

```

////////////////////////////////////
// Lab Kit - Final Project
// Vivek Shah
//
// 5/16/2006
//
////////////////////////////////////
wire      reset;
wire [2:0] resolution_select;
wire [4:0] max_lines, max_blocks, line_active;
wire [71:0] row_in;
wire [63:0] row;
wire      encode_busy;
wire      transmit_busy;
wire [2:0] inter_row_cnt;
wire [4:0] block_read, line_read;
wire [194:0] data_output;
wire [103:0] mult_column;
wire [9:0] write_address, read_addr;
wire      wen;
wire [2:0] output_select;
wire [143:0] shift_reg_output;
wire [17:0] stage1_output;

// Noel's wires
wire [77:0] dout;
wire [3:0] rs232_state;
    wire [1:0] shift_state, tx_control_state;
wire      ready, send_block, done, send, enable;

reg [12:0] coef, analyzer_out, mult_out;
reg [7:0] row_out;
reg [7:0] data_in_out;
reg [17:0] sr_out;
wire [3:0] num0, num1, num2, num3, num4, num5, num6, num7;
wire [79:0] dots0, dots1, dots2, dots3, dots4, dots5, dots6, dots7;
wire [639:0] dots;

// Ray's wires
wire [7:0] write_enables;
wire [9:0] gray_addr;
wire [23:0] vga_out;
wire [63:0] data_in;

assign vga_out_red = vga_out[23:16];
assign vga_out_green = vga_out[15:8];
assign vga_out_blue = vga_out[7:0];
assign vga_out_pixel_clock = ~clock_27mhz;

// Analyzer inputs

assign analyzer1_clock = clock_27mhz;
assign analyzer2_clock = clock_27mhz;
assign analyzer3_clock = clock_27mhz;
assign analyzer4_clock = clock_27mhz;

```

```

assign analyzer1_data[12:0] = mult_out;
assign analyzer1_data[15:13] = output_select;

assign analyzer2_data = {rs232_rts, rs232_cts, rs232_txd, rs232_rxd,
rs232_state,
                                                                    send_block, done,
send, ready, enable, 3'd0};
/* -----\/\----- EXCLUDED -----\/\-----
assign analyzer2_data[0] = reset;
assign analyzer2_data[1] = encode_busy;
assign analyzer2_data[2] = wen;
assign analyzer2_data[15:3] = analyzer_out;
-----/\\----- EXCLUDED -----/\\----- */

assign analyzer3_data[7:0] = row_out;
assign analyzer3_data[15:8] = data_in_out;

assign analyzer4_data[7:0] = write_enables;
assign analyzer4_data[15:8] = {3'b0, line_active[4:0]};

///

assign max_lines = 29;
assign max_blocks = 29;
//assign line_active = {3'b0, switch[5:4]};
assign resolution_select = {2'b0, switch[6]};
assign led[0] = wen;
assign led[1] = encode_busy;
assign num0 = coef[3:0];
assign num1 = coef[7:4];
assign num2 = coef[11:8];
assign num3 = {3'b000, coef[12]};
assign num4 = line_read[3:0];
assign num5 = block_read[3:0];
assign num6 = write_address[3:0];
assign num7 = inter_row_cnt;
assign dots = {dots7, dots6, dots5, dots4, dots3, dots2, dots1, dots0};
//assign row_in = {1'b0, row[63:56], 1'b0, row[55:48], 1'b0, row[47:40],
1'b0, row[39:32],
// 1'b0, row[31:24], 1'b0, row[23:16], 1'b0, row[15: 8], 1'b0, row[
7: 0]};
assign row_in = (switch[6]) ? {9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255, 9'd255, 9'd255} :
{1'b0, row[63:56], 1'b0, row[55:48], 1'b0, row[47:40], 1'b0,
row[39:32], 1'b0, row[31:24], 1'b0, row[23:16], 1'b0, row[15: 8], 1'b0, row[ 7:
0]};

always @(switch[2:0])
case (switch[2:0])
7: coef[12:0] = data_output[12:0];
6: coef[12:0] = data_output[25:13];
5: coef[12:0] = data_output[38:26];
4: coef[12:0] = data_output[51:39];
3: coef[12:0] = data_output[64:52];
2: coef[12:0] = data_output[77:65];
1: coef[12:0] = data_output[90:78];

```

```

    0: coef[12:0] = data_output[103:91];
endcase // case(switch[2:0])

always @(switch[2:0])
case (switch[2:0])
  0: analyzer_out[12:0] = data_output[12:0];
  1: analyzer_out[12:0] = data_output[25:13];
  2: analyzer_out[12:0] = data_output[38:26];
  3: analyzer_out[12:0] = data_output[51:39];
  4: analyzer_out[12:0] = data_output[64:52];
  5: analyzer_out[12:0] = data_output[77:65];
  6: analyzer_out[12:0] = data_output[90:78];
  7: analyzer_out[12:0] = data_output[103:91];
endcase // case(switch[2:0])

always @(switch[2:0])
case (switch[2:0])
  0: mult_out[12:0] = mult_column[12:0];
  1: mult_out[12:0] = mult_column[25:13];
  2: mult_out[12:0] = mult_column[38:26];
  3: mult_out[12:0] = mult_column[51:39];
  4: mult_out[12:0] = mult_column[64:52];
  5: mult_out[12:0] = mult_column[77:65];
  6: mult_out[12:0] = mult_column[90:78];
  7: mult_out[12:0] = mult_column[103:91];
endcase // case(switch[2:0])

always @(switch[2:0])
case (switch[2:0])
  0: sr_out[17:0] = shift_reg_output[17:0];
  1: sr_out[17:0] = shift_reg_output[35:18];
  2: sr_out[17:0] = shift_reg_output[53:36];
  3: sr_out[17:0] = shift_reg_output[71:54];
  4: sr_out[17:0] = shift_reg_output[89:72];
  5: sr_out[17:0] = shift_reg_output[107:90];
  6: sr_out[17:0] = shift_reg_output[125:108];
  7: sr_out[17:0] = shift_reg_output[143:126];
endcase // case(switch[2:0])

always @(switch[2:0])
case (switch[2:0])
  0: row_out[7:0] = row[7:0];
  1: row_out[7:0] = row[15:8];
  2: row_out[7:0] = row[23:16];
  3: row_out[7:0] = row[31:24];
  4: row_out[7:0] = row[39:32];
  5: row_out[7:0] = row[47:40];
  6: row_out[7:0] = row[55:48];
  7: row_out[7:0] = row[63:56];
endcase // case(switch[1:0])

always @(switch[2:0])
case (switch[2:0])
  0: data_in_out[7:0] = data_in[7:0];
  1: data_in_out[7:0] = data_in[15:8];
  2: data_in_out[7:0] = data_in[23:16];
  3: data_in_out[7:0] = data_in[31:24];

