9 Appendix

9.1 Angle Checker

```
module anglechecker(lags, valid, satisfied, angle);

          input [27:0] lags;
          input [4:0] valid;
          output satisfied;
          output[7:0] angle;

        parameter ANGLE = 0;
        parameter L12 = 0;
        parameter H12 = 0;
        parameter L23 = 0;
        parameter H23 = 0;
        parameter L34 = 0;
        parameter H34 = 0;
        parameter L45 = 0;
        parameter H45 = 0;

        wire t12;
        wire t23;
        wire t34;
        wire t45;

        assign t12 = (H12 > L12) ? (((lags[6:0] >= L12)&&(lags[6:0] <= H12))
                                    ? 1 : 0) :
                                   (((lags[6:0] >= L12)||(lags[6:0] <= H12))
                                    ? 1 : 0);

        assign t23 = (H23 > L23) ? (((lags[13:7] >= L23)&&(lags[13:7] <= H23))
                                    ? 1 : 0) :
                                   (((lags[13:7] >= L23)||(lags[13:7] <= H23))
                                   ? 1 : 0);

        assign t34 = (H34 > L34) ? (((lags[20:14] >= L34)&&(lags[20:14] <= H34))
                                    ? 1 : 0) :
                                   (((lags[20:14] >= L34)||(lags[20:14] <= H34))
                                    ? 1 : 0);

        assign t45 = (H45 > L45) ? (((lags[27:21] >= L45)&&(lags[27:21] <= H45))
                                    ? 1 : 0) :
                                   (((lags[27:21] >= L45)||(lags[27:21] <= H45))
                                    ? 1 : 0);
```

```
        assign tt12 = ((!valid[0])|(!valid[1])|(t12));
        assign tt23 = ((!valid[1])|(!valid[2])|(t23));
        assign tt34 = ((!valid[2])|(!valid[3])|(t34));
        assign tt45 = ((!valid[3])|(!valid[4])|(t45));

        assign satisfied = (tt12)&(tt23)&(tt34)&(tt45);

        assign angle = ANGLE;

endmodule
```

9.2 Angle Extract

```
 module angleextract(clk, reset_sync, pulsecounter, max_lag, valid, start_extract,
                       done_extract, angle);
          input clk;
          input reset_sync;
          input [3:0] pulsecounter;
          input [27:0] max_lag;   // array of lags
          input [4:0] valid;
          input start_extract;
          output done_extract;

        reg done_extract;
        output [7:0] angle;
        reg [7:0] angle;

        parameter NUM_ANGLES = 11;

        wire [(8*NUM_ANGLES - 1):0] angles;
        wire [(NUM_ANGLES-1):0] satisfied;

        reg [3:0] state;

        parameter S_IDLE = 0;
        parameter S_EXTRACTING = 1;
        parameter S_DECIDING = 2;

        //wire up anglecheckers for each of the NUM_ANGLES angles
        anglechecker angle45(
             .lags(max_lag),
           .valid(valid),
           .satisfied(satisfied[0]),
           .angle(angles[7:0]));
```

```
defparam angle45.ANGLE = 45;
defparam angle45.L12 = 8;//3;
defparam angle45.H12 = 13;//8;
defparam angle45.L23 = 15;//11;
defparam angle45.H23 = 20;//16;
defparam angle45.L34 = 4;//24;
defparam angle45.H34 = 8;//5;
defparam angle45.L45 = 4;//24;
defparam angle45.H45 = 8;//5;

anglechecker angle55(
     .lags(max_lag),
   .valid(valid),
   .satisfied(satisfied[1]),
   .angle(angles[15:8]));

defparam angle55.ANGLE = 55;
defparam angle55.L12 = 19;//13;
defparam angle55.H12 = 0;//19;
defparam angle55.L23 = 2;//19;
defparam angle55.H23 = 6;//2;
defparam angle55.L34 = 13;//7;
defparam angle55.H34 = 16;//12;
defparam angle55.L45 = 15;//9;
defparam angle55.H45 = 19;//15;

anglechecker angle65(
     .lags(max_lag),
   .valid(valid),
   .satisfied(satisfied[2]),
   .angle(angles[23:16]));

defparam angle65.ANGLE = 65;
defparam angle65.L12 = 22;//24;
defparam angle65.H12 = 0;//4;
defparam angle65.L23 = 1;//4;
defparam angle65.H23 = 6;//10;
defparam angle65.L34 = 10;//14;
defparam angle65.H34 = 14;//19;
defparam angle65.L45 = 0;//17;
defparam angle65.H45 = 21;//4;

anglechecker angle75(
     .lags(max_lag),
```

```
    .valid(valid),
    .satisfied(satisfied[3]),
    .angle(angles[31:24]));

defparam angle75.ANGLE = 75;
defparam angle75.L12 = 8;//6;
defparam angle75.H12 = 13;//12;
defparam angle75.L23 = 11;//10;
defparam angle75.H23 = 15;//16;
defparam angle75.L34 = 25;//20;
defparam angle75.H34 = 2;//2;
defparam angle75.L45 = 5;//5;
defparam angle75.H45 = 8;//10;

anglechecker angle85(
     .lags(max_lag),
    .valid(valid),
    .satisfied(satisfied[4]),
    .angle(angles[39:32]));

defparam angle85.ANGLE = 85;
defparam angle85.L12 = 0;//13;
defparam angle85.H12 = 1;//18;
defparam angle85.L23 = 0;//16;
defparam angle85.H23 = 1;//20;
defparam angle85.L34 = 0;//1;
defparam angle85.H34 = 1;//6;
defparam angle85.L45 = 0;//11;
defparam angle85.H45 = 1;//14;

anglechecker angle90(
     .lags(max_lag),
    .valid(valid),
    .satisfied(satisfied[5]),
    .angle(angles[47:40]));

defparam angle90.ANGLE = 90;
defparam angle90.L12 = 18;//19;
defparam angle90.H12 = 23;//6;
defparam angle90.L23 = 19;//20;
defparam angle90.H23 = 2;//6;
defparam angle90.L34 = 5;//6;
defparam angle90.H34 = 9;//14;
defparam angle90.L45 = 16;//19;
defparam angle90.H45 = 19;//4;
```

```
anglechecker angle95(
     .lags(max_lag),
   .valid(valid),
   .satisfied(satisfied[6]),
   .angle(angles[55:48]));

defparam angle95.ANGLE = 95;
defparam angle95.L12 = 7;//8;
defparam angle95.H12 = 9;//14;
defparam angle95.L23 = 5;//7;
defparam angle95.H23 = 10;//12;
defparam angle95.L34 = 12;//14;
defparam angle95.H34 = 18;//18;
defparam angle95.L45 = 3;//5;
defparam angle95.H45 = 10;//15;

anglechecker angle110(
     .lags(max_lag),
   .valid(valid),
   .satisfied(satisfied[7]),
   .angle(angles[63:56]));

defparam angle110.ANGLE = 110;
defparam angle110.L12 = 19;//14;
defparam angle110.H12 = 1;//21;
defparam angle110.L23 = 17;//13;
defparam angle110.H23 = 20;//19;
defparam angle110.L34 = 25;//24;
defparam angle110.H34 = 2;//4;
defparam angle110.L45 = 18;//18;
defparam angle110.H45 = 21;//23;

anglechecker angle125(
     .lags(max_lag),
   .valid(valid),
   .satisfied(satisfied[8]),
   .angle(angles[71:64]));

defparam angle125.ANGLE = 125;
defparam angle125.L12 = 9;//25;
defparam angle125.H12 = 12;//7;
defparam angle125.L23 = 4;//25;
defparam angle125.H23 = 8;//1;
defparam angle125.L34 = 9;//4;
```

```
defparam angle125.H34 = 13;//8;
defparam angle125.L45 = 9;//1;
defparam angle125.H45 = 13;//5;

  anglechecker angle135(
     .lags(max_lag),
   .valid(valid),
   .satisfied(satisfied[9]),
   .angle(angles[79:72]));

defparam angle135.ANGLE = 135;
defparam angle135.L12 = 11;//5;
defparam angle135.H12 = 18;//7;
defparam angle135.L23 = 5;//24;
defparam angle135.H23 = 11;//4;
defparam angle135.L34 = 10;//6;
defparam angle135.H34 = 16;//10;
defparam angle135.L45 = 14;//6;
defparam angle135.H45 = 19;//10;

  anglechecker angle118(
     .lags(max_lag),
   .valid(valid),
   .satisfied(satisfied[10]),
   .angle(angles[87:80]));

defparam angle118.ANGLE = 118;
defparam angle118.L12 = 2;//5;
defparam angle118.H12 = 4;//7;
defparam angle118.L23 = 25;//24;
defparam angle118.H23 = 1;//4;
defparam angle118.L34 = 5;//6;
defparam angle118.H34 = 7;//10;
defparam angle118.L45 = 4;//6;
defparam angle118.H45 = 6;//10;

//integer i; //index for "for" loop
reg onevalid; //one of the angles is valid

always@(posedge clk)
if (!reset_sync) begin
         done_extract <= 0;
         state <= S_IDLE;

         angle = 0;
```

```
            onevalid = 0;
    end
    else case(state)
            S_IDLE:
                if (start_extract) begin
                    state <= S_EXTRACTING;
                    $display("Starting to extract!");
                    angle = 0; //just in case, reset it
                    onevalid = 0;
                    end
                else done_extract <= 0;

            S_EXTRACTING:
                state <= S_DECIDING; // just a clock cycle delay to let the checkers se

            S_DECIDING: begin

                /*for (i = 0; i < NUM_ANGLES; i = i+1) begin
                    if (satisfied[i])
                        if (onevalid) angle = 0;
                        else begin
                            angle = angles[(8*i + 7):(8*i)];
                            onevalid = 1;
                            end
                end

            */ //needs rolling out b/c verilog doesn't like variable ranges

                if (satisfied[0])
                    if (onevalid) angle = 1;
                    else begin
                        angle = angles[7:0];
                        onevalid = 1;
                        end

                if (satisfied[1])
                    if (onevalid) angle = 1;
                    else begin
                        angle = angles[15:8];
                        onevalid = 1;
                        end

                if (satisfied[2])
                    if (onevalid) angle = 1;
                    else begin
```

```
            angle = angles[23:16];
            onevalid = 1;
            end

    if (satisfied[3])
        if (onevalid) angle = 1;
        else begin
            angle = angles[31:24];
            onevalid = 1;
            end

    if (satisfied[4])
        if (onevalid) angle = 1;
        else begin
            angle = angles[39:32];
            onevalid = 1;
            end

    if (satisfied[5])
        if (onevalid) angle = 1;
        else begin
            angle = angles[47:40];
            onevalid = 1;
            end

    if (satisfied[6])
        if (onevalid) angle = 1;
        else begin
            angle = angles[55:48];
            onevalid = 1;
            end

    if (satisfied[7])
        if (onevalid) angle = 1;
        else begin
            angle = angles[63:56];
            onevalid = 1;
            end

    if (satisfied[8])
        if (onevalid) angle = 1;
        else begin
            angle = angles[71:64];
            onevalid = 1;
            end
```

```
                    if (satisfied[9])
                        if (onevalid) angle = 1;
                        else begin
                            angle = angles[79:72];
                            onevalid = 1;
                            end

                    if (satisfied[10])
                        if (onevalid) angle = 1;
                        else begin
                            angle = angles[87:80];
                            onevalid = 1;
                            end

                     done_extract <= 1;
                     state <= S_IDLE;
            end
        endcase

endmodule
```

9.3 BRAM Wrapper

```
module bram_wrapper(reset,
    we1, we2, we3, wer,
    addr1, addr2a, addr2b, addr3, addrra, addrrb,
    din1, din2, din3, dinr,
    controller
    );

    parameter ID = 1;
    parameter ADDR_MSB = 14;
    parameter DATA_MSB = 9;

    parameter GATHER_ID = 0;
    parameter PROC_ID = 1;
    parameter SERIAL_ID = 2;

    input reset;

    input we1, we2, we3;
    input [DATA_MSB:0] din1, din2, din3;
    input [ADDR_MSB:0] addr1, addr2a, addr2b, addr3;
```

```
output wer;
output [DATA_MSB:0] dinr;
output [ADDR_MSB:0] addrra, addrrb;

reg wer;
reg [DATA_MSB:0] dinr;
reg [ADDR_MSB:0] addrra, addrrb;

input [1:0] controller;

always @ ( controller or
                din1 or din2 or din3 or
                addr1 or addr2a or addr2b or addr3 or
                we1 or we2 or we3 or
                reset )

        if ( !reset ) begin
            wer = 0;
            addrra = 0;
            addrrb = 0;
            dinr = 0;
            end
        else    begin
            case ( controller )
                GATHER_ID: begin
                    dinr = din1;
                    wer = we1;
                    addrra = addr1;
                    addrrb = addr1+1; // Addresses addrra and addrrb cannot be the same
                    end

                PROC_ID: begin
                    dinr = din2;
                    wer = we2;
                    addrra = addr2a;
                    addrrb = addr2b;
                    end

                SERIAL_ID: begin
                    dinr = din3;
                    wer = we3;
                    addrra = addr3;
                    addrrb = addr3+1;
                    end
```

```
                    default: begin
                        dinr = din1;
                        wer = we1;
                        addrra = addr1;
                        addrrb = addr1+1;
                        end
                    endcase

                //$display("BRAM wrapper: addrr=",addrr, " we=", wer, " id=", ID);
            end

endmodule
```

