 =    103;    // These are pretty much arbitrary

    parameter com4 =    104;    // as long as the host knows what they are!

    parameter com5 =    105;
    parameter com6 =    106;
    parameter com7 =    107;
    parameter com8 =    108;
    parameter com9 =    109;
    parameter com10 =   110;

    alwyas @ (posedge clk) begin
        if(reset) begin
            state <= idle;
        end
        else begin
            case(state)
                idle: begin
                    if(rx_ready) state <= command;
                    tx_reg <= 0;
                    tx_flag <= 1'b0;
                command: begin
                    combyte <= rx_reg;      // Store for addressing of data
later

                    case(rx_reg)
                        com1: state <= idle2;   // Valid command, wait for
```

```
data!

                        com2: state <= idle2;
                        com3: state <= idle2;
                        com4: state <= idle2;
                        com5: state <= idle2;
                        com6: state <= idle2;
                        com7: state <= idle2;
                        com8: state <= idle2;
                        com9: state <= idle2;
                        com10: state <= idle2;
                        default: begin
                            state <= idle;  // Command bad, back to idle

                            tx_reg <= errorbyte;
                            tx_flag <= 1'b1;
                        end
                    endcase
                    end
                idle2: if(rx_ready) state <= data;
                data: begin
                    databyte <= rx_reg;     // Store data for writing later

                    case(combyte)
                        com1: out1 <= rx_reg;   // Store data in proper
register

                        com2: out2 <= rx_reg;
                        com3: out3 <= rx_reg;
                        com4: out4 <= rx_reg;
                        com5: out5 <= rx_reg;
                        com6: out6 <= rx_reg;
                        com7: out7 <= rx_reg;
                        com8: out8 <= rx_reg;
                        com9: out9 <= rx_reg;
                        com10: out10 <= rx_reg;
                        default: state <= idle;
                    endcase
                    state <= idle;
                    tx_reg <= ackbyte;
                    tx_flag <= 1'b1;
                    end
            endcase
        end
    end
endmodule
```

synchronizer.v Verilog code

```
// 6.111 Final Project
// Actively Stabilized Gimbal Sensor Platform
// Scott Torborg
// storborg@mit.edu
//
// Version 0.01
// Generic 2-cycle Synchronizer

module synchronizer(clk, in, out);
    parameter WIDTH = 1;

    input clk;
    input[WIDTH-1:0] in;
    output[WIDTH-1:0] out;
    reg[WIDTH-1:0] out;
    reg[WIDTH-1:0] int;

    always @ (posedge clk)
    begin
        int <= in;
        out <= int;
    end
endmodule
```

Testbench Code

filter_test.v Verilog code

```
`include "filter.v"
`include "synchronizer.v"

module filter_test;
    reg clk, reset;
    reg[31:0] in;
    reg[17:0] coeff_data;
    reg[3:0] coeff_addr;
    reg coeff_we;

    wire[31:0] out;

    wire[31:0] in_sync;

    synchronizer #(32) input_sync(
        .clk(clk),
        .in(in),
        .out(in_sync)
    );

    filter foo(
        .clk(clk),
        .reset(reset),
        .in(in_sync),
        .coeff_data(coeff_data),
        .coeff_addr(coeff_addr),
        .coeff_we(coeff_we),
        .out(out)
    );

    initial begin
        clk = 0;
        reset = 1;
        coeff_we = 0;
        in = 32'h000fffff;
        #4
        reset = 0;
    end

    always #1 clk = ~clk;

    always #2 in = -in;

    initial begin
        $display("\t\ttime,\tclk,\trst,\tin,\t\tcd,\tca,\tcwe,\tout");
        $monitor("%d,\t%b,\t%b,\t%h,\t%h,\t%b,\t%b,\t%h",$time,clk,reset,in_sy
nc,coeff_data,coeff_addr,coeff_we,out);
        #800
        $finish;
    end
```

uart_test.v Verilog code

```
//////////////////////////////////////////////////////////////////////
// Engineer: Kyle Vogt
//
// Create Date:    23:20:21 05/10/05
// Module Name:    uart_test
// Revision 0.01 - File Created
//////////////////////////////////////////////////////////////////////
module uart_test(clk, reset, rx, tx);
    input clk;
    input reset;
    input rx;
    output tx;

    wire tick, tick8;

    uart_rx uart_rx_instance(
        .clk(clk),
        .reset(reset),
        .rx_data(rx_data),
        .rx_flag(rx_flag),
        .rx(rx_pin),
        .tick8(tick8)
    );

    uart_tx uart_tx_instance(
        .clk(clk),
        .reset(reset),
        .tx_data(tx_data),
        .tx_flag(tx_flag),
        .tx(tx_pin),
        .tick(tick)
    );

    baudtick bt(
        .clk(clk),
        .tick(tick)
    );

    baudtick8 bt8(
        .clk(clk),
        .tick8(tick8)
    );
endmodule
```

transpose_tap_test.v Verilog code

```
`include "transpose_tap.v"

module transpose_tap_test;
    reg clk, reset;
    reg[31:0] cascade_in;
    reg[17:0] coeff_in, filter_in;

    wire[31:0] cascade_out;

    transpose_tap foo(
        .clk(clk),
        .reset(reset),
        .filter_in(filter_in),
        .cascade_in(cascade_in),
        .coeff_in(coeff_in),
        .cascade_out(cascade_out)
    );

    initial begin
        clk = 0;
        reset = 1;
        filter_in = 0;
        coeff_in = 0;
        cascade_in = 128;
        #4
        reset = 0;
    end

    always #1 clk = ~clk;

    initial begin
        $display("\t\ttime,\tclk,\trst,\tfin,\tcin,\tceff,\tcout");
        $monitor("%d,\t%b,\t%b,\t%h,\t%h,\t%h,\t%h",$time,clk,reset,filter_in,
cascade_in,coeff_in,cascade_out);
        #800
        $finish;
    end
```

abs_ctl_test.v Verilog code

```
`include "abs_ctl.v"

module abs_ctl_test;
    reg clk, reset;
    reg[17:0] desired, actual, gain;

    reg[31:0] filter;
    wire[31:0] out;
    wire[17:0] diff;

    abs_ctl foo(
        .clk(clk),
        .reset(reset),
        .desired(desired),
        .actual(actual),
        .gain(gain),
        .filter(filter),
        .out(out),
        .diff(diff)
    );

    initial begin
        clk = 0;
        reset = 1;
        filter = 32'hffccffcc;
        desired = 18'h04000;
        actual = 18'h00001;
        gain = 18'h1cccc;
        #4
        reset = 0;
    end

    always #1 clk = ~clk;

    always #2 actual = actual << 1;

    initial begin
        $display("\t\ttime,\tdes,\tact,\tgn,\tfil,\tout,\tdiff");
        $monitor("%d,\t%h,\t%h,\t%h,\t%h,\t%h,\t%h",$time,desired,actual,gain,
filter,out,diff);
        #100
        $finish;
    end

endmodule
```

synchronizer_test.v Verilog code

```
`include "synchronizer.v"

module synchronizer_test;
    reg in;
    wire out;
    reg clk;

    synchronizer sync(
        .clk(clk),
        .in(in),
        .out(out)
    );

    initial begin
        clk = 0;
        in = 0;
        #5
        in = 1;
        #3
        in = 0;
    end

    always #2 clk = ~clk;

    initial begin
        $display("\t\ttime,\tclk,\tin,\tout");
        $monitor("%d,\t%b,\t%b,\t%b",$time,clk,in,out);
        #30
        $finish;
    end
endmodule
```