```

```

        4: data_in_out[7:0] = data_in[39:32];
        5: data_in_out[7:0] = data_in[47:40];
        6: data_in_out[7:0] = data_in[55:48];
        7: data_in_out[7:0] = data_in[63:56];
    endcase // case(switch[1:0])

/* -----\ /----- EXCLUDED -----\ /-----
    always @(switch[2:0])
        case (switch[2:0])
            0: row = {9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255};
            1: row = {9'd127, 9'd127, 9'd127, 9'd127, 9'd127, 9'd127, 9'd127,
9'd127};
            2: row = {9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63};
            3: row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};
            4: row = {9'd1, 9'd100, 9'd1, 9'd100, 9'd1, 9'd100, 9'd1, 9'd100};
            5: row = {9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255};
            6: row = {9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255};
            7: row = {9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255};
        endcase // case(switch[2:0])
    -----\ /----- EXCLUDED -----\ /----- */

    debounce DB0(.clock(clock_27mhz), .reset(1'b0), .noisy(button0),
.clean(reset));

    video_capture VIDCAP01(
        .clock_27mhz(clock_27mhz),
        .tv_in_line_clock1(tv_in_line_clock1),
        .reset_button(button1),
        .encoder_busy(encode_busy),
        .tv_in_ycrCb(tv_in_ycrCb),
        .vga_out_sync_b(vga_out_sync_b),
        .vga_out_blank_b(vga_out_blank_b),
        .vga_out_hsync(vga_out_hsync),
        .vga_out_vsync(vga_out_vsync),
        .vga_out(vga_out),
        .macro_line(line_active),
        .write_enables(write_enables),
        .data_in(data_in),
        .gray_addra(gray_addra),
        .tv_in_reset_b(tv_in_reset_b),
        .tv_in_i2c_clock(tv_in_i2c_clock),
        .tv_in_i2c_data(tv_in_i2c_data)
    );

    video_memory VIDMEM001(
        .wr_clk(tv_in_line_clock1),
        .addra(gray_addra),
        .dina(data_in),
        .write_enables(write_enables),
        .r_clk(clock_27mhz),
        .line_read(line_read),
        .block_read(block_read),

```

```

        .inter_row_cnt(inter_row_cnt),
        .addrb(),
        .output_row(row)
    );
encoder ENCODER01(
    .clk(clock_27mhz),
    .reset(reset),
    .resolution_select(resolution_select),
    .max_lines(max_lines),
    .max_blocks(max_blocks),
    .line_active(line_active),
    .row(row_in),
    .encode_busy(encode_busy),
    .transmit_busy(transmit_busy),
    .inter_row_cnt(inter_row_cnt),
    .block_read(block_read),
    .line_read(line_read),
    .data_output(data_output),
    .write_address(write_address),
    .wen(wen),
    .mult_column(mult_column),
    .column_select(),
    .output_select(output_select),
    .stagel_output(stagel_output),
    .shift_reg_output(shift_reg_output)
);

wireless_memory WIREMEM001 (
    .wr_clk(clock_27mhz),
    .wr_addr(write_address),
    .din(data_output[77:0]),
    .wen(wen),
    .r_clk(clock_27mhz),
    .read_addr(read_addr),
    .dout(dout)
);

alphanumeric_displays ALPHA1(
    .global_clock(clock_27mhz),
    .manual_reset(reset),
    .disp_test(1'b0),
    .disp_blank(disp_blank),
    .disp_clock(disp_clock),
    .disp_rs(disp_rs),
    .disp_ce_b(disp_ce_b),
    .disp_reset_b(disp_reset_b),
    .disp_data_out(disp_data_out),
    .dots(dots)
);

tx_control_unit tx_control(.clk(clock_27mhz),
    .reset(reset),
    .cts_b(rs232_cts),
    .read_data(dout),
    .encode_wr_addr(write_address),
    .rts_b(rs232_rts),
    .txd(rs232_txd),