9.4 Controller FSM

```
module controller_fsm(clk, reset_sync,
       trans_enable, recv_enable, serial_send_req, recv_done,
         data1_controller, data2_controller,
         pulse1_controller, pulse2_controller,
         td1_controller, td2_controller,
         pixel1_controller, pixel2_controller,
       serial_start, serial_done,
         proc_done, proc_enable,
         preproc_done, preproc_enable,
         conv_start, conv_done );

       input clk, reset_sync, recv_done, serial_send_req, serial_done, proc_done, conv_done
       output [1:0] data1_controller, data2_controller;
       reg [1:0] data1_controller, data2_controller;

       output [1:0] pulse1_controller, pulse2_controller;
       reg [1:0] pulse1_controller, pulse2_controller;

       output [1:0] td1_controller, td2_controller;
       reg [1:0] td1_controller, td2_controller;

       output [1:0] pixel1_controller, pixel2_controller;
       reg [1:0] pixel1_controller, pixel2_controller;

       output serial_start, conv_start;
       reg serial_start, conv_start;

       input preproc_done;
       reg recv_done_buf, preproc_done_buf, proc_done_buf, conv_done_buf;

       output proc_enable;
```

```
reg proc_enable;

  // data ram id's
parameter GATHER_DATA_ID = 0;
parameter PROC_DATA_ID = 1;
parameter SERIAL_DATA_ID = 2;

// pulse ram id's
parameter PREPROC_PULSE_ID = 0;
parameter PROC_PULSE_ID = 1;
parameter SERIAL_PULSE_ID = 2;

  // td ram id's
parameter PROC_TD_ID = 0;
parameter TD_TD_ID = 1;

  // pixel ram id's
parameter TD_PIXEL_ID = 0;
parameter DISP_PIXEL_ID = 1;

output recv_enable, trans_enable, preproc_enable;
reg recv_enable, trans_enable, preproc_enable;

parameter STATE_TRANSMIT_WAIT = 0;
parameter STATE_RECV_PROC = 1;
parameter STATE_SERIAL_WAIT = 2;
parameter STATE_PROC_TRANSMIT = 3;
parameter STATE_PROC_RECV = 4;

parameter TRANSMIT_CYCLES = 4820; // was  28200

reg [3:0] state;
reg [15:0] counter;

always @ ( posedge clk )
        if ( !reset_sync ) begin
          trans_enable <= 1;
          recv_enable <= 0;
          preproc_enable <= 0;
          proc_enable <= 0;
               conv_start <= 0;

          counter <= 0;

          data1_controller <= GATHER_DATA_ID;
```

```
  data2_controller <= PROC_DATA_ID;

   pulse1_controller <= PREPROC_PULSE_ID;
  pulse2_controller <= PROC_PULSE_ID;

  td1_controller <= PROC_TD_ID;
  td2_controller <= TD_TD_ID;

       pixel1_controller <= TD_PIXEL_ID;
  pixel2_controller <= DISP_PIXEL_ID;

  serial_start <= 0;
  state <= STATE_TRANSMIT_WAIT;
  end
else
  case ( state )
    STATE_TRANSMIT_WAIT:
            //assume memory switching already taken care of.
      if ( counter == 0 ) begin
        trans_enable <= 1;
        counter <= counter + 1;
               $display("Control, Starting to transmit");
        end
      else if ( counter == TRANSMIT_CYCLES ) begin
        trans_enable <= 0;

        recv_enable <= 1;    // these two work together
        preproc_enable <= 1;
        proc_enable <= 1;  // this works on the previously written memory
               conv_start <= 1;

        counter <= 0;

        state <= STATE_RECV_PROC;
        serial_start <= 0;

        $display("Control module, done transmitting");
        end
      else begin
        counter <= counter + 1;
        end

    STATE_RECV_PROC: begin
      if (recv_done) begin
                     recv_done_buf <= 1;
```

```
                end
if (preproc_done) begin
                preproc_done_buf <= 1;
                end
if (proc_done) begin
                proc_done_buf <= 1;
                end

   if (conv_done) begin
                conv_done_buf <= 1;
                end

if (   recv_done_buf &&
                preproc_done_buf &&
                proc_done_buf &&
                serial_send_req &&
                conv_done_buf ) begin

     $display("Control module, recv, proc done, wait for processing to
     recv_done_buf <= 0;
     preproc_done_buf <= 0;
     proc_done_buf <= 0;
             conv_done_buf <= 0;

     serial_start <= 0;

     data1_controller <= (data1_controller == GATHER_DATA_ID)
       ? PROC_DATA_ID : GATHER_DATA_ID;
     data2_controller <= (data2_controller == GATHER_DATA_ID)
       ? PROC_DATA_ID : GATHER_DATA_ID;

     pulse1_controller <= (pulse1_controller == PREPROC_PULSE_ID)
       ? PROC_PULSE_ID : PREPROC_PULSE_ID;
     pulse2_controller <= (pulse2_controller == PREPROC_PULSE_ID)
       ? PROC_PULSE_ID : PREPROC_PULSE_ID;

     td1_controller <= (td1_controller == TD_TD_ID) ? PROC_TD_ID : TD_
     td2_controller <= (td2_controller == TD_TD_ID) ? PROC_TD_ID : TD_

                          pixel1_controller <= (pixel1_controller ==
                              ? DISP_PIXEL_ID : TD_PIXEL_ID;
     pixel2_controller <= (pixel2_controller == TD_PIXEL_ID)
       ? DISP_PIXEL_ID : TD_PIXEL_ID;

     state <= STATE_TRANSMIT_WAIT;
```

```
        end

    else if (  recv_done_buf &&
                          preproc_done_buf &&
                          proc_done_buf &&
                          conv_done_buf &&
                          !serial_send_req ) begin      // this is when tl

        $display("Control module, recv, proc done, dumping serial");
        recv_done_buf <= 0;
        preproc_done_buf <= 0;
        proc_done_buf <= 0;
                conv_done_buf <= 0;

        serial_start <= 1;

        data1_controller <= (data1_controller == GATHER_DATA_ID)
          ? GATHER_DATA_ID : SERIAL_DATA_ID;
        data2_controller <= (data2_controller == GATHER_DATA_ID)
          ? GATHER_DATA_ID : SERIAL_DATA_ID;

        pulse1_controller <= (pulse1_controller == PREPROC_PULSE_ID)
          ? PREPROC_PULSE_ID : SERIAL_PULSE_ID;
        pulse2_controller <= (pulse2_controller == PREPROC_PULSE_ID)
          ? PREPROC_PULSE_ID : SERIAL_PULSE_ID;

        state <= STATE_SERIAL_WAIT;
        end

    else begin
        state <= STATE_RECV_PROC;
        recv_enable <= 0;
        proc_enable <= 0;
        preproc_enable <= 0;
                conv_start <= 0;
        end
 end


STATE_SERIAL_WAIT:
    if ( serial_done ) begin
      $display("Control module, serial done");

      data1_controller <= (data1_controller == SERIAL_DATA_ID)
       ? GATHER_DATA_ID : PROC_DATA_ID;
```

```
                    data2_controller <= (data2_controller == SERIAL_DATA_ID)
                     ? GATHER_DATA_ID : PROC_DATA_ID;

                    pulse1_controller <= (pulse1_controller == SERIAL_PULSE_ID)
                     ? PREPROC_PULSE_ID : PROC_PULSE_ID;
                    pulse2_controller <= (pulse2_controller == SERIAL_PULSE_ID)
                     ? PREPROC_PULSE_ID : PROC_PULSE_ID;

                    state <= STATE_TRANSMIT_WAIT;
                    end
                  else
                    serial_start <= 0;

                default: begin
                  trans_enable <= 0;
                  recv_enable <= 0;
                  proc_enable <= 0;
                  preproc_enable <= 0;
                  counter <= 0;
                         serial_start <= 0;
                         conv_start <= 0;

                         data1_controller <= GATHER_DATA_ID;
                         data2_controller <= PROC_DATA_ID;

                         pulse1_controller <= PREPROC_PULSE_ID;
                         pulse2_controller <= PROC_PULSE_ID;

                         td1_controller <= PROC_TD_ID;
                         td2_controller <= TD_TD_ID;

                         pixel1_controller <= TD_PIXEL_ID;
                         pixel2_controller <= DISP_PIXEL_ID;

            state <= STATE_TRANSMIT_WAIT;
            end
               endcase


endmodule
```

9.5 Correlator

```
module correlator(
    clk, reset_sync,
    lag, start_addr, end_addr, start,
```

```
addra, addrb, douta, doutb, done,
sum1, sum2, sum3, sum4);

input clk, reset_sync, start;
input [9:0] douta, doutb;
input [6:0] lag;

input [14:0] start_addr, end_addr;

output [14:0] addra, addrb;
reg [14:0] addra, addrb;

output [25:0] sum1, sum2, sum3, sum4;
reg [25:0] sum1, sum2, sum3, sum4;

output done;
reg done;

reg [1:0] state;

parameter STATE_IDLE = 0;
parameter STATE_MULT_ACC = 1;

always @ (posedge clk)

    if ( !reset_sync ) begin
        done <= 0;
        addra <= 0;
        addrb <= 0;
        sum1 <= 0;
        sum2 <= 0;
        sum3 <= 0;
        sum4 <= 0;
        state <= STATE_IDLE;
        end
    else
        case ( state )

            STATE_IDLE:
                if ( start ) begin
                    addra <= start_addr + lag;
                    addrb <= start_addr;
                    sum1 <= 0; // was 50000
                    sum2 <= 0;
                    sum3 <= 0;
```

```
                sum4 <= 0;
                state <= STATE_MULT_ACC;

                $display("Corr, starting. start_addr = ", start_addr, ", end_addr="
                end
            else
                done <= 0;

        STATE_MULT_ACC: begin
            /*if ( (douta[1:0] == 2'd0) || (doutb[3:2] == 2'd0) )
                sum1 <= sum1; // Don't do anything
            else if (
                ((douta[1] == 1) && (doutb[3] == 1)) ||
                ((douta[1] == 0) && (doutb[3] ==0)) ) // Both negative or positive
                sum1 <= sum1 + 1;
            else
                sum1 <= sum1 - 1;

            if ( (douta[3:2] == 2'd0) || (doutb[5:4] == 2'd0) )
                sum2 <= sum2; // Don't do anything
            else if (
                ((douta[3] == 1) && (doutb[5] == 1)) ||
                ((douta[3] == 0) && (doutb[5] ==0)) ) // Both negative or positive
                sum2 <= sum2 + 1;
            else
                sum2 <= sum2 - 1;

            if ( (douta[5:4] == 2'd0) || (doutb[7:6] == 2'd0) )
                sum3 <= sum3; // Don't do anything
            else if (
                ((douta[5] == 1) && (doutb[7] == 1)) ||
                ((douta[5] == 0) && (doutb[7] ==0)) ) // Both negative or positive
                sum3 <= sum3 + 1;
            else
                sum3 <= sum3 - 1;

            if ( (douta[7:6] == 2'd0) || (doutb[9:8] == 2'd0) )
                sum4 <= sum4; // Don't do anything
            else if (
                ((douta[7] == 1) && (doutb[9] == 1)) ||
                ((douta[7] == 0) && (doutb[9] ==0)) ) // Both negative or positive
                sum4 <= sum4 + 1;
            else
                sum4 <= sum4 - 1;
            */
```

```
                    sum1 <= sum1 + douta[1:0]*doutb[3:2];
                    sum2 <= sum2 + douta[3:2]*doutb[5:4];
                    sum3 <= sum3 + douta[5:4]*doutb[7:6];
                    sum4 <= sum4 + douta[7:6]*doutb[9:8];


                    addra <= addra + 1'd1;
                    addrb <= addrb + 1'd1;

                    if ( addra == end_addr ) begin
                        done <= 1;
                        state <= STATE_IDLE;
                        end
                    end

                default:
                    state <= STATE_IDLE;

            endcase

endmodule
```

9.6 Data Gatherer

```
module data_gather(
    clk, reset_sync,
    enable, adc_data, we, addr, din, recv_done, on_sample, adc_data_buffer);

    input clk, reset_sync, enable;
    input [9:0] adc_data;

    output on_sample;
    reg on_sample;
    output [9:0] adc_data_buffer;
    reg [9:0] adc_data_buffer;

    output we;
    output [9:0] din;
    output [14:0] addr;
    reg we;
    reg [9:0] din;
    reg [14:0] addr;

    output recv_done;
```

```
reg recv_done;

reg [4:0] state;
reg [12:0] counter;

parameter FINAL_ADDR = 25000;//25000;

parameter STATE_IDLE = 0;
parameter STATE_WAIT = 1;
parameter STATE_BUFFER = 2;
parameter STATE_WRITE = 3;

always @ ( posedge clk )
    if ( !reset_sync ) begin
        addr <= 0;
        we <= 0;
        din <= 0;

          recv_done <= 0;
        state <= STATE_IDLE;
          counter <= 0;
        end
    else
        case ( state )

                STATE_IDLE:
                    if ( enable ) begin
                        $display("DataG, start gathering");
                        addr <= 0;
                        state <= STATE_WAIT;
                        end
                else
                        recv_done <= 0;

                STATE_WAIT:
                    if ( counter == 32 ) begin // Corresponds to 1Mhz sampling rate
                        adc_data_buffer <= adc_data;
                        counter <= 0;
                        state <= STATE_BUFFER;
                        on_sample <= 1;
                        //$display("DataG, adc_data=", adc_data);
                        end
                    else
                        counter <= counter + 1;
```

```
                    STATE_BUFFER:   begin
                        din <= adc_data_buffer;
                        we <= 1;
                        state <= STATE_WRITE;
                        on_sample <= 0;
                        end

                    STATE_WRITE:    begin
                        we <= 0;

                        if ( addr == FINAL_ADDR ) begin
                            state <= STATE_IDLE;
                            recv_done <= 1;
                            $display("DataG, done gathering");
                            addr <= 0;
                            end
                        else begin
                            addr <= addr + 1;
                            state <= STATE_WAIT;
                            end
                        end

                default:
                    state <= STATE_IDLE;

                endcase
endmodule
```