```

```

        .read_addr(read_addr),
        .tx_busy(transmit_busy),
        .rs232_state(rs232_state),
        .state(tx_control_state),
        .shift_once(shift_once),
        .ready(ready),
        .enable(enable));

/* -----\ /----- EXCLUDED -----\ /-----
rs232_senderFSM_new sender(.clk(clock_27mhz),
        .reset(reset),
        .data(switch[7:0]),
        .cts_b(rs232_cts),
        .send(1'b1),
        .txd(rs232_txd),
        .rts_b(rs232_rts),
        .ready(ready),
        .state(state));

toplevel_test top(.clk(clock_27mhz),
        .reset(reset),
        .cts_b(rs232_cts),
        .txd(rs232_txd),
        .rts_b(rs232_rts));

-----\ /----- EXCLUDED -----\ /----- */
//assign rs232_rts = switch[0];

dots DOTS0 (.clk(clock_27mhz), .num(num0), .dots(dots0));
dots DOTS1 (.clk(clock_27mhz), .num(num1), .dots(dots1));
dots DOTS2 (.clk(clock_27mhz), .num(num2), .dots(dots2));
dots DOTS3 (.clk(clock_27mhz), .num(num3), .dots(dots3));
dots DOTS4 (.clk(clock_27mhz), .num(num4), .dots(dots4));
dots DOTS5 (.clk(clock_27mhz), .num(num5), .dots(dots5));
dots DOTS6 (.clk(clock_27mhz), .num(num6), .dots(dots6));
dots DOTS7 (.clk(clock_27mhz), .num(num7), .dots(dots7));

endmodule // labkit

```

```

module video_capture(clock_27mhz, tv_in_line_clock1, reset_button, encoder_busy,
    tv_in_ycrCb, vga_out_sync_b, vga_out_blank_b, vga_out_hsync,
    vga_out_vsync, vga_out, macro_line, write_enables, data_in,
    gray_addra, tv_in_reset_b, tv_in_i2c_clock, tv_in_i2c_data);

    input clock_27mhz, tv_in_line_clock1, reset_button, encoder_busy;
    input [19:0] tv_in_ycrCb;
    output vga_out_sync_b, vga_out_blank_b, vga_out_hsync, vga_out_vsync;
    output [23:0] vga_out;
    output [4:0] macro_line;
    output [63:0] data_in;
    output [7:0] write_enables;
    output [9:0] gray_addra;
    output tv_in_reset_b, tv_in_i2c_clock, tv_in_i2c_data;

wire [9:0] pixel_count, line_count, line_count_sync;
wire [7:0] R, G, B;
wire [23:0] vga_out;
wire source;
wire hsync, vsync, hblank, vblank;
wire [38:0] dout;
assign source = 1'b0;
wire [15:0] addra, addrb;
wire [9:0] gray_addra;
wire [23:0] doutb;
wire start, write;

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

debounce top_debounce(1'b0, clock_27mhz, ~reset_button, reset_sync);

// Instantiate VGA Controller

VGA_controller Controller(clock_27mhz, reset_sync, hsync, vsync, pixel_count,
    line_count, vga_out_sync_b, vga_out_blank_b, hblank, vblank, dout);

// Instantiate delay

delay sync_delay(clock_27mhz, hsync, vsync, vga_out_hsync, vga_out_vsync);

// Instantiate display

wire [23:0] RGB_out;

display top_display(clock_27mhz, R, G, B, pixel_count,
    line_count, doutb, addrb, vga_out, RGB_out);

// Top Level

wire [9:0] Y;
wire data_valid;
wire [7:0] Y_out;
wire encoder_busy;
wire [4:0] macro_line;

```

```
test pleasework(tv_in_line_clock1, clock_27mhz, clock_27mhz, reset_sync,
    tv_in_ycrCb, R, G, B, dout, doutb, addrb, start, line_count_sync, write,
    addra,
    pixel_count, line_count, RGB_out, Y, data_valid, Y_out, encoder_busy,
macro_line,
    write_enables, data_in, gray_addra);

adv7185init ADV(reset_sync, clock_27mhz, source, tv_in_reset_b,
    tv_in_i2c_clock, tv_in_i2c_data);

endmodule
```



```

module display(clk, R, G, B, pixel_count, line_count, doutb, addrb, vga_out,
RGB_out);
    input clk;
    input [7:0] R, G, B;
    input [9:0] pixel_count, line_count;
    input [23:0] doutb;
    input [23:0] RGB_out;
    output [23:0] vga_out;
    output [15:0] addrb;

reg [23:0] vga_out;
reg [15:0] addrb;

always @ (posedge clk)
begin
    if ((pixel_count < 240) && (pixel_count >= 0) && (line_count < 240))
        begin
            addrb = ((pixel_count == 0) && (line_count == 0))? 3 :
                    (addrb == 57599)? 0 : addrb + 1;
            vga_out = doutb;
        end
    else if ((pixel_count <= 542) && (pixel_count > 302) && (line_count <
240))
        vga_out = RGB_out;
    else
        begin
            addrb = addrb;
            vga_out = 24'habcdef;
        end
end

endmodule

```

```
// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output
module debounce (reset, clock, noisy, clean);
    parameter DELAY = 270000; // .01 sec with a 27Mhz clock
    input reset, clock, noisy;
    output clean;

    reg [18:0] count;
    reg new, clean;

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

endmodule
```

```

module VGA_controller(clk, reset, hsync, vsync, pixel_count, line_count,
    vga_out_sync_b, vga_out_blank_b, hblank, vblank, dout);
    input clk, reset;
    input [38:0] dout;
    output [9:0] line_count;
    output [9:0] pixel_count;
    output hsync, vsync, vga_out_sync_b, vga_out_blank_b, hblank, vblank;

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

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

assign hblank = (pixel_count < 640)? 1'b1 : 1'b0;
                //assign these based on counters
assign hsync = ((pixel_count < 751) && (pixel_count >= 655))? 1'b0 : 1'b1;
assign vblank = (line_count < 480)? 1'b1 : 1'b0;
assign vsync = ((line_count <= 492) && (line_count > 490))? 1'b0 : 1'b1;

always @(posedge clk) //use two
counters, one for pixel, one for line
begin
    //line incremented based on pixel
    if (reset)
        begin
            line_count <= 0;
            pixel_count <= 0;
        end
    else
        begin
pixel_count <= (pixel_count == 799) ? 10'b0 : pixel_count + 1'b1;
line_count <= ((pixel_count == 799) && (line_count == 524)) ? 10'b0 :
                (pixel_count == 799) ? line_count + 1'b1 :
line_count;
        end
end

endmodule