9.7 Display

```
module Display_Mod(clk, reset, disp_sel, pixel_cnt, line_cnt, data_in,
                   addr, disp_RGB);

    input clk; //vga_out_pixel_clock
     input reset;
    input [1:0] disp_sel;   // set by two switches on labkit
                             // 00 for front display
                                     // 01 for top display
                                     // 10 and 11 TBD
     input [9:0] pixel_cnt; // from VGA
     input [9:0] line_cnt;  // from VGA
     input [5:0] data_in; // from the RAM that the conversion modules wrote to
     output [16:0] /*[2:0]*/ addr; // addresses data in a RAM that has data for each pixel
    output [23:0] /*[5:0]*/ disp_RGB; // to lab kit

     wire [9:0] /*[2:0]*/ next_pixel_cnt;
```

```
     wire [9:0] /*[1:0]*/ next_line_cnt;

     parameter num_cols_b = 640; //6; // change according to desired resolution
     parameter num_rows_b = 480; //4;   // change according to desired resolution
     parameter num_cols_s = 320; //3;
     parameter num_rows_s = 240;    //2;

     ////////////////// for testing purposes only ////////////////////////
    //wire [2:0] data_in;
     //sim_data_table sim_data_lookup(.clk(clk), .addr(addr), .dout(data_in));
     ////////////////////////////////////////////////////////////////////

    assign addr = (reset) ? 10'b0:
                  ((pixel_cnt > num_cols_b) && (line_cnt <= num_rows_b)) ?
                         (next_line_cnt[9:1]*num_cols_s) :
                  ((pixel_cnt <= num_cols_b - 1) && (line_cnt <= num_rows_b)) ?
                         (line_cnt[9:1]*num_cols_s + next_pixel_cnt[9:1] ) :
                       addr;

     assign next_pixel_cnt = (reset) ? 10'b0 /*3'b0*/:
                             (pixel_cnt >= num_cols_b - 1) ? 10'b0 /*3'b0*/:
                                     pixel_cnt + 1;

     assign next_line_cnt = (reset) ? 10'b0 /*2'b0*/:
                            (line_cnt >= num_rows_b - 1) ? 10'b0 /*2'b0*/:
                            line_cnt + 1;

     assign disp_RGB = (reset) ? 24'b0 :
                       (disp_sel == 2'b00) ? {data_in[5:4], 6'b0, data_in[3:2], 6'b0, data
                             (disp_sel == 2'b01) ? {data_in[5:4], 6'b0, data_in[3:2], 6'b0
                            24'b0 ;

endmodule
```

9.8 Dumper

```
module dumper(
            clk, clock_27mhz, reset_sync,
          start, done,
          rs232_cts, rs232_rxd, rs232_txd, rs232_rts,
          data_addr, data_dout,
          pulse_addr, pulse_dout,    pulsecounter,
          max_lag1, max_lag2, max_lag3, max_lag4,
             max_lag_sum1, max_lag_sum2, max_lag_sum3, max_lag_sum4,
             corr_start_addr,
             addr_mode,
```

```
    angle, min_dist
);

  input clk, reset_sync, clock_27mhz;
  input [6:0] max_lag1, max_lag2, max_lag3, max_lag4;
  input [25:0] max_lag_sum1, max_lag_sum2, max_lag_sum3, max_lag_sum4;

  input [3:0] pulsecounter;
  input [14:0] corr_start_addr;

  input start;
  input rs232_cts;
  input rs232_rxd;
  output rs232_txd, rs232_rts;

  output done;
  reg done;

  input[9:0] data_dout;
output [14:0] data_addr;
reg [14:0] data_addr;

input [34:0] pulse_dout;
output [3:0] pulse_addr;
reg [3:0] pulse_addr;

reg push;
wire [7:0] dummy_recv;

  input [7:0] angle;
  input addr_mode;
  input [12:0] min_dist;

parameter S_IDLE = 0;         // in between dumps
parameter S_DATAWAIT = 1;     // wait for dout to become valid at new address
parameter S_WRITE = 2;        // force out data


parameter S_PULSEWRITE0 = 3;      // send number of pulses
parameter S_PULSEWRITE1 = 4;     // force out pulsedata begin
parameter S_PULSEWRITE2 = 5;      // force out pulsedata end
parameter S_PULSEWRITE3 = 6;     // force out pulsedata valids

parameter S_LAGWRITE1 = 7;    //write the different lags
parameter S_LAGWRITE2 = 9;
```

```
parameter S_LAGWRITE3 = 8;
parameter S_LAGWRITE4 = 12;



parameter BIG_FINAL_DATA_ADDR = 9;
    parameter SHORT_FINAL_DATA_ADDR = 24999;

parameter BAUD_RATE_DELAY = 5000; // was 5000

  wire [14:0] final_addr;
  assign final_addr = addr_mode ?  SHORT_FINAL_DATA_ADDR : BIG_FINAL_DATA_ADDR;

reg[4:0] state;
reg[15:0] counter;
reg[8:0] sub_counter;

reg [7:0] serial_out;

rs232 rs232i (
   .reset(reset_sync),
   .clock_27mhz(clock_27mhz),
   .rxd(rs232_rxd),
   .cts(rs232_cts),
   .txd(rs232_txd),
   .rts(rs232_rts),
   .rx_data(dummy_recv),
   .tx_data(serial_out),
   .tx_start(push)
   );

wire [14:0] start_addr, end_addr;
wire [4:0] valid;

assign start_addr = pulse_dout[14:0];
assign end_addr = pulse_dout[29:15];
assign valid = pulse_dout[34:30];

 always @ (posedge clk)
        if ( !reset_sync ) begin
            data_addr <= 0;
            pulse_addr <= 0;

            counter <= 0;
            sub_counter <= 0;
```

```
    done <= 0;

          state <= S_IDLE;
    serial_out <= 0;
          push <= 0;
    end
else
 case (state)
     S_IDLE: begin
      push <= 0;
      if ( start ) begin
         $display("Dumper, starting");
               $display("Dumper, pulsecounter=", pulsecounter);
         state <= S_DATAWAIT;
         counter <= 0;
               sub_counter <= 0;

         data_addr <= 0;
               pulse_addr <= 0;
         end
      else
        done <= 0;
     end

     S_DATAWAIT: begin
      push <= 0;
      counter <= counter + 1;
      if (counter == 3) begin
        //$display("Dumper, Valid serial_out =", serial_out, ", sub_counter="

        push <= 1;
        counter <= 0;
        state <= S_WRITE;
        end
      else if ( counter == 2 ) begin //dout is now valid
        case ( sub_counter )
              0: serial_out <= {6'd0, data_dout[1:0]};
              1: serial_out <= {6'd0, data_dout[3:2]};
              2: serial_out <= {6'd0, data_dout[5:4]};
              3: serial_out <= {6'd0, data_dout[7:6]};
              4: serial_out <= {6'd0, data_dout[9:8]};

                      // Other things of data
                      5: serial_out <= {1'b0 , max_lag1};
                      6: serial_out <=   {1'b0 , max_lag2};
```

```
                        7: serial_out <=   {1'b0 , max_lag3};
                        8: serial_out <=   {1'b0 , max_lag4};

                        // send max sums
                        9: serial_out <= angle;//max_lag_sum1[7:0];
                        10: serial_out <= 8'd0;//max_lag_sum1[15:8];
                        11: serial_out <= min_dist[7:0];//max_lag_sum2[7:0];
                        12: serial_out <= {3'd0, min_dist[12:8]};//max_lag_sum
                        13: serial_out <= max_lag_sum3[7:0];
                        14: serial_out <= max_lag_sum3[15:8];
                        15: serial_out <= max_lag_sum4[7:0];
                        16: serial_out <= max_lag_sum4[15:8];

                        17: serial_out <= corr_start_addr[7:0];
                        18: serial_out <= {2'd0, corr_start_addr[14:8]};

                        19: serial_out <=  pulsecounter;
                        20: serial_out <= pulse_dout[34:27];
                        21: serial_out <= pulse_dout[19:12];
                        22: serial_out <= {3'b0, pulse_dout[4:0]};
                default: serial_out <= 0;
           endcase
         end
        end

        S_WRITE: begin
         push <= 0;
         counter <= counter + 1;
         if (counter == BAUD_RATE_DELAY) begin
               counter <= 0;
                      //$display("Dumper, data address: ", data_addr);
               if ((data_addr == final_addr) && (sub_counter >= 4) ) begin
                 // subcounter >= 4, sending extra data
                           if ( sub_counter == 22 )
                               if ( pulse_addr == pulsecounter ) begin
                           state <= S_IDLE;
                                   done <= 1;
                                   end
                               else begin
                                   pulse_addr <= pulse_addr + 1;
                                   state <= S_DATAWAIT;
                                   sub_counter <= 20;
                                   end
                           else begin
                               sub_counter <= sub_counter + 1;
```

```
                                        state <= S_DATAWAIT;
                                        end
                            end

                                    // Send normal data
                           else begin
                              state <= S_DATAWAIT;

                              if ( sub_counter == 4 ) begin
                                data_addr <= data_addr + 1;
                                sub_counter <= 0;
                                end
                              else
                                sub_counter <= sub_counter + 1;
                                end
                     end
                         end



                    default:
                        state <= S_IDLE;

                endcase

endmodule
```

9.9 Labkit

```
   wire pclk, pixel_clock;

    DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

    //assign pixel_clock = clock_27mhz;

    wire reset_sync, serial_req_sync, angle_button_sync;

    // debouncing ////////////////////////////////////////////////////////////////

    // For debugging
```

```
//assign reset_sync = button0;
//assign serial_req_sync = button1;

debounce reset_debounce(
    .reset(1'd0),
    .clock(pixel_clock),
    .noisy(button0),
    .clean(reset_sync));

debounce serial_req_debounce (
    .reset(!reset_sync),
    .clock(pixel_clock),
    .noisy(button1),
    .clean(serial_req_sync));

debounce angle_debounce (
    .reset(!reset_sync),
    .clock(pixel_clock),
    .noisy(button2),
    .clean(angle_button_sync));

// td bram control ////////////////////////////////////////////////////////////////

// Actual we, din, dout used by memories
wire [15:0] td_mem1_doutr, td_mem2_doutr;

wire td_mem1_wer, td_mem2_wer;
wire [15:0] td_mem1_dinr, td_mem2_dinr;
wire [3:0] td_mem1_addrr, td_mem2_addrr;

// Muxed dout
wire [15:0] td_mem_doutr;

// Local copies of addr, dout, din, we
wire [3:0] td_mem_addr1, td_mem_addr2;
wire [15:0] td_mem_din1, td_mem_din2;
wire td_mem_we1, td_mem_we2;
wire [1:0] td_mem1_controller, td_mem2_controller;

// processor must write, others should leave we low
assign td_mem_we2 = 0;
assign td_mem_din2 = 6'd0;

// muxing of different memories
assign td_mem_doutr = (td_mem1_controller == 0) ? td_mem2_doutr : td_mem1_doutr;
```

```
td_mem1 td_mem1i (
    .addr(td_mem1_addrr),
    .clk(pixel_clock),
    .din(td_mem1_dinr),
    .dout(td_mem1_doutr),
    .we(td_mem1_wer)
);

wire [3:0] dummy1_addrrb, dummy2_addrrb;

bram_wrapper td_mem1_bram(
    .reset(reset_sync),
    .we1(td_mem_we1),
    .we2(td_mem_we2),
    .we3(1'd0),
    .wer(td_mem1_wer),
    .addr1(td_mem_addr1),
    .addr2a(td_mem_addr2),
    .addr2b(td_mem_addr2),
    .addr3(4'd0),
    .addrra(td_mem1_addrr),
    .addrrb(dummy1_addrrb),
    .din1(td_mem_din1),
    .din2(td_mem_din2),
    .din3(16'd0),
    .dinr(td_mem1_dinr),
    .controller(td_mem1_controller)
);

defparam td_mem1_bram.ADDR_MSB = 3;
defparam td_mem1_bram.DATA_MSB = 15;

td_mem2 td_mem2i (
    .addr(td_mem2_addrr),
    .clk(pixel_clock),
    .din(td_mem2_dinr),
    .dout(td_mem2_doutr),
    .we(td_mem2_wer)
);

bram_wrapper td_mem2_bram(
    .reset(reset_sync),
    .we1(td_mem_we1),
    .we2(td_mem_we2),
```

```
        .we3(1'd0),
        .wer(td_mem2_wer),
        .addr1(td_mem_addr1),
        .addr2a(td_mem_addr2),
        .addr2b(td_mem_addr2),
        .addr3(4'd0),
        .addrra(td_mem2_addrr),
        .addrrb(dummy2_addrrb),
        .din1(td_mem_din1),
        .din2(td_mem_din2),
        .din3(16'd0),
        .dinr(td_mem2_dinr),
        .controller(td_mem2_controller)
    );

    defparam td_mem2_bram.ADDR_MSB = 3;
    defparam td_mem2_bram.DATA_MSB = 15;

    // pixel bram control ////////////////////////////////////////////////////////////

    // Actual we, din, dout used by memories
    wire [5:0] pixel_mem1_doutr, pixel_mem2_doutr;

    wire pixel_mem1_wer, pixel_mem2_wer;
    wire [5:0] pixel_mem1_dinr, pixel_mem2_dinr;
    wire [16:0] pixel_mem1_addrr, pixel_mem2_addrr;

    // Muxed dout
    wire [5:0] pixel_mem_doutr;

    // Local copies of addr, dout, din, we
    wire [16:0] pixel_mem_addr1, pixel_mem_addr2;
    wire [5:0] pixel_mem_din1, pixel_mem_din2;
    wire pixel_mem_we1, pixel_mem_we2;
    wire [1:0] pixel_mem1_controller, pixel_mem2_controller;

    // preprocessor must write, others should leave we low
    assign pixel_mem_we2 = 0;
    assign pixel_mem_din2 = 6'd0;

    // muxing of different memories
    assign pixel_mem_doutr = (pixel_mem1_controller == 0) ? pixel_mem2_doutr : pixel_mem1_do

    pixel_mem1 pixel_mem1i (
        .addr(pixel_mem1_addrr),
```

```
        .clk(pixel_clock),
        .din(pixel_mem1_dinr),
        .dout(pixel_mem1_doutr),
        .we(pixel_mem1_wer)
    );

    wire [16:0] dummy3_addrrb, dummy4_addrrb;

    bram_wrapper pixel_mem1_bram(
        .reset(reset_sync),
        .we1(pixel_mem_we1),
        .we2(pixel_mem_we2),
        .we3(1'd0),
        .wer(pixel_mem1_wer),
        .addr1(pixel_mem_addr1),
        .addr2a(pixel_mem_addr2),
        .addr2b(pixel_mem_addr2),
        .addr3(17'd0),
        .addrra(pixel_mem1_addrr),
        .addrrb(dummy3_addrrb),
        .din1(pixel_mem_din1),
        .din2(pixel_mem_din2),
        .din3(6'd0),
        .dinr(pixel_mem1_dinr),
        .controller(pixel_mem1_controller)
    );

    defparam pixel_mem1_bram.ADDR_MSB = 16;
    defparam pixel_mem1_bram.DATA_MSB = 5;

    pixel_mem2 pixel_mem2i (
        .addr(pixel_mem2_addrr),
        .clk(pixel_clock),
        .din(pixel_mem2_dinr),
        .dout(pixel_mem2_doutr),
        .we(pixel_mem2_wer)
    );

    bram_wrapper pixel_mem2_bram(
        .reset(reset_sync),
        .we1(pixel_mem_we1),
        .we2(pixel_mem_we2),
        .we3(1'd0),
        .wer(pixel_mem2_wer),
        .addr1(pixel_mem_addr1),
```

```
        .addr2a(pixel_mem_addr2),
        .addr2b(pixel_mem_addr2),
        .addr3(17'd0),
        .addrra(pixel_mem2_addrr),
        .addrrb(dummy4_addrrb),
        .din1(pixel_mem_din1),
        .din2(pixel_mem_din2),
        .din3(6'd0),
        .dinr(pixel_mem2_dinr),
        .controller(pixel_mem2_controller)
    );

    defparam pixel_mem2_bram.ADDR_MSB = 16;
    defparam pixel_mem2_bram.DATA_MSB = 5;

    // pulse bram control ////////////////////////////////////////////////////////////

    // Actual we, din, dout used by memories
    wire [34:0] pulse_mem1_doutr, pulse_mem2_doutr;

    wire pulse_mem1_wer, pulse_mem2_wer;
    wire [34:0] pulse_mem1_dinr, pulse_mem2_dinr;
    wire [3:0] pulse_mem1_addrr, pulse_mem2_addrr;

    // Muxed dout
    wire [34:0] pulse_mem_doutr;

    // Local copies of addr, dout, din, we
    wire [3:0] pulse_mem_addr1, pulse_mem_addr2, pulse_mem_addr3;
    wire [34:0] pulse_mem_din1, pulse_mem_din2, pulse_mem_din3;
    wire pulse_mem_we1, pulse_mem_we2, pulse_mem_we3;
    wire [1:0] pulse_mem1_controller, pulse_mem2_controller;

    // preprocessor must write, others should leave we low
    assign pulse_mem_we2 = 0;
    assign pulse_mem_we3 = 0;
    assign pulse_mem_din2 = 35'd0;
    assign pulse_mem_din3 = 35'd0;

    // muxing of different memories
    assign pulse_mem_doutr = (pulse_mem1_controller == 0) ? pulse_mem2_doutr : pulse_mem1_do

    pulse_mem1 pulse_mem1i (
        .addr(pulse_mem1_addrr),
        .clk(pixel_clock),
```

```
        .din(pulse_mem1_dinr),
        .dout(pulse_mem1_doutr),
        .we(pulse_mem1_wer)
    );

    wire [3:0] dummy5_addrrb, dummy6_addrrb;

    bram_wrapper pulse_mem1_bram(
        .reset(reset_sync),
        .we1(pulse_mem_we1),
        .we2(pulse_mem_we2),
        .we3(pulse_mem_we3),
        .wer(pulse_mem1_wer),
        .addr1(pulse_mem_addr1),
        .addr2a(pulse_mem_addr2),
        .addr2b(pulse_mem_addr2),
        .addr3(pulse_mem_addr3),
        .addrra(pulse_mem1_addrr),
        .addrrb(dummy5_addrrb),
        .din1(pulse_mem_din1),
        .din2(pulse_mem_din2),
        .din3(pulse_mem_din3),
        .dinr(pulse_mem1_dinr),
        .controller(pulse_mem1_controller)
    );

    defparam pulse_mem1_bram.ADDR_MSB = 3;
    defparam pulse_mem1_bram.DATA_MSB = 34;

    pulse_mem2 pulse_mem2i (
        .addr(pulse_mem2_addrr),
        .clk(pixel_clock),
        .din(pulse_mem2_dinr),
        .dout(pulse_mem2_doutr),
        .we(pulse_mem2_wer)
    );

    bram_wrapper pulse_mem2_bram(
        .reset(reset_sync),
        .we1(pulse_mem_we1),
        .we2(pulse_mem_we2),
        .we3(pulse_mem_we3),
        .wer(pulse_mem2_wer),
        .addr1(pulse_mem_addr1),
        .addr2a(pulse_mem_addr2),
```

```
        .addr2b(pulse_mem_addr2),
        .addr3(pulse_mem_addr3),
        .addrra(pulse_mem2_addrr),
        .addrrb(dummy6_addrrb),
        .din1(pulse_mem_din1),
        .din2(pulse_mem_din2),
        .din3(pulse_mem_din3),
        .dinr(pulse_mem2_dinr),
        .controller(pulse_mem2_controller)
    );

    defparam pulse_mem2_bram.ADDR_MSB = 3;
    defparam pulse_mem2_bram.DATA_MSB = 34;

    // data bram control ////////////////////////////////////////////////////////////

    // Actual we, din, dout used by memories
    wire [9:0] data_mem1_doutra, data_mem1_doutrb, data_mem2_doutra, data_mem2_doutrb;

    wire data_mem1_wer, data_mem2_wer;
    wire [9:0] data_mem1_dinr, data_mem2_dinr;
    wire [14:0] data_mem1_addrra, data_mem1_addrrb, data_mem2_addrra, data_mem2_addrrb;

    // Muxed douts
    wire [9:0] data_mem_doutra, data_mem_doutrb;

    // Local copies of addr, dout, din, we
    wire [14:0] data_mem_addr1, data_mem_addr2a, data_mem_addr2b, data_mem_addr3;
    wire [9:0] data_mem_din1, data_mem_din2, data_mem_din3;
    wire data_mem_we1, data_mem_we2, data_mem_we3;
    wire [1:0] data_mem1_controller, data_mem2_controller;

    // Data Gather must write, others should leave we low
    assign data_mem_we2 = 0;
    assign data_mem_we3 = 0;
    assign data_mem_din2 = 10'd0;
    assign data_mem_din3 = 10'd0;

    // muxing of different memories
    assign data_mem_doutra = (data_mem1_controller == 0) ? data_mem2_doutra : data_mem1_dou
    assign data_mem_doutrb = (data_mem1_controller == 0) ? data_mem2_doutrb : data_mem1_dou

    // data memory 1

    wire [9:0] dummy1_dinb;
```

```
data_mem1 data_mem1i (
    .addra(data_mem1_addrra),
    .addrb(data_mem1_addrrb),
    .clka(pixel_clock),
    .clkb(pixel_clock),
    .dina(data_mem1_dinr),
    .dinb(dummy1_dinb),
    .douta(data_mem1_doutra),
    .doutb(data_mem1_doutrb),
    .wea(data_mem1_wer),
    .web(1'd0) // Never write through second channel
);

bram_wrapper data_mem1_bram(
    .reset(reset_sync),
    .we1(data_mem_we1),
    .we2(data_mem_we2),
    .we3(data_mem_we3),
    .wer(data_mem1_wer),
    .addr1(data_mem_addr1),
    .addr2a(data_mem_addr2a),
    .addr2b(data_mem_addr2b),
    .addr3(data_mem_addr3),
    .addrra(data_mem1_addrra),
    .addrrb(data_mem1_addrrb),
    .din1(data_mem_din1),
    .din2(data_mem_din2),
    .din3(data_mem_din3),
    .dinr(data_mem1_dinr),
    .controller(data_mem1_controller)
);

defparam data_mem1_bram.ID = 1;
defparam data_mem1_bram.ADDR_MSB = 14;
defparam data_mem1_bram.DATA_MSB = 9;

// data memory 2

wire [9:0] dummy2_dinb;

data_mem2 data_mem2i (
    .addra(data_mem2_addrra),
    .addrb(data_mem2_addrrb),
    .clka(pixel_clock),
```

```
        .clkb(pixel_clock),
        .dina(data_mem2_dinr),
        .dinb(dummy2_dinb),
        .douta(data_mem2_doutra),
        .doutb(data_mem2_doutrb),
        .wea(data_mem2_wer),
        .web(1'd0) // Never write through second channel
    );

    bram_wrapper data_mem2_bram(
        .reset(reset_sync),
        .we1(data_mem_we1),
        .we2(data_mem_we2),
        .we3(data_mem_we3),
        .wer(data_mem2_wer),
        .addr1(data_mem_addr1),
        .addr2a(data_mem_addr2a),
        .addr2b(data_mem_addr2b),
        .addr3(data_mem_addr3),
        .addrra(data_mem2_addrra),
        .addrrb(data_mem2_addrrb),
        .din1(data_mem_din1),
        .din2(data_mem_din2),
        .din3(data_mem_din3),
        .dinr(data_mem2_dinr),
        .controller(data_mem2_controller)
    );

    defparam data_mem2_bram.ID = 2;
    defparam data_mem2_bram.ADDR_MSB = 14;
    defparam data_mem2_bram.DATA_MSB = 9;

    // processor ////////////////////////////////////////////////

    // debugging lag outputs
    wire [6:0] max_lag1, max_lag2, max_lag3, max_lag4;
    wire [25:0] max_lag_sum1, max_lag_sum2, max_lag_sum3, max_lag_sum4;
    wire [14:0] corr_start_addr;

    wire proc_start, proc_done;
    wire [3:0] pulsecounter, prev_pulsecounter;

    wire [7:0] angle;

    reg [7:0] angle_chooser;
```

```
reg angle_button_state;

always @ (posedge pixel_clock)
    if ( !reset_sync) begin
        angle_chooser <= 0;
        angle_button_state <= 0;
        end
    else
        case(angle_button_state)
            0:
                if (!angle_button_sync ) begin
                    angle_chooser <= angle_chooser + 1;
                    angle_button_state <= 1;
                    end

            1:
                if ( angle_button_sync )
                    angle_button_state <= 0;
        endcase


wire [7:0] newangle; //immediate angle - useful for debugging angle identification
wire [12:0] min_dist;

processor processori (
    .clk(pixel_clock),
    .reset_sync(reset_sync),
    .done(proc_done),
    .start(proc_start),
    .data_addra(data_mem_addr2a),
    .data_addrb(data_mem_addr2b),
    .data_douta(data_mem_doutra),
    .data_doutb(data_mem_doutrb),
    .pulse_addr(pulse_mem_addr2),
    .pulse_dout(pulse_mem_doutr),
    .max_lag1(max_lag1),
    .max_lag2(max_lag2),
    .max_lag3(max_lag3),
    .max_lag4(max_lag4),
    .max_lag_sum1(max_lag_sum1),
    .max_lag_sum2(max_lag_sum2),
    .max_lag_sum3(max_lag_sum3),
    .max_lag_sum4(max_lag_sum4),
    .start_addr(corr_start_addr),
    .pulsecounter(prev_pulsecounter),
```

```
        .angle(angle),
        .newangle(newangle),
        .td_we(td_mem_we1),
        .td_addr(td_mem_addr1),
        .td_din(td_mem_din1),
        .angle_chooser(angle_chooser),
        .angle_valid_thresh(switch[7:4]),
        .min_dist(min_dist)
        );

    // serial module //////////////////////////////////////////////

    wire serial_start, serial_done;

    dumper dumper1 (
        .clk(pixel_clock),
        .clock_27mhz(clock_27mhz),
        .reset_sync(reset_sync),
        .start(serial_start),
        .rs232_cts(rs232_cts),
        .rs232_rxd(rs232_rxd),
        .rs232_txd(rs232_txd),
        .rs232_rts(rs232_rts),
        .data_dout(data_mem_doutra),
        .data_addr(data_mem_addr3),
        .pulse_dout(pulse_mem_doutr),
        .pulse_addr(pulse_mem_addr3),
        .max_lag1(max_lag1),
        .max_lag2(max_lag2),
        .max_lag3(max_lag3),
        .max_lag4(max_lag4),
        .max_lag_sum1(max_lag_sum1),
        .max_lag_sum2(max_lag_sum2),
        .max_lag_sum3(max_lag_sum3),
        .max_lag_sum4(max_lag_sum4),
        .corr_start_addr(corr_start_addr),
        .done(serial_done),
        .pulsecounter(prev_pulsecounter),
        .addr_mode(switch[0]),
        .angle(newangle),
        .min_dist(min_dist)
    );

    // data gathering module //////////////////////////////////////
   assign user1 = 32'hZ;
```

```
assign user2 = 32'hZ;
assign user3 = 32'hZ;

assign user4[9:0] = 10'hZ;
 wire recv_enable, recv_done, on_sample;
 wire [9:0] adc_data_buffer;

 assign led[3:0] = ~pulsecounter;
 assign led[7:4] = ~prev_pulsecounter;

 data_gather data_gatheri (
     .clk(pixel_clock),
     .reset_sync(reset_sync),
     .enable(recv_enable),
     .adc_data(user4[9:0]),
     .we(data_mem_we1),
     .addr(data_mem_addr1),
     .din(data_mem_din1),
     .recv_done(recv_done),
     .on_sample(on_sample),
     .adc_data_buffer(adc_data_buffer)
 );

 mic_controller mic1(
     .clk(pixel_clock),
     .reset_sync(reset_sync),
     .on_sample(on_sample),
     .mic_stopper(user4[11]),
     .mic_val(adc_data_buffer[1:0]));

 // preprocess ////////////////////////////////////////////////////

 wire preproc_done, preproc_start;
 reg recv_done_buffer;

 always @ (recv_done or preproc_done) begin
     if ( recv_done == 1 )
         recv_done_buffer = 1;

     if ( preproc_done == 1 )
         recv_done_buffer = 0;

     end

 preprocess preprocessi(
```

```
        .clk(pixel_clock),
        .reset_sync(reset_sync),
        .data(adc_data_buffer),
        .time_index(data_mem_addr1),
        .on_sample(on_sample),
        .pulse_addr(pulse_mem_addr1),
        .pulse_din(pulse_mem_din1),
        .pulse_we(pulse_mem_we1),
        .done(preproc_done),
        .start(preproc_start),
        .recv_done(recv_done_buffer),
        .pulsecounter(pulsecounter),
        .prev_pulsecounter(prev_pulsecounter)
    );

    // transmitter module ////////////////////////////////////////

    wire trans_enable;

    transmit transmiti(
        .clk(pixel_clock),
        .reset_sync(reset_sync),
        .enable(trans_enable),
        .transmit_bit(user4[10]));

    // controller fsm ////////////////////////////////////////////////////

    wire conv_start, conv_done;
   assign vga_out_pixel_clock = ~pixel_clock;

    wire [9:0] pixel_count, line_count;
    wire hblank, vblank;
    wire [23:0] rgb_out;

    assign is_top_mode = (switch[2:1] == 2'd1);

    // Muxing for start, done signals
    assign top_conv_start = is_top_mode ? conv_start : 0;
    assign front_conv_start = is_top_mode ? 0 : conv_start;

    wire top_conv_done, front_conv_done;
    assign conv_done = is_top_mode ?  top_conv_done : front_conv_done;

    // Muxing for td mem
    wire [3:0] td_mem_addr2_top, td_mem_addr2_front;
```

```
assign td_mem_addr2 = is_top_mode ? td_mem_addr2_top : td_mem_addr2_front;

// Muxing for pixel mem
wire [16:0] pixel_mem_addr1_top, pixel_mem_addr1_front;
assign pixel_mem_addr1 = is_top_mode ? pixel_mem_addr1_top : pixel_mem_addr1_front;

wire [5:0] pixel_mem_din1_top, pixel_mem_din1_front;
assign pixel_mem_din1 = is_top_mode ? pixel_mem_din1_top : pixel_mem_din1_front;

wire pixel_mem_we1_top, pixel_mem_we1_front;
assign pixel_mem_we1 = is_top_mode ? pixel_mem_we1_top : pixel_mem_we1_front;

t_d_conv_front f_converter(
   .clk(pixel_clock),
    .reset(~reset_sync), // different standards
    .max_td_addr(4'd0),
    .start_fwrite(front_conv_start),
    .t_d_pair(td_mem_doutr),
    .td_addr(td_mem_addr2_front),
    .pd_addr(pixel_mem_addr1_front),
    .pd_we(pixel_mem_we1_front),
    .pix_data(pixel_mem_din1_front),
    .done(front_conv_done));

t_d_conv_top t_converter(
    .clk(pixel_clock),
    .reset(~reset_sync),
  .max_td_addr(4'd0),
    .start_twrite(top_conv_start),
    .td_addr(td_mem_addr2_top),
    .t_d_pair(td_mem_doutr),
    .pd_addr(pixel_mem_addr1_top),
    .pd_we(pixel_mem_we1_top),
    .pix_data(pixel_mem_din1_top),
    .done(top_conv_done));

Display_Mod disp_mod(
    .clk(vga_out_pixel_clock),
    .reset(~reset_sync),
    .disp_sel(switch[2:1]),
    .pixel_cnt(pixel_count),
  .line_cnt(line_count),
    .data_in(pixel_mem_doutr),
    .addr(pixel_mem_addr2),
    .disp_RGB(rgb_out));
```

```
 // Vga stuff ////////////////////////////////////////////////////////////////

 VGA_Mod vga_control(
     .clk(pixel_clock),
     .reset(reset_sync),
     .pixel_count(pixel_count),
     .line_count(line_count),
     .hsync(hsync),
     .vsync(vsync),
     .hblank(hblank),
     .vblank(vblank));

 reg [1:0] hsync_delay = 2'b11;
 reg [1:0] vsync_delay = 2'b11;

 reg vga_out_hsync, vga_out_vsync;

always @(posedge pixel_clock) begin
    vga_out_hsync <= hsync_delay[0];
    hsync_delay[1] <= hsync_delay[0];
    hsync_delay[0] <= hsync;

    vga_out_vsync <= vsync_delay[0];
    vsync_delay[1] <= vsync_delay[0];
    vsync_delay[0] <= vsync;
      end

assign vga_out_red = rgb_out[23:16];
assign vga_out_green = rgb_out[15:8];
assign vga_out_blue = rgb_out[7:0];

  assign vga_out_blank_b = (hblank & vblank);
assign vga_out_sync_b = 1'b1;

 controller_fsm controller_fsm1(
     .clk(pixel_clock),
     .reset_sync(reset_sync),
   .trans_enable(trans_enable),
     .recv_enable(recv_enable),
     .serial_send_req(serial_req_sync),
     .recv_done(recv_done),
     .data1_controller(data_mem1_controller),
     .data2_controller(data_mem2_controller),
```

```
        .pulse1_controller(pulse_mem1_controller),
        .pulse2_controller(pulse_mem2_controller),
        .td1_controller(td_mem1_controller),
        .td2_controller(td_mem2_controller),
        .pixel1_controller(pixel_mem1_controller),
        .pixel2_controller(pixel_mem2_controller),
    .serial_start(serial_start),
        .serial_done(serial_done),
        .proc_done(proc_done),
        .proc_enable(proc_start),
        .preproc_done(preproc_done),
        .preproc_enable(preproc_start),
        .conv_start(conv_start),
        .conv_done(conv_done)
        );
endmodule
```