```

```

//
// File:   video_decoder.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
// This file contains the ntsc_decode and adv7185init modules
//
// These modules are used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//

////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrCb, ycrCb, f, v, h, data_valid,
current_state);

    // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
    // reset - system reset
    // tv_in_ycrCb - 10-bit input from chip. should map to pins [19:10]
    // ycrCb - 24 bit luminance and chrominance (8 bits each)
    // f - field: 1 indicates an even field, 0 an odd field
    // v - vertical sync: 1 means vertical sync
    // h - horizontal sync: 1 means horizontal sync

    input clk;
    input reset;
    input [9:0] tv_in_ycrCb; // modified for 10 bit input - should be P[19:10]
    output [29:0] ycrCb;
    output f;
    output v;
    output h;
    output data_valid;
    output [4:0] current_state;
    // output [4:0] state;

    parameter SYNC_1 = 0;
    parameter SYNC_2 = 1;
    parameter SYNC_3 = 2;
    parameter SAV_f1_cb0 = 3;
    parameter SAV_f1_y0 = 4;
    parameter SAV_f1_cr1 = 5;
    parameter SAV_f1_y1 = 6;
    parameter EAV_f1 = 7;
    parameter SAV_VBI_f1 = 8;
    parameter EAV_VBI_f1 = 9;
    parameter SAV_f2_cb0 = 10;
    parameter SAV_f2_y0 = 11;
    parameter SAV_f2_cr1 = 12;

```

```

parameter      SAV_f2_y1 = 13;
parameter      EAV_f2 = 14;
parameter      SAV_VBI_f2 = 15;
parameter      EAV_VBI_f2 = 16;

// In the start state, the module doesn't know where
// in the sequence of pixels, it is looking.

// Once we determine where to start, the FSM goes through a normal
// sequence of SAV process_YCrCb EAV... repeat

// The data stream looks as follows
// SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... |
EAV sequence
// There are two things we need to do:
// 1. Find the two SAV blocks (stands for Start Active Video perhaps?)
// 2. Decode the subsequent data

reg [4:0]       current_state = 5'h00;
reg [9:0]       y = 10'h000; // luminance
reg [9:0]       cr = 10'h000; // chrominance
reg [9:0]       cb = 10'h000; // more chrominance

assign state = current_state;

always @ (posedge clk)
begin
    if (reset)
        begin

            end
        else
            begin
                // these states don't do much except allow us to know where we are in
the stream.
                // whenever the synchronization code is seen, go back to the
sync_state before
                // transitioning to the new state
                case (current_state)
                    SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 :
SYNC_1;
                    SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 :
SYNC_1;
                    SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
(tv_in_ycrcb == 10'h274) ? EAV_f1 :
(tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
(tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
(tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
(tv_in_ycrcb == 10'h368) ? EAV_f2 :
(tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
(tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

                    SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_y0;
                    SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_cr1;

```

```

        SAV_f1_cr1: current_state <= (tv_in_ycrCb == 10'h3ff) ? SYNC_1 :
SAV_f1_y1;
        SAV_f1_y1: current_state <= (tv_in_ycrCb == 10'h3ff) ? SYNC_1 :
SAV_f1_cb0;

        SAV_f2_cb0: current_state <= (tv_in_ycrCb == 10'h3ff) ? SYNC_1 :
SAV_f2_y0;
        SAV_f2_y0: current_state <= (tv_in_ycrCb == 10'h3ff) ? SYNC_1 :
SAV_f2_cr1;
        SAV_f2_cr1: current_state <= (tv_in_ycrCb == 10'h3ff) ? SYNC_1 :
SAV_f2_y1;
        SAV_f2_y1: current_state <= (tv_in_ycrCb == 10'h3ff) ? SYNC_1 :
SAV_f2_cb0;

signals
    // These states are here in the event that we want to cover these
    // in the future. For now, they just send the state machine back to
SYNC_1
    EAV_f1: current_state <= SYNC_1;
    SAV_VBI_f1: current_state <= SYNC_1;
    EAV_VBI_f1: current_state <= SYNC_1;
    EAV_f2: current_state <= SYNC_1;
    SAV_VBI_f2: current_state <= SYNC_1;
    EAV_VBI_f2: current_state <= SYNC_1;

    endcase
end
end // always @ (posedge clk)

// implement our decoding mechanism

wire y_enable;
wire cr_enable;
wire cb_enable;

// if y is coming in, enable the register
// likewise for cr and cb
assign y_enable = (current_state == SAV_f1_y0) ||
    (current_state == SAV_f1_y1) ||
    (current_state == SAV_f2_y0) ||
    (current_state == SAV_f2_y1);
assign cr_enable = (current_state == SAV_f1_cr1) ||
    (current_state == SAV_f2_cr1);
assign cb_enable = (current_state == SAV_f1_cb0) ||
    (current_state == SAV_f2_cb0);

// f, v, and h only go high when active
assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrCb[7:6] : {v,h};

// data is valid when we have all three values: y, cr, cb
assign data_valid = y_enable;
assign ycrCb = {y,cr,cb};

reg        f = 0;

always @ (posedge clk)
    begin