9.10 Max Lag Finder

```
module max_lag_finder(
    clk, reset_sync, start, done,
    data_addra, data_addrb, data_douta, data_doutb,
    start_addr, end_addr,
    max_lag1, max_lag2, max_lag3, max_lag4,
    max_lag_sum1, max_lag_sum2, max_lag_sum3, max_lag_sum4);

    input clk, reset_sync, start;

    output [6:0] max_lag1, max_lag2, max_lag3, max_lag4;
    reg [6:0] max_lag1, max_lag2, max_lag3, max_lag4;
    output [25:0] max_lag_sum1, max_lag_sum2, max_lag_sum3, max_lag_sum4;
    reg [25:0] max_lag_sum1, max_lag_sum2, max_lag_sum3, max_lag_sum4;

    output done;
    reg done;

    // Create the correlator

    input [9:0] data_douta, data_doutb;
    output [14:0] data_addra, data_addrb;

    reg corr_start;
    wire corr_done;

    wire [25:0] sum1, sum2, sum3, sum4;
    reg [6:0] cur_lag;
```

```
input [14:0] start_addr;
input [14:0] end_addr;

correlator correlatori (
    .clk(clk),
    .reset_sync(reset_sync),
    .lag(cur_lag),
    .start_addr(start_addr),
    .end_addr(end_addr),
    .start(corr_start),
    .addra(data_addra),
    .addrb(data_addrb),
    .douta(data_douta),
    .doutb(data_doutb),
    .done(corr_done),
    .sum1(sum1),
    .sum2(sum2),
    .sum3(sum3),
    .sum4(sum4)
);


// FSM

parameter MAX_LAG = 25;

parameter STATE_IDLE = 0;
parameter STATE_LAG_LOOP = 1;
parameter STATE_FALSE_START = 2;

reg [3:0] state;

always @ (posedge clk)
    if ( !reset_sync ) begin
        done <= 0;
        max_lag1 <= 11; // special value for no max found???
        max_lag2 <= 12;
        max_lag3 <= 13;
        max_lag4 <= 14;

        max_lag_sum1 <= 11;
        max_lag_sum2 <= 12;
        max_lag_sum3 <= 13;
        max_lag_sum4 <= 14;
```

```
        state <= STATE_IDLE;
        end
    else
        case ( state )
            STATE_IDLE:
                if ( start ) begin
                    max_lag1 <= 27; // special value for no max found???
                    max_lag2 <= 28;
                    max_lag3 <= 29;
                    max_lag4 <= 30;

                    max_lag_sum1 <= 0;
                    max_lag_sum2 <= 0;
                    max_lag_sum3 <= 0;
                    max_lag_sum4 <= 0;

                    if ( end_addr <= start_addr ) begin
                        done <= 1;
                        state <= STATE_FALSE_START;
                        end
                    else begin
                        $display("Lagger starting");

                        cur_lag <= 0;
                        corr_start <= 1;
                        state <= STATE_LAG_LOOP;
                        end
                    end
                else
                    done <= 0;

            STATE_FALSE_START:
                state <= STATE_IDLE;

            STATE_LAG_LOOP:
                if ( corr_done ) begin
                    $display("lag=", cur_lag, " sum4=", sum4);

                    if ( sum1 > max_lag_sum1 ) begin
                        max_lag1 <= cur_lag;
                        max_lag_sum1 <= sum1;
                        end

                    if ( sum2 > max_lag_sum2 ) begin
```

```
                            max_lag2 <= cur_lag;
                            max_lag_sum2 <= sum2;
                            end

                        if ( sum3 > max_lag_sum3 ) begin
                            max_lag3 <= cur_lag;
                            max_lag_sum3 <= sum3;
                            end

                        if ( sum4 > max_lag_sum4 ) begin
                            max_lag4 <= cur_lag;
                            max_lag_sum4 <= sum4;
                            end

                        if ( cur_lag == MAX_LAG ) begin
                            state <= STATE_IDLE;
                            done <= 1;
                            end
                        else begin
                            cur_lag <= cur_lag + 1;
                            corr_start <= 1;
                            end
                        end
                    else
                        corr_start <= 0;

                default:
                    state <= STATE_IDLE;
                endcase
endmodule
```

9.11 Mic FSM

```
module mic_fsm(
    clk, reset_sync, start, time_index,
    data, on_sample, begin_time, end_time, dead);

    input clk;
    input reset_sync;
    input start;
    input [1:0] data;
    input on_sample;

    input [14:0] time_index;
```

```
output [14:0] begin_time, end_time;
reg [14:0] begin_time, end_time;

output dead;
reg dead;


parameter STATE_DONE = 0;
parameter STATE_DEAD = 1;
parameter STATE_ALIVE = 2;

parameter DEAD_THRESH = 100;//100;

reg [14:0] dead_counter;
reg [2:0] state;

always @ ( posedge clk )
    if ( !reset_sync ) begin
        begin_time <= 0;
        end_time <= 0;
        state <= STATE_DONE;
        dead_counter <= 0;
        dead <= 1;
        end
    else
        case (state)

            STATE_DONE:
                if ( start ) begin
                    $display("mic fsm: starting");
                    state <= STATE_DEAD;
                    begin_time <= 0;
                    end_time <= 0;
                    dead_counter <= 0;
                    dead <= 1;
                    end

            STATE_DEAD:
                if ( on_sample )
                    if ( data != 2'd0 ) begin
                        $display("mic fsm: begin_time=", time_index);
                        begin_time <= time_index;
                        dead <= 0;
                        state <= STATE_ALIVE;
                        end
```

```
                STATE_ALIVE:
                    if ( on_sample ) begin
                        if ( data == 0 ) begin
                            dead_counter <= dead_counter + 1;
                            //$display(time_index);
                            end
                        else
                            dead_counter <= 0;

                        if ( dead_counter > DEAD_THRESH ) begin
                        $display("mic fsm: end_time=", time_index);
                            end_time <= time_index;
                            dead <= 1;
                            state <= STATE_DONE;
                            end

                        end

                default:
                    state <= STATE_DONE;

            endcase
endmodule
```

9.12 Pre-Processor

```
module preprocess(
    clk, reset_sync,
    data, // data sampled by the gatherer
    time_index, // time index, essentially address in data memory
    on_sample, // tells when a sample has been gathered
    pulse_addr, pulse_din, pulse_we, // access to pulse_ram
    done, start,
    recv_done, // data gatherer done
    pulsecounter,
    prev_pulsecounter
    );

    input clk;
    input reset_sync;
    input [9:0] data;
    input [14:0] time_index;
    input on_sample;
    input start;
```

```
input recv_done;

output[3:0] pulse_addr;
reg[3:0] pulse_addr;
output [34:0] pulse_din;
reg [34:0] pulse_din;
output pulse_we;
reg pulse_we;

output done;
reg done;

parameter FPULSELENGTH = 2000;//2000; //changed from 2000
parameter MIN_PULSE_LEN = 150;//150;

parameter STATE_IDLE = 0;
parameter STATE_DELAY = 1;
parameter STATE_PULSE_WAIT = 2;
parameter STATE_PULSE_END = 3;
parameter STATE_FIND_LIMITS = 4;
parameter STATE_RECORD1 = 5;
parameter STATE_RECORD2 = 6;

reg[4:0] state;

reg[4:0] valid;

output [3:0] pulsecounter, prev_pulsecounter;
reg [3:0] pulsecounter, prev_pulsecounter;

reg mic_start;
wire dead1, dead2, dead3, dead4, dead5;

wire [14:0] begin_time1, end_time1;
wire [14:0] begin_time2, end_time2;
wire [14:0] begin_time3, end_time3;
wire [14:0] begin_time4, end_time4;
wire [14:0] begin_time5, end_time5;

reg [14:0] best_begin_time, best_end_time;

mic_fsm mic1 (
    .clk(clk),
    .reset_sync(reset_sync),
    .start(mic_start),
```

```
        .time_index(time_index),
        .data(data[1:0]),
        .on_sample(on_sample),
        .begin_time(begin_time1),
        .end_time(end_time1),
        .dead(dead1)
    );

    mic_fsm mic2 (
        .clk(clk),
        .reset_sync(reset_sync),
        .start(mic_start),
        .time_index(time_index),
        .data(data[3:2]),
        .on_sample(on_sample),
        .begin_time(begin_time2),
        .end_time(end_time2),
        .dead(dead2)
    );

    mic_fsm mic3 (
        .clk(clk),
        .reset_sync(reset_sync),
        .start(mic_start),
        .time_index(time_index),
        .data(data[5:4]),
        .on_sample(on_sample),
        .begin_time(begin_time3),
        .end_time(end_time3),
        .dead(dead3)
    );

    mic_fsm mic4 (
        .clk(clk),
        .reset_sync(reset_sync),
        .start(mic_start),
        .time_index(time_index),
        .data(data[7:6]),
        .on_sample(on_sample),
        .begin_time(begin_time4),
        .end_time(end_time4),
        .dead(dead4)
    );

    mic_fsm mic5 (
```

```
        .clk(clk),
        .reset_sync(reset_sync),
        .start(mic_start),
        .time_index(time_index),
        .data(data[9:8]),
        .on_sample(on_sample),
        .begin_time(begin_time5),
        .end_time(end_time5),
        .dead(dead5)
    );

    always @ (posedge clk)
            if (!reset_sync) begin
                done <= 0;
                pulse_we <= 0;
                pulse_addr <= 0;
                mic_start <= 0;
                state <= STATE_IDLE;

                pulsecounter <= 0;
                prev_pulsecounter <= 0;
                end
            else
                case(state)
                    STATE_IDLE: begin
                        done <= 0;
                        if (start) begin
                            $display("PreProc, starting");

                            state <= STATE_DELAY;
                            pulse_addr <= 0;
                            pulse_we <= 0;

                            valid <= 0;

                            pulsecounter <= 0;
                            prev_pulsecounter <= pulsecounter;
                            end
                    end

                    STATE_DELAY:
                        if (time_index > FPULSELENGTH) begin
                            state   <= STATE_PULSE_WAIT;
                            $display("preproc, Start mic: ", time_index);
                            mic_start <= 1;
```

```
            end
        else if ( recv_done ) begin
            done <= 1;
            state <= STATE_IDLE;
            end
        //else
        //  $display("preproc, timeindex not high enough, bitch= ", time_inc

    STATE_PULSE_WAIT:   begin
        mic_start <= 0;

        if ( !dead1 || !dead2 || !dead3 || !dead4 || !dead5 ) // !dead => go
            state <= STATE_PULSE_END;
        else if ( recv_done ) begin
            done <= 1;
            state <= STATE_IDLE;
            end

        end

    STATE_PULSE_END:
        if ( dead1 && dead2 && dead3 && dead4 && dead5 ) begin
            valid[0] <= (end_time1 > begin_time1 + MIN_PULSE_LEN) && (begin_
            valid[1] <= (end_time2 > begin_time2 + MIN_PULSE_LEN) && (begin_
            valid[2] <= (end_time3 > begin_time3 + MIN_PULSE_LEN) && (begin_
            valid[3] <= (end_time4 > begin_time4 + MIN_PULSE_LEN) && (begin_
            valid[4] <= (end_time5 > begin_time5 + MIN_PULSE_LEN) && (begin_

            state <= STATE_FIND_LIMITS;
            end
        else if ( recv_done ) begin
            done <= 1;
            state <= STATE_IDLE;
            end

    STATE_FIND_LIMITS: begin
            best_begin_time = 0;
            best_end_time = 30000;

            if ( valid[0] ) begin
                if ( begin_time1 > best_begin_time )
                    best_begin_time = begin_time1;

                if ( end_time1 < best_end_time )
                    best_end_time = end_time1;
```

```
            end

        if ( valid[1] ) begin
            if ( begin_time2 > best_begin_time )
                best_begin_time = begin_time2;

            if ( end_time2 < best_end_time )
                best_end_time = end_time2;
            end

        if ( valid[2] ) begin
            if ( begin_time3 > best_begin_time )
                best_begin_time = begin_time3;

            if ( end_time3 < best_end_time )
                best_end_time = end_time3;
            end

        if ( valid[3] ) begin
            if ( begin_time4 > best_begin_time )
                best_begin_time = begin_time4;

            if ( end_time4 < best_end_time )
                best_end_time = end_time4;
            end

        if ( valid[4] ) begin
            if ( begin_time5 > best_begin_time )
                best_begin_time = begin_time5;

            if ( end_time5 < best_end_time )
                best_end_time = end_time5;
            end

        $display("preproc, pulse_addr=",pulse_addr, " valid =", valid);

        pulse_we <= 1;
        pulse_din <= {best_begin_time, best_end_time, valid};
        state <= STATE_RECORD1;
    end


STATE_RECORD1: begin
    pulse_we <= 0;
    state <= STATE_RECORD2;
```

```
                        end

                    STATE_RECORD2: begin
                        pulse_addr <= pulse_addr + 1;
                        pulsecounter <= pulsecounter + 1;
                        $display("preproc, recorded data to pulsecounter=",pulsecounter);
                        mic_start <= 1;
                        state <= STATE_PULSE_WAIT;
                        end

                    default:
                        state <= STATE_IDLE;

            endcase
endmodule
```

9.13 Processor

```
module processor(
    clk, reset_sync,
    data_addra, data_addrb, data_douta, data_doutb,
    pulse_dout, pulse_addr,
    max_lag1, max_lag2, max_lag3, max_lag4,
    max_lag_sum1, max_lag_sum2, max_lag_sum3, max_lag_sum4,
    start_addr,
    start, done,
    pulsecounter, angle, newangle,
    td_addr, td_we, td_din,
    angle_chooser, angle_valid_thresh,
    min_dist);

    input clk, reset_sync, start;
    input [3:0] pulsecounter;

    input [7:0] angle_chooser;

    output done;
    reg done;

    output td_we;
    reg td_we;
    output [15:0] td_din;
    reg [15:0] td_din;
    output [3:0] td_addr; // change me!!
    reg [3:0] td_addr;
```

```
input [9:0] data_douta, data_doutb;
output [14:0] data_addra, data_addrb;

input [34:0] pulse_dout;
output [3:0] pulse_addr;
reg [3:0] pulse_addr;

reg start_finder;
wire done_finder;

output [14:0] start_addr;

reg [14:0] start_addr;
reg [14:0] end_addr;
reg [4:0] valid;

output [6:0] max_lag1, max_lag2, max_lag3, max_lag4;
output [25:0] max_lag_sum1, max_lag_sum2, max_lag_sum3, max_lag_sum4;

max_lag_finder max_lag_finderi (
    .clk(clk),
    .reset_sync(reset_sync),
    .start(start_finder),
    .done(done_finder),
    .data_addra(data_addra),
    .data_addrb(data_addrb),
    .data_douta(data_douta),
    .data_doutb(data_doutb),
    .start_addr(start_addr),
    .end_addr(end_addr),
    .max_lag1(max_lag1),
    .max_lag2(max_lag2),
    .max_lag3(max_lag3),
    .max_lag4(max_lag4),
    .max_lag_sum1(max_lag_sum1),
    .max_lag_sum2(max_lag_sum2),
    .max_lag_sum3(max_lag_sum3),
    .max_lag_sum4(max_lag_sum4)
);

parameter STATE_IDLE = 0;
```

```
parameter STATE_FIRST_RUN = 1;
parameter STATE_READ_PULSE = 3;
parameter STATE_WAIT_READ1 = 2;
parameter STATE_CORR_WAIT = 4;
parameter STATE_WAIT_READ2 = 5;
parameter STATE_WAIT_EXTRACT = 6;
parameter STATE_TD_RECORD = 7;

reg start_extract;
wire done_extract;
input [3:0] angle_valid_thresh;
output [7:0] angle;
reg [7:0] angle;
reg [7:0] lastangle;
output [7:0] newangle;

parameter ANGLE_INVALID = 1;

reg[7:0] anglecounter; //postprocessing counter

output [12:0] min_dist;

angleextract angleextracti(
    .clk(clk),
    .reset_sync(reset_sync),
    .pulsecounter(pulsecounter),
    .max_lag({max_lag4, max_lag3, max_lag2, max_lag1}),
    .valid(valid),
    .start_extract(start_extract),
  .done_extract(done_extract),
    .angle(newangle));

reg first_run;
reg [3:0] state;

always @ (posedge clk)
    if ( !reset_sync ) begin
        first_run <= 1;
        state <= STATE_IDLE;

        start_finder <= 0;
        start_extract <= 0;
        done <= 0;
        td_din <= 0;
        lastangle <= 0;
```

```
            angle <= 0;
            anglecounter <= 0;
            end
        else
            case ( state )
                STATE_IDLE:
                    if ( start ) begin
                        if ( first_run == 1 ) begin
                            state <= STATE_FIRST_RUN;
                            done <= 1;
                            first_run <= 0;
                            end
                        else begin
                            state <= STATE_WAIT_READ1;
                            pulse_addr <= 0;

                            td_addr <= 0;
                            td_we <= 0;
                            //td_din <= 0;
                            end
                        end
                    else begin
                        done <= 0;
                        start_finder <= 0;
                        start_extract <= 0;
                        end

                STATE_FIRST_RUN:begin
                    state <= STATE_IDLE;
                end

                STATE_WAIT_READ1:
                    state <= STATE_WAIT_READ2;

                STATE_WAIT_READ2: begin
                    start_addr <= pulse_dout[34:20];
                    end_addr <= pulse_dout[19:5];
                    valid <= pulse_dout[4:0];
                    $display("Proc, this is what valid should be: ", pulse_dout[4:0], " ", v

                    td_we <= 0;
                    state <= STATE_READ_PULSE;
                    end

                STATE_READ_PULSE: begin
```

```
                if ( pulse_addr == pulsecounter )  begin
                    done <= 1;
                    start_addr <= 256;
                    end_addr <= 0;
                    start_finder <= 1; // to reset lags
                    state <= STATE_FIRST_RUN;   // Give extra cycle delay
                    end
                else if (   (angle_chooser == 0) &&
                                (valid != 5'b11111) &&
                                (start_addr > 8000) &&
                                (end_addr < start_addr) ) begin
                    $display("Proc, trying the next pulse after, ", pulse_addr, " start_
                    pulse_addr <= pulse_addr + 1;

                    state <= STATE_WAIT_READ2;
                    end
                else begin // all valid
                    $display("Proc, All valid!, start_addr=", start_addr, " cur value o
                    state <= STATE_CORR_WAIT;
                    start_finder <= 1;
                    end
            end

            STATE_CORR_WAIT: begin
                if ( done_finder ) begin
                    start_extract <= 1;
                    state <= STATE_WAIT_EXTRACT;
                    end
                else
                    start_finder <= 0;
                end

        STATE_WAIT_EXTRACT:
            if ( done_extract ) begin
                //td_addr <= td_addr + 1;

                if ( angle_chooser == 0 ) begin
                    if ((newangle != 0) && (start_addr[12:5] > 10) ) begin
                        if ( (newangle == lastangle) && (anglecounter < 30))
                            anglecounter <= anglecounter + 1;
                        else
                            anglecounter <= 0;

                        lastangle <= newangle;
```

```
                    if (anglecounter > angle_valid_thresh)
                        angle <= newangle;
                end

                case(angle)
                    45: td_din <= {8'd32, start_addr[12:5]};
                    55: td_din <= {8'd39, start_addr[12:5]};
                    65: td_din <= {8'd46, start_addr[12:5]};
                    75: td_din <= {8'd53, start_addr[12:5]};
                    85: td_din <= {8'd60, start_addr[12:5]};
                    90: td_din <= {8'd64, start_addr[12:5]};
                    95: td_din <= {8'd67, start_addr[12:5]};
                    110: td_din <= {8'd78, start_addr[12:5]};
                    118: td_din <= {8'd84, start_addr[12:5]};
                    125: td_din <= {8'd89, start_addr[12:5]};
                    135: td_din <= {8'd96, start_addr[12:5]};
                    default: td_din <= td_din; // keep old value
                endcase

                end
            else
                td_din <= {angle_chooser, 8'd70};

            state <= STATE_TD_RECORD;

            /*if ( angle == 0 ) // INVALID
                state <= STATE_WAIT_READ2;
            else begin
                td_din <= {angle, start_addr[14:7]};
                state <= STATE_TD_RECORD;
                end */

            end
        else
             start_extract <= 0;

    STATE_TD_RECORD: begin
        td_we <= 1;
        done <= 1;
        state <= STATE_IDLE;
        //state <= STATE_WAIT_READ2;
        end

    endcase
```

```
endmodule
```