```

```

        y <= y_enable ? tv_in_ycrcb : y;
        cr <= cr_enable ? tv_in_ycrcb : cr;
        cb <= cb_enable ? tv_in_ycrcb : cb;
        f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
    end

endmodule

/////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////
// Register 0
/////////////////////////////////////////////////////////////////

`define INPUT_SELECT                                4'h0
    // 0: CVBS on AIN1 (composite video in)
    // 7: Y on AIN2, C on AIN5 (s-video in)
    // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                                  4'h0
    // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
    // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
    // 2: Autodetect: NTSC or PAL (N), w/o pedestal
    // 3: Autodetect: NTSC or PAL (N), w/pedestal
    // 4: NTSC w/o pedestal
    // 5: NTSC w/pedestal
    // 6: NTSC 4.43 w/o pedestal
    // 7: NTSC 4.43 w/pedestal
    // 8: PAL BGHID w/o pedestal
    // 9: PAL N w/pedestal
    // A: PAL M w/o pedestal
    // B: PAL M w/pedestal
    // C: PAL combination N
    // D: PAL combination N w/pedestal
    // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

/////////////////////////////////////////////////////////////////
// Register 1
/////////////////////////////////////////////////////////////////

`define VIDEO_QUALITY                              2'h0
    // 0: Broadcast quality
    // 1: TV quality
    // 2: VCR quality
    // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                       1'b0
    // 0: Normal mode

```

```

// 1: Square pixel mode
`define DIFFERENTIAL_INPUT 1'b0
// 0: Single-ended inputs
// 1: Differential inputs
`define FOUR_TIMES_SAMPLING 1'b0
// 0: Standard sampling rate
// 1: 4x sampling rate (NTSC only)
`define BETACAM 1'b0
// 0: Standard video input
// 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE 1'b1
// 0: Change of input triggers reacquire
// 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM,
`FOUR_TIMES_SAMPLING, `DIFFERENTIAL_INPUT, `SQUARE_PIXEL_IN_MODE,
`VIDEO_QUALITY}

////////////////////////////////////
// Register 2
////////////////////////////////////

`define Y_PEAKING_FILTER 3'h4
// 0: Composite = 4.5dB, s-video = 9.25dB
// 1: Composite = 4.5dB, s-video = 9.25dB
// 2: Composite = 4.5dB, s-video = 5.75dB
// 3: Composite = 1.25dB, s-video = 3.3dB
// 4: Composite = 0.0dB, s-video = 0.0dB
// 5: Composite = -1.25dB, s-video = -3.0dB
// 6: Composite = -1.75dB, s-video = -8.0dB
// 7: Composite = -3.0dB, s-video = -8.0dB
`define CORING 2'h0
// 0: No coring
// 1: Truncate if Y < black+8
// 2: Truncate if Y < black+16
// 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

////////////////////////////////////
// Register 3
////////////////////////////////////

`define INTERFACE_SELECT 2'h0
// 0: Philips-compatible
// 1: Broktree API A-compatible
// 2: Broktree API B-compatible
// 3: [Not valid]
`define OUTPUT_FORMAT 4'h0
// 0: 10-bit @ LLC, 4:2:2 CCIR656
// 1: 20-bit @ LLC, 4:2:2 CCIR656
// 2: 16-bit @ LLC, 4:2:2 CCIR656
// 3: 8-bit @ LLC, 4:2:2 CCIR656
// 4: 12-bit @ LLC, 4:1:1
// 5-F: [Not valid]
// (Note that the 6.111 labkit hardware provides only a 10-bit interface to
// the ADV7185.)

```



```

`define TRISTATE_OUTPUT_DRIVERS                1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                            1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS,
`OUTPUT_FORMAT, `INTERFACE_SELECT}

////////////////////////////////////
// Register 4
////////////////////////////////////

`define OUTPUT_DATA_RANGE                    1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                            1'b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110, `OUTPUT_DATA_RANGE}

////////////////////////////////////
// Register 5
////////////////////////////////////

`define GENERAL_PURPOSE_OUTPUTS              4'b0000
`define GPO_0_1_ENABLE                       1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                       1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI                  1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                         1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI,
`GPO_2_3_ENABLE, `GPO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}

////////////////////////////////////
// Register 7
////////////////////////////////////

`define FIFO_FLAG_MARGIN                     5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                           1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET                 1'b0
  // 0: No automatic reset
  // 1: FIFO is automatically reset at the end of each video field

```

```

`define FIFO_FLAG_SELF_TIME                1'b1
    // 0: FIFO flags are synchronized to CLKIN
    // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET,
`FIFO_RESET, `FIFO_FLAG_MARGIN}

////////////////////////////////////
// Register 8
////////////////////////////////////

`define INPUT_CONTRAST_ADJUST                8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////
// Register 9
////////////////////////////////////

`define INPUT_SATURATION_ADJUST            8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////
// Register A
////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST            8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////
// Register B
////////////////////////////////////

`define INPUT_HUE_ADJUST                    8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////
// Register C
////////////////////////////////////

`define DEFAULT_VALUE_ENABLE                1'b0
    // 0: Use programmed Y, Cr, and Cb values
    // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE      1'b0
    // 0: Use programmed Y, Cr, and Cb values
    // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                      6'h0C
    // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE, `DEFAULT_VALUE_AUTOMATIC_ENABLE,
`DEFAULT_VALUE_ENABLE}

////////////////////////////////////
// Register D

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
`define DEFAULT_CR_VALUE                4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Register E
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE      1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL     2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE        4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE,
`TEMPORAL_DECIMATION_CONTROL, `TEMPORAL_DECIMATION_ENABLE}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Register F
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL              2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY      1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE            1'b0
  // 0: Reference is functional
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR        1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                 1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                       1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

```

```

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP,
`POWER_DOWN_LLC_GENERATOR, `POWER_DOWN_REFERENCE, `POWER_DOWN_SOURCE_PRIORITY,
`POWER_SAVE_CONTROL}

////////////////////////////////////
// Register 33
////////////////////////////////////

`define PEAK_WHITE_UPDATE                1'b1
    // 0: Update gain once per line
    // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES      1'b1
    // 0: Use lines 33 to 310
    // 1: Use lines 33 to 270
`define MAXIMUM_IRE                      3'h0
    // 0: PAL: 133, NTSC: 122
    // 1: PAL: 125, NTSC: 115
    // 2: PAL: 120, NTSC: 110
    // 3: PAL: 115, NTSC: 105
    // 4: PAL: 110, NTSC: 100
    // 5: PAL: 105, NTSC: 100
    // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                      1'b1
    // 0: Disable color kill
    // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE,
`AVERAGE_BIRIGHTNESS_LINES, `PEAK_WHITE_UPDATE}

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0

```

```

`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80

```

```

module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                   tv_in_i2c_clock, tv_in_i2c_data);

```

```

    input reset;
    input clock_27mhz;
    output tv_in_reset_b; // Reset signal to ADV7185
    output tv_in_i2c_clock; // I2C clock output to ADV7185
    output tv_in_i2c_data; // I2C data line to ADV7185
    input source; // 0: composite, 1: s-video

```

```

    initial begin

```

```

        $display("ADV7185 Initialization values:");
        $display(" Register 0: 0x%X", `ADV7185_REGISTER_0);
        $display(" Register 1: 0x%X", `ADV7185_REGISTER_1);
        $display(" Register 2: 0x%X", `ADV7185_REGISTER_2);
        $display(" Register 3: 0x%X", `ADV7185_REGISTER_3);
        $display(" Register 4: 0x%X", `ADV7185_REGISTER_4);
        $display(" Register 5: 0x%X", `ADV7185_REGISTER_5);
        $display(" Register 7: 0x%X", `ADV7185_REGISTER_7);
        $display(" Register 8: 0x%X", `ADV7185_REGISTER_8);
        $display(" Register 9: 0x%X", `ADV7185_REGISTER_9);
        $display(" Register A: 0x%X", `ADV7185_REGISTER_A);
        $display(" Register B: 0x%X", `ADV7185_REGISTER_B);
        $display(" Register C: 0x%X", `ADV7185_REGISTER_C);
        $display(" Register D: 0x%X", `ADV7185_REGISTER_D);
        $display(" Register E: 0x%X", `ADV7185_REGISTER_E);
        $display(" Register F: 0x%X", `ADV7185_REGISTER_F);
        $display(" Register 33: 0x%X", `ADV7185_REGISTER_33);

```

```

    end

```

```

    //
    // Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
    //

```

```

reg [7:0] clk_div_count, reset_count;
reg clock_slow;
wire reset_slow;

initial
begin
    clk_div_count <= 8'h00;
    // synthesis attribute init of clk_div_count is "00";
    clock_slow <= 1'b0;
    // synthesis attribute init of clock_slow is "0";
end

always @(posedge clock_27mhz)
if (clk_div_count == 26)
begin
    clock_slow <= ~clock_slow;
    clk_div_count <= 0;
end
else
    clk_div_count <= clk_div_count+1;

always @(posedge clock_27mhz)
if (reset)
    reset_count <= 100;
else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
        .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
        .sda(tv_in_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_in_reset_b;
reg old_source;

always @(posedge clock_slow)
if (reset_slow)
begin
    state <= 0;
    load <= 0;
    tv_in_reset_b <= 0;
    old_source <= 0;
end
end

```

```

else
case (state)
8'h00:
begin
// Assert reset
load <= 1'b0;
tv_in_reset_b <= 1'b0;
if (!ack)
state <= state+1;
end
8'h01:
state <= state+1;
8'h02:
begin
// Release reset
tv_in_reset_b <= 1'b1;
state <= state+1;
end
8'h03:
begin
// Send ADV7185 address
data <= 8'h8A;
load <= 1'b1;
if (ack)
state <= state+1;
end
8'h04:
begin
// Send subaddress of first register
data <= 8'h00;
if (ack)
state <= state+1;
end
8'h05:
begin
// Write to register 0
data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
if (ack)
state <= state+1;
end
8'h06:
begin
// Write to register 1
data <= `ADV7185_REGISTER_1;
if (ack)
state <= state+1;
end
8'h07:
begin
// Write to register 2
data <= `ADV7185_REGISTER_2;
if (ack)
state <= state+1;
end
8'h08:
begin
// Write to register 3

```

```

        data <= `ADV7185_REGISTER_3;
        if (ack)
            state <= state+1;
    end
8'h09:
    begin
        // Write to register 4
        data <= `ADV7185_REGISTER_4;
        if (ack)
            state <= state+1;
    end
8'h0A:
    begin
        // Write to register 5
        data <= `ADV7185_REGISTER_5;
        if (ack)
            state <= state+1;
    end
8'h0B:
    begin
        // Write to register 6
        data <= 8'h00; // Reserved register, write all zeros
        if (ack)
            state <= state+1;
    end
8'h0C:
    begin
        // Write to register 7
        data <= `ADV7185_REGISTER_7;
        if (ack)
            state <= state+1;
    end
8'h0D:
    begin
        // Write to register 8
        data <= `ADV7185_REGISTER_8;
        if (ack)
            state <= state+1;
    end
8'h0E:
    begin
        // Write to register 9
        data <= `ADV7185_REGISTER_9;
        if (ack)
            state <= state+1;
    end
8'h0F: begin
    // Write to register A
    data <= `ADV7185_REGISTER_A;
    if (ack)
        state <= state+1;
end
8'h10:
    begin
        // Write to register B
        data <= `ADV7185_REGISTER_B;
        if (ack)