9.14 Front Conversion

```
module t_d_conv_front(clk, reset, t_d_pair, max_td_addr, start_fwrite,
                      td_addr, pd_addr, pd_we, pix_data, done);

     input clk;
     input reset;
     input [15:0] t_d_pair;    // from theta/distance pairs RAM
                          // assume t_d_pair[15:8] is the angle
                               //        t_d_pair[7:0] is the distance at that angle
     input [3:0] max_td_addr; // maximum theta-distance pair address
     input start_fwrite; // external enable signal to start writing to memory
     output [7:0] td_addr; // to theta/distance pairs RAM
     output [16:0] pd_addr;   // to RAM that the display module will read from (320x240)
     output pd_we;  // to RAM that the display module will read from
     output [5:0] pix_data; // to RAM that the display module will read from
     output done;

     reg [7:0] td_addr;
     reg [7:0] theta_req;
     reg [16:0] pd_addr; // 320 x 240 = 76800 locations (17 bits)
     reg pd_we;
     reg [5:0] pix_data;
     reg done;
     reg [15:0] dcostheta;
     reg [8:0] xcoord;
     reg [7:0] ycntr;
     reg [7:0] y_init;
     reg [7:0] y_max;
     reg [3:0] ext_width;
     reg [3:0] width_cnt;
     reg [3:0] front_state;
     reg [3:0] counter;
     reg [7:0] neg_cos;

     parameter frconv_START    = 4'd0;
     parameter blank_mem       = 4'd1;
     parameter blank_mem_write = 4'd2;
     parameter read_tdpair     = 4'd3;
     parameter trig_lookup     = 4'd4;
     parameter find_xcoord     = 4'd5;
     parameter prep_pixdata    = 4'd6;
```

```
parameter prep_width      = 4'd7;
parameter init_cntrs      = 4'd8;
parameter write_pixdata   = 4'd9;
//parameter write_pixdata1  = 4'd10;
//parameter write_pixdata2  = 4'd11;
//parameter write_pixdata3  = 4'd12;

wire [7:0] cos_theta;
cosine_lookup cosine_table(.THETA(theta_req), .COSINE(cos_theta));

///////// for testing purposes only ////////////////////////////////////////
//wire [15:0] t_d_pair;
//tdpair_test_table tdpair_lookup(.clk(clk), .addr(td_addr), .dout(t_d_pair));
////////////////////////////////////////////////////////////////////////////

always @ (posedge clk) begin
  if (reset)
     begin
      td_addr <= 8'b0;
       pd_addr <= 17'b0;
      front_state <= frconv_START;
     end
   else
     begin
     case (front_state)
        frconv_START:    begin // this is also an idle state
                          pd_we <= 0;
                                 done <= 0;
                                 td_addr <= 8'b0;
                                theta_req <= 8'b0;
                                //blank_cntr <= 17'b0;
                           pd_addr <= 17'b0;
                                pix_data <= 6'b0;
                                ycntr <= 8'd0;
                                 counter <= 0;
                                if (start_fwrite) front_state <= blank_mem;
                                else front_state <= frconv_START;
                         end

        blank_mem:       begin
                           pd_we <= 0;
                          if (pd_addr < /*17'd10 - 1*/ 17'd76800) // address 0 never g
                                 begin
                                    pix_data <= 6'b0;
                                    pd_addr <= pd_addr + 1;
```

```
                                front_state <= blank_mem_write;
                              end
                            else
                              begin
                                pd_addr <= 17'b0;
                                front_state <= read_tdpair;
                              end
                        end

blank_mem_write: begin
                   pd_we <= 1;
                        front_state <= blank_mem;
                        end

 read_tdpair:      begin
                  if (td_addr > max_td_addr) // if there are no more
                          begin                              //  theta-distance
                            td_addr <= 8'b0;            //  go back to being ic
                            front_state <= frconv_START;
                             done <= 1;
                           end
                   else front_state <= trig_lookup;
                  end

trig_lookup:      begin
                   theta_req <= t_d_pair[15:8]; // to cosine lookup table
                   front_state <= find_xcoord;
                  end

find_xcoord:      begin
                   if (cos_theta[7]) // if cosine is negative
                          begin
                             neg_cos = ~cos_theta + 1'b1;
                            dcostheta = t_d_pair[7:0] * neg_cos;
                            //xcoord = 9'd4 - dcostheta[13:7];
                            xcoord = 9'd159 - dcostheta[13:7];
                          end
                        else // cosine is positive
                          begin
                           dcostheta = t_d_pair[7:0] * cos_theta;
                           //xcoord = 9'd4 + dcostheta[13:7];
                            xcoord = 9'd159 + dcostheta[13:7];
                          end
                        front_state <= prep_pixdata;
                         counter <= 0;
```

```
                        end

prep_pixdata:    begin // determine what color to make the bars based on distan
                  if ((t_d_pair[7:0] >= 0) &&
                            (t_d_pair[7:0] < 16)) pix_data <= 6'd33;
                        else if ((t_d_pair[7:0] >= 16) &&
                                  (t_d_pair[7:0] < 32)) pix_data <= 6'd35;
                       else if ((t_d_pair[7:0] >= 32) &&
                                  (t_d_pair[7:0] < 48)) pix_data <= 6'd37;
                       else if ((t_d_pair[7:0] >= 48) &&
                                  (t_d_pair[7:0] < 64)) pix_data <= 6'd39;
                        else if ((t_d_pair[7:0] >= 64) &&
                                  (t_d_pair[7:0] < 80)) pix_data <= 6'd41;
                        else if ((t_d_pair[7:0] >= 80) &&
                                  (t_d_pair[7:0] < 96)) pix_data <= 6'd43;
                        else if ((t_d_pair[7:0] >= 96) &&
                                  (t_d_pair[7:0] < 112)) pix_data <= 6'd45;
                        else if ((t_d_pair[7:0] >= 112) &&
                                  (t_d_pair[7:0] < 128)) pix_data <= 6'd47;
                        else if ((t_d_pair[7:0] >= 128) &&
                                  (t_d_pair[7:0] < 144)) pix_data <= 6'd49;
                        else if ((t_d_pair[7:0] >= 144) &&
                                  (t_d_pair[7:0] < 160)) pix_data <= 6'd51;
                        else if ((t_d_pair[7:0] >= 160) &&
                                  (t_d_pair[7:0] < 176)) pix_data <= 6'd53;
                        else if ((t_d_pair[7:0] >= 176) &&
                                  (t_d_pair[7:0] < 192)) pix_data <= 6'd55;
                        else if ((t_d_pair[7:0] >= 192) &&
                                  (t_d_pair[7:0] < 208)) pix_data <= 6'd57;
                        else if ((t_d_pair[7:0] >= 208) &&
                                  (t_d_pair[7:0] < 224)) pix_data <= 6'd59;
                        else if ((t_d_pair[7:0] >= 224) &&
                                  (t_d_pair[7:0] < 240)) pix_data <= 6'd61;
                        else pix_data <= 6'd63;
                         y_max <= 8'd119 + (8'd119 - t_d_pair[7:1]);
                         y_init <= 8'd119 - (8'd119 - t_d_pair[7:1]);
                        front_state <= prep_width;
                       end

prep_width:      begin // determine what width to make the bars based on distan
                   if ((t_d_pair[7:0] >= 0) &&
                             (t_d_pair[7:0] < 32)) ext_width <= 8;
                         else if ((t_d_pair[7:0] >= 32) &&
                                   (t_d_pair[7:0] < 64)) ext_width <= 7;
                         else if ((t_d_pair[7:0] >= 64) &&
```

```
                                        (t_d_pair[7:0] < 96)) ext_width <= 6;
                                else if ((t_d_pair[7:0] >= 96) &&
                                        (t_d_pair[7:0] < 128)) ext_width <= 5;
                                else if ((t_d_pair[7:0] >= 128) &&
                                        (t_d_pair[7:0] < 160)) ext_width <= 4;
                                else if ((t_d_pair[7:0] >= 160) &&
                                        (t_d_pair[7:0] < 192)) ext_width <= 3;
                                else if ((t_d_pair[7:0] >= 192) &&
                                        (t_d_pair[7:0] < 224)) ext_width <= 2;
                                else ext_width <= 1;
                             front_state <= init_cntrs;
                            end

init_cntrs:          begin
                             ycntr <= y_init;
                                width_cnt <= 0;
                                front_state <= write_pixdata;
                               end

write_pixdata:       begin
                             if (width_cnt <= ext_width)
                                     begin
                                if (ycntr < y_max /*8'd5 8'd240*/)
                                       begin
                                         pd_addr <= ycntr * 9'd320 + xcoord + width_cnt;
                                             // all pixels in a column should be the same
                                          counter <= counter + 1;
                                          if (counter == 2)
                                            begin
                                              pd_we <= 1;
                                              front_state <= write_pixdata;
                                             end
                                          else if (counter == 3)
                                            begin
                                            pd_we <= 0;
                                              ycntr <= ycntr + 1;
                                            front_state <= write_pixdata;
                                            counter <= 0;
                                          end
                                          else front_state <= write_pixdata;
                                       end
                                      else
                                        begin
                                          ycntr <= y_init; //8'b0;
                                            width_cnt <= width_cnt + 1;
```

```
                                        front_state <= write_pixdata;
                                       end
                                     end
                                   else
                                     begin
                                       td_addr <= td_addr + 1;
                                       front_state <= read_tdpair;
                                     end
                               end

            default: front_state <= frconv_START;
          endcase
         end
     end

endmodule
```