```



```

        state <= state+1;
    end
8'h11:
    begin
        // Write to register C
        data <= `ADV7185_REGISTER_C;
        if (ack)
            state <= state+1;
    end
8'h12:
    begin
        // Write to register D
        data <= `ADV7185_REGISTER_D;
        if (ack)
            state <= state+1;
    end
8'h13:
    begin
        // Write to register E
        data <= `ADV7185_REGISTER_E;
        if (ack)
            state <= state+1;
    end
8'h14:
    begin
        // Write to register F
        data <= `ADV7185_REGISTER_F;
        if (ack)
            state <= state+1;
    end
8'h15:
    begin
        // Wait for I2C transmitter to finish
        load <= 1'b0;
        if (idle)
            state <= state+1;
    end
8'h16:
    begin
        // Write address
        data <= 8'h8A;
        load <= 1'b1;
        if (ack)
            state <= state+1;
    end
8'h17:
    begin
        data <= 8'h33;
        if (ack)
            state <= state+1;
    end
8'h18:
    begin
        data <= `ADV7185_REGISTER_33;
        if (ack)
            state <= state+1;
    end
end

```

```

8'h19:
  begin
    load <= 1'b0;
    if (idle)
      state <= state+1;
  end

8'h1A: begin
  data <= 8'h8A;
  load <= 1'b1;
  if (ack)
    state <= state+1;
end
8'h1B:
  begin
    data <= 8'h33;
    if (ack)
      state <= state+1;
  end
8'h1C:
  begin
    load <= 1'b0;
    if (idle)
      state <= state+1;
  end
8'h1D:
  begin
    load <= 1'b1;
    data <= 8'h8B;
    if (ack)
      state <= state+1;
  end
8'h1E:
  begin
    data <= 8'hFF;
    if (ack)
      state <= state+1;
  end
8'h1F:
  begin
    load <= 1'b0;
    if (idle)
      state <= state+1;
  end
8'h20:
  begin
    // Idle
    if (old_source != source) state <= state+1;
    old_source <= source;
  end
8'h21: begin
  // Send ADV7185 address
  data <= 8'h8A;
  load <= 1'b1;
  if (ack) state <= state+1;
end
8'h22: begin

```

```

        // Send subaddress of register 0
        data <= 8'h00;
        if (ack) state <= state+1;
    end
    8'h23: begin
        // Write to register 0
        data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
        if (ack) state <= state+1;
    end
    8'h24: begin
        // Wait for I2C transmitter to finish
        load <= 1'b0;
        if (idle) state <= 8'h20;
    end
endcase

endmodule

// i2c module for use with the ADV7185

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

    input reset;
    input clock4x;
    input [7:0] data;
    input load;
    output ack;
    output idle;
    output scl;
    output sda;

    reg [7:0] ldata;
    reg ack, idle;
    reg scl;
    reg sdai;

    reg [7:0] state;

    assign sda = sdai ? 1'bZ : 1'b0;

    always @(posedge clock4x)
        if (reset)
            begin
                state <= 0;
                ack <= 0;
            end
        else
            case (state)
            8'h00: // idle
                begin
                    scl <= 1'b1;
                    sdai <= 1'b1;
                    ack <= 1'b0;
                    idle <= 1'b1;
                    if (load)
                        begin
                            ldata <= data;
                        end
                end
            endcase

```

```

        ack <= 1'b1;
        state <= state+1;
    end
end
8'h01: // Start
begin
    ack <= 1'b0;
    idle <= 1'b0;
    sdai <= 1'b0;
    state <= state+1;
end
8'h02:
begin
    scl <= 1'b0;
    state <= state+1;
end
8'h03: // Send bit 7
begin
    ack <= 1'b0;
    sdai <= ldata[7];
    state <= state+1;
end
8'h04:
begin
    scl <= 1'b1;
    state <= state+1;
end
8'h05:
begin
    state <= state+1;
end
8'h06:
begin
    scl <= 1'b0;
    state <= state+1;
end
8'h07:
begin
    sdai <= ldata[6];
    state <= state+1;
end
8'h08:
begin
    scl <= 1'b1;
    state <= state+1;
end
8'h09:
begin
    state <= state+1;
end
8'h0A:
begin
    scl <= 1'b0;
    state <= state+1;
end
8'h0B:
begin

```

```

        sdai <= ldata[5];
        state <= state+1;
    end
8'h0C:
    begin
        scl <= 1'b1;
        state <= state+1;
    end
8'h0D:
    begin
        state <= state+1;
    end
8'h0E:
    begin
        scl <= 1'b0;
        state <= state+1;
    end
8'h0F:
    begin
        sdai <= ldata[4];
        state <= state+1;
    end
8'h10:
    begin
        scl <= 1'b1;
        state <= state+1;
    end
8'h11:
    begin
        state <= state+1;
    end
8'h12:
    begin
        scl <= 1'b0;
        state <= state+1;
    end
8'h13:
    begin
        sdai <= ldata[3];
        state <= state+1;
    end
8'h14:
    begin
        scl <= 1'b1;
        state <= state+1;
    end
8'h15:
    begin
        state <= state+1;
    end
8'h16:
    begin
        scl <= 1'b0;
        state <= state+1;
    end
8'h17:
    begin

```

```

        sdai <= ldata[2];
        state <= state+1;
    end
8'h18:
    begin
        scl <= 1'b1;
        state <= state+1;
    end
8'h19:
    begin
        state <= state+1;
    end
8'h1A:
    begin
        scl <= 1'b0;
        state <= state+1;
    end
8'h1B:
    begin
        sdai <= ldata[1];
        state <= state+1;
    end
8'h1C:
    begin
        scl <= 1'b1;
        state <= state+1;
    end
8'h1D:
    begin
        state <= state+1;
    end
8'h1E:
    begin
        scl <= 1'b0;
        state <= state+1;
    end
8'h1F:
    begin
        sdai <= ldata[0];
        state <= state+1;
    end
8'h20:
    begin
        scl <= 1'b1;
        state <= state+1;
    end
8'h21:
    begin
        state <= state+1;
    end
8'h22:
    begin
        scl <= 1'b0;
        state <= state+1;
    end
8'h23: // Acknowledge bit
    begin