9.15 Top Conversion

```
module t_d_conv_top(clk, reset, t_d_pair, max_td_addr, start_twrite,
                    td_addr, pd_addr, pd_we, pix_data, done);

    input clk;
    input reset;
    input [15:0] t_d_pair; // from theta/distance pairs RAM
                           // assume t_d_pair[15:8] is the angle
                                  //       t_d_pair[7:0] is the distance at that angle
    input [3:0] max_td_addr; // maximum theta-distance pair address
     input start_twrite; // external enable signal to start writing to memory
    output [7:0] td_addr; // to theta/distance pairs RAM
    output [16:0] pd_addr; // to RAM that the display module will read from
    output pd_we;             // to RAM that the display module will read from
    output [5:0] pix_data; // to RAM that the display module will read from
     output done;

     reg [7:0] td_addr;
     reg [7:0] theta_req;
     reg [16:0] pd_addr;
     reg [5:0] pix_data;
     reg [15:0] dsintheta;
     reg [15:0] dcostheta;
     reg [8:0] xcoord;
     reg [7:0] ycoord;
     reg pd_we;
     reg done;
```

```
reg [3:0] top_state;
reg [7:0] neg_cos;

reg [3:0] counter; //used for write timing

parameter topconv_START  = 4'd0;
parameter blank_mem      = 4'd1;
parameter blank_mem_write= 4'd2;
parameter read_tdpair    = 4'd3;
parameter trig_lookup    = 4'd4;
parameter prep_pixdata   = 4'd5;
parameter write_pixdata1 = 4'd6;
parameter write_pixdata2 = 4'd7;
parameter write_pixdata3 = 4'd8;
parameter write_pixdata4 = 4'd9;
parameter write_pixdata5 = 4'd10;
parameter write_pixdata6 = 4'd11;
parameter write_pixdata7 = 4'd12;
parameter write_pixdata8 = 4'd13;
parameter write_pixdata9 = 4'd14;

wire [7:0] sin_theta;
wire [7:0] cos_theta;

sine_lookup sin_table(.THETA(theta_req), .SINE(sin_theta));
cosine_lookup cos_table(.THETA(theta_req), .COSINE(cos_theta));

//////// for testing purposes only ////////////////////////////////////
/* wire [15:0] t_d_pair;
tdpair_test_table tdpair_lookup(.clk(clk), .addr(td_addr), .dout(t_d_pair));*/
////////////////////////////////////////////////////////////////////////


always @ (posedge clk) begin

if (reset)
     begin
      td_addr <= 8'b0;
       pd_addr <= 17'b0;
      top_state <= topconv_START;
    end
else
  case (top_state)
     topconv_START:   begin // this is also an idle state
                            pd_we <= 0;
```

```
                        done <= 0;
                    td_addr <= 8'b0;
                        theta_req <= 8'b0;
                     pd_addr <= 17'b0;
                        pix_data <= 6'b0;
                        counter <= 0;
                        if (start_twrite) top_state <= blank_mem;
                        else top_state <= topconv_START;
                    end

blank_mem:          begin
                        pd_we <= 0;
                    if (pd_addr < 17'd76800)
                          begin
                            pix_data <= 6'b000000;
                            pd_addr <= pd_addr + 1;
                            top_state <= blank_mem_write;
                           end
                        else
                          begin
                            pd_addr <= 17'b0;
                            top_state <= read_tdpair;
                           end
                    end


blank_mem_write: begin
                        pd_we <= 1;
                        top_state <= blank_mem;
                       end

read_tdpair:       begin
                    if (td_addr > max_td_addr) // if there are no more
                          begin                          //  theta-distance pairs,
                            td_addr <= 0;           //  go back to being idle
                            top_state <= topconv_START;
                            done <= 1; //we're done
                           end
                     else top_state <= trig_lookup;
                    end

trig_lookup:  begin
                    theta_req <= t_d_pair[15:8];
                    top_state <= prep_pixdata;
                    end
```

```
prep_pixdata:    begin
                        dsintheta = t_d_pair[7:0] * sin_theta;
                        ycoord = 230 - dsintheta[13:7]; // ycoord only goes up to

                        if ( cos_theta[7] ) // if cosine is negative
                          begin
                            neg_cos = ~cos_theta + 1'b1;
                            dcostheta = t_d_pair[7:0] * neg_cos;
                           xcoord = 9'd159 - {2'd0, dcostheta[13:7]};
                          end
                        else // cosine is positive
                          begin
                           dcostheta = t_d_pair[7:0] * cos_theta;
                            xcoord = 9'd159 + dcostheta[13:7];
                          end

                        top_state <= write_pixdata1;
                        counter <= 0;
                      end

write_pixdata1:  begin // this is the middle of the square

                     $display("theta:,", theta_req);
                     $display("som theta:,", sin_theta);
                    $display("xcoord:,", xcoord);

                        pix_data <= 6'b111100;
                        pd_addr <= ycoord * 9'd320 + xcoord;
                        counter <= counter + 1;
                        if (counter == 2) pd_we <= 1;
                        else if (counter == 3)
                          begin
                            pd_we <= 0;
                            top_state <= write_pixdata2;
                            counter <= 0;
                          end
                   end

write_pixdata2:  begin // this the top left of the square
                        pix_data <= 6'b111100;
                        pd_addr <= (ycoord - 1'b1) * 9'd320 + (xcoord - 1'b1);
                        counter <= counter + 1;
                        if (counter == 2) pd_we <= 1;
                        else if (counter == 3)
```

```
                                begin
                                  pd_we <= 0;
                                  top_state <= write_pixdata3;
                                  counter <= 0;
                                end
                        end

        write_pixdata3:  begin // this the middle of the top row of the square
                                pix_data <= 6'b111100;
                                pd_addr <= (ycoord - 1'b1) * 9'd320 + xcoord;
                                counter <= counter + 1;
                                if (counter == 2) pd_we <= 1;
                                else if (counter == 3)
                                  begin
                                    pd_we <= 0;
                                    top_state <= write_pixdata4;
                                    counter <= 0;
                                  end
                        end

        write_pixdata4:  begin // this the top right of the square
                                pix_data <= 6'b111100;
                                pd_addr <= (ycoord - 1'b1) * 9'd320 + (xcoord + 1'b1);
                                counter <= counter + 1;
                                if (counter == 2) pd_we <= 1;
                                else if (counter == 3)
                                  begin
                                    pd_we <= 0;
                                    top_state <= write_pixdata5;
                                    counter <= 0;
                                  end
                        end

      write_pixdata5:  begin // this the middle left of the square
                                pix_data <= 6'b111100;
                                pd_addr <= ycoord * 9'd320 + (xcoord - 1'b1);
                                counter <= counter + 1;
                                if (counter == 2) pd_we <= 1;
                                else if (counter == 3)
                                  begin
                                    pd_we <= 0;
                                    top_state <= write_pixdata6;
                                    counter <= 0;
                                  end
                        end
```

```
write_pixdata6:  begin // this the middle right of the square
                        pix_data <= 6'b111100;
                        pd_addr <= ycoord * 9'd320 + (xcoord + 1'b1);
                        counter <= counter + 1;
                        if (counter == 2) pd_we <= 1;
                        else if (counter == 3)
                          begin
                            pd_we <= 0;
                            top_state <= write_pixdata7;
                            counter <= 0;
                          end
                 end

write_pixdata7:  begin // this the bottom left of the square
                        pix_data <= 6'b111100;
                        pd_addr <= (ycoord + 1'b1) * 9'd320 + (xcoord - 1'b1);
                        counter <= counter + 1;
                        if (counter == 2) pd_we <= 1;
                        else if (counter == 3)
                          begin
                            pd_we <= 0;
                            top_state <= write_pixdata8;
                            counter <= 0;
                          end
                 end

write_pixdata8:  begin // this the bottom middle of the square
                        pix_data <= 6'b111100;
                        pd_addr <= (ycoord + 1'b1) * 9'd320 + xcoord;
                        counter <= counter + 1;
                        if (counter == 2) pd_we <= 1;
                        else if (counter == 3)
                          begin
                            pd_we <= 0;
                            top_state <= write_pixdata9;
                            counter <= 0;
                          end
                 end

write_pixdata9:  begin // this the bottom right of the square
                        pix_data <= 6'b111100;
                        pd_addr <= (ycoord + 1'b1) * 9'd320 + (xcoord + 1'b1);
                        counter <= counter + 1;
                        if (counter == 2) pd_we <= 1;
```

```
                                    else if (counter == 3)
                                      begin
                                        pd_we <= 0;
                                        top_state <= read_tdpair;
                                        td_addr <= td_addr + 1; //increment for the next loop
                                        counter <= 0;
                                      end
                                end

          default: top_state <= topconv_START;
        endcase
      end

endmodule
```

9.16 Transmitter

```
module transmit(clk, reset_sync, enable, transmit_bit);

    input clk, reset_sync;
    input enable;

    reg [11:0] counter;
   reg [3:0] state;

    output transmit_bit;
    reg transmit_bit;

    // Square wave

    always @ ( posedge clk )
        if ( !reset_sync ) begin
            state <= 0;
            counter <= 0;
            transmit_bit <= 0;
            end
        else
            case ( state )

                0:
                    if ( enable ) begin
                        state <= 1;
                        end
                    else
                        state <= 0;
```

```
                1:
                    if ( (counter == 390) && enable ) begin   //was 389
                        counter <= 0;
                        transmit_bit <= ~transmit_bit;
                        end
                    else if ( !enable ) begin
                        state <= 0;
                        counter <= 0;
                        transmit_bit <= 0;
                        end
                    else
                        counter <= counter + 1;

                default:
                    state <= 0;

            endcase

endmodule
```

9.17 VGA Controller

```
module VGA_Mod(reset, clk, pixel_count, line_count, hsync, vsync, vblank, hblank);
   input reset;
   input clk;
   output [9:0] pixel_count;
   output [9:0] line_count;
   output hsync;
   output vsync;
        output vblank;
        output hblank;

        parameter WIDTH = 640;
        parameter HEIGHT = 480;
        parameter H_FRONT_PORCH = 16;
        parameter H_SYNC_LENGTH = 96;
        parameter H_BACK_PORCH = 48;
        parameter V_FRONT_PORCH = 11;
        parameter V_SYNC_LENGTH = 2;
        parameter V_BACK_PORCH = 32;

        reg [9:0] pixel_count = 1;
        reg [9:0] line_count = 1;
        reg hsync = 1;
        reg vsync = 1;
        reg vblank = 1;
```

```
        reg hblank = 1;

        always @(posedge clk)
        begin
               if (!reset) begin
                       vsync = 1;
                       hsync = 1;
                       pixel_count <= 1;
                       line_count <= 1;
               end
               else
               begin
                       //hsync
                       if (pixel_count == WIDTH + 1 + H_FRONT_PORCH) hsync = 0;
                       if (pixel_count == WIDTH + 1 + H_FRONT_PORCH + H_SYNC_LENGTH) hsync =

                       //hblank
                       if (pixel_count == WIDTH + 1) hblank = 0;
                       if (pixel_count == WIDTH + 1 + H_FRONT_PORCH + H_SYNC_LENGTH + H_BACK

                       //vsync
                       if (line_count == HEIGHT + 1 + V_FRONT_PORCH) vsync = 0;
                       if (line_count == HEIGHT + 1 + V_FRONT_PORCH + V_SYNC_LENGTH) vsync =

                       //vblank
                       if (line_count == HEIGHT + 1) vblank = 0;
                       if (line_count == HEIGHT + 1 + V_FRONT_PORCH + V_SYNC_LENGTH + V_BACK


                       //update pixel_count and line_count
                       if (pixel_count == WIDTH + 1 + H_FRONT_PORCH + H_SYNC_LENGTH + H_BACK
                       begin
                               if (line_count == HEIGHT + 1 + V_FRONT_PORCH + V_SYNC_LENGTH
                                       line_count <= 1;
                               else line_count <= line_count + 1;
                       end

                       if (pixel_count == WIDTH + 1 + H_FRONT_PORCH + H_SYNC_LENGTH + H_BACK
                               else pixel_count <= pixel_count + 1;
               end
        end //always

endmodule
```

9.18 RS-232 Transmitter

```
////////////////////////////////////////////////////////////////////////////
// TOP-LEVEL  RS232 TRANSMITTER
//
// 6.111: J.Lee
// Last Modified: 4/18/2006
//  Source: partially adapted from
//                  fpga4fun.com KNJN LLC - 2003, 2004, 2005
//
//
//  Current Configuration: (based on 27Mhz clock)
//          Baud Rate:    115200 (reconfigure @ BaudGen module)
//          Parity:       NONE
//          Stop-bit:     1
//          Flow Control: NONE
////////////////////////////////////////////////////////////////////////////

module rs232(reset, clock_27mhz, rxd, cts, txd, rts, rx_data, tx_data, tx_start);
  input reset, clock_27mhz;
  input rxd, cts;                   //ports from Labkit
  output txd, rts;              //ports from Labkit
  output [7:0] rx_data;         //8-bit data received
  input [7:0] tx_data;          //8-bit data to send
  input tx_start;                   //==HIGH to initiate transmission

  //Flow control: NONE
  //cts (not used);
  assign rts = 1'b1;

  //NO Receiver
  assign rx_data = 8'hFF;


  wire tx_busy;
  async_transmitter tx(.clock_27mhz(clock_27mhz), .txd(txd),
        .tx_start(tx_start & ~tx_busy), .tx_data(tx_data), .tx_busy(tx_busy));

endmodule
////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////
// TRANSMITTER MODULE
//
```

```
// Adapted from
//      RS-232 TX module
//      (c) fpga4fun.com KNJN LLC - 2003, 2004, 2005

module async_transmitter(clock_27mhz, tx_start, tx_data, txd, tx_busy);
  input clock_27mhz, tx_start;
  input [7:0] tx_data;
  output txd, tx_busy;

  wire tx_busy;

  ////////////////////////////////////////////////////////////////
  //connect baud generator
  wire serial_clk;
  baud_gen tx_clk(.clock_27mhz(clock_27mhz), .serial_clk(serial_clk),
              .enable(tx_busy), .rx_mode(1'b0) );


  ////////////////////////////////////////////////////////////////
  reg txd;
  reg [3:0] state;
  reg [2:0] index;

  parameter idle        = 0;
  parameter tx_ready = 1;
  parameter start_bit= 2;
  parameter stop_bit = 3;
  parameter sending_packet = 4;

  always @(posedge clock_27mhz) begin
    txd <= (state==start_bit) ? 1'b0 :             //start-bit==0
           (state==sending_packet) ? tx_data[index] : //data-bit at i^th bit (i=0..7)
           1'b1;                   //else, pulled HIGH
   if (serial_clk)
        index <= (state==start_bit)         ? 3'b000 :
                 (state==sending_packet) ? index+1:
                                                 index;
    case(state)
        idle:            if(tx_start)                       state <= tx_ready;
        tx_ready:        if(serial_clk)                     state <= start_bit;
        start_bit:       if(serial_clk)                     state <= sending_packet;  // st
        sending_packet: if(serial_clk && index==3'd7) state <= stop_bit;        // bit[0..7]
        stop_bit:        if(serial_clk)                     state <= idle;
        default:         if(serial_clk)                     state <= idle;
    endcase
```

```
  end

  assign tx_busy = (state!=idle);
endmodule



//////////////////////////////////////////////////////////////
// BAUD GENERATOR
// Source from
//      RS-232 RX module
//      (c) fpga4fun.com KNJN LLC - 2003, 2004, 2005

module baud_gen(clock_27mhz, serial_clk, enable, rx_mode);
  input clock_27mhz, enable, rx_mode;
  output serial_clk;

  parameter clkfreq = 27000000;     // 27 MHz
  parameter baudrate = 115200;  //
  parameter baudrate_8 = 115200*8;  // 8 times over sampling (when receiving)

  // Baud generator
  parameter acc_width = 16;
  reg [acc_width:0] accumulator;
  parameter inc = ((baudrate<<(acc_width-4))+(clkfreq>>5))/(clkfreq>>4);
  parameter inc8 = ((baudrate_8<<(acc_width-7))+(clkfreq>>8))/(clkfreq>>7);

  assign serial_clk = accumulator[acc_width];

  always @(posedge clock_27mhz) begin
    if(enable)
        accumulator <= accumulator[acc_width-1:0] + inc;
    else if (rx_mode)
        accumulator <= accumulator[acc_width-1:0] + inc8;
  end
endmodule
```