```

```

        state <= state+1;
    end
8'h24:
    begin
        scl <= 1'b1;
        state <= state+1;
    end
8'h25:
    begin
        state <= state+1;
    end
8'h26:
    begin
        scl <= 1'b0;
        if (load)
            begin
                ldata <= data;
                ack <= 1'b1;
                state <= 3;
            end
        else
            state <= state+1;
        end
    end
8'h27:
    begin
        sdai <= 1'b0;
        state <= state+1;
    end
8'h28:
    begin
        scl <= 1'b1;
        state <= state+1;
    end
8'h29:
    begin
        sdai <= 1'b1;
        state <= 0;
    end
endcase

```

```
endmodule
```

```

module synchronizer(clk1, clk2, reset, tv_in_ycrbc, dout, write,
line_count_sync, pixel_count_sync,
    ycrbc, data_valid, current_state, tv_in_sync, almost_empty);
    input clk1, clk2, reset;
    input [19:0] tv_in_ycrbc;
    output [58:0] dout;
    output write;
    output [9:0] line_count_sync, pixel_count_sync;
    output [29:0] ycrbc;
    output data_valid;
    output [4:0] current_state;
    output [19:0] tv_in_sync;
    output almost_empty;

    wire [29:0] ycrbc;
    wire f, v, h, data_valid;
    wire [58:0] din;
    wire [4:0] current_state;
    wire full, empty, almost_full, almost_empty;
    //assign din = {tv_in_ycrbc, current_state, data_valid, f, v, h, ycrbc};

    reg wr_en = 0;
    reg rd_en = 0;
    reg [9:0] pixel_count_sync, line_count_sync;
    reg write;
    wire [58:0] dout_int;
    wire [58:0] dout;

    // Totally different approach
    wire [19:0] tv_in_sync;
    assign din = tv_in_ycrbc;
    assign tv_in_sync = dout;

    ntsc_decode decoder(clk1, reset, tv_in_ycrbc [19:10], ycrbc, f, v, h,
data_valid, current_state);

endmodule

```



```

module set_address(clk, reset, data_counter, data_counter2, data_out,
pixel_count_internal,
    line_count_internal, addr, iteration_counter, one_time, write_enables,
write_counter, write_counter_int,
    encoder_busy, macro_line);
    input clk, reset, encoder_busy;
    input [3:0] data_counter;
    input [4:0] data_counter2;
    input [63:0] data_out;
    input [9:0] pixel_count_internal, line_count_internal;
    output [9:0] addr;
    output [7:0] iteration_counter, write_enables;
    output one_time;
    output [3:0] write_counter, write_counter_int;
    output [4:0] macro_line;

reg [9:0] addr, base_addr;
reg [7:0] iteration_counter;
reg one_time, one_write_cycle;
reg [7:0] write_enables;
reg [3:0] write_counter, write_counter_int;
reg [4:0] macro_line;

reg encoder_busy_sync1, encoder_busy_sync2, halt;

always @ (posedge clk)
begin
    encoder_busy_sync1 <= encoder_busy;
    encoder_busy_sync2 <= encoder_busy_sync1;
end

always @ (posedge clk)
begin
    if (reset)
        begin
            addr <= 0;
            write_counter_int <= 0;
            iteration_counter <= 0;
            one_time <= 0;
            one_write_cycle <= 0;
            macro_line <= 0;
        end
    //new

else if (line_count_internal > 251)
    begin
        addr <= 0;
        write_counter_int <= 0;
        iteration_counter <= 0;
        one_time <= 0;
        one_write_cycle <= 0;
        macro_line <= 0;
    end
else if ((line_count_internal > 11) && (line_count_internal <= 251))
    begin
        if (data_counter == 8)
            if (~halt)

```

```

if (data_counter2 == 29)
    //place halt loop after data_counter loop

begin
case (write_counter)
0:
    if (~one_write_cycle)
        begin
            write_enables <= 8'b10000000;
            one_write_cycle <= 1;
        end
    else
        write_enables <= 8'h00;
1:
    if (~one_write_cycle)
        begin
            write_enables <= 8'b01000000;
            one_write_cycle <= 1;
        end
    else
        write_enables <= 8'h00;
2:
    if (~one_write_cycle)
        begin
            write_enables <= 8'b00100000;
            one_write_cycle <= 1;
        end
    else
        write_enables <= 8'h00;
3:
    if (~one_write_cycle)
        begin
            write_enables <= 8'b00010000;
            one_write_cycle <= 1;
        end
    else
        write_enables <= 8'h00;
4:
    if (~one_write_cycle)
        begin
            write_enables <= 8'b00001000;
            one_write_cycle <= 1;
        end
    else
        write_enables <= 8'h00;
5:
    if (~one_write_cycle)
        begin
            write_enables <= 8'b00000100;
            one_write_cycle <= 1;
        end
    else
        write_enables <= 8'h00;
6:
    if (~one_write_cycle)
        begin
            write_enables <= 8'b00000010;

```

```

        one_write_cycle <= 1;
        end
    else
        write_enables <= 8'h00;
7:
        if (~one_write_cycle)
            begin
                write_enables <= 8'b00000001;
                one_write_cycle <= 1;
                halt <= (encoder_busy_sync2)? 1 : 0;
            end
        else
            write_enables <= 8'h00;
default: write_enables <= 8'h00;
endcase

case (one_time)
0:
    begin
        iteration_counter <= (iteration_counter ==
239)? 0 : iteration_counter + 1;
        write_counter_int <= (write_counter_int ==
7)? 0 : write_counter_int + 1;
        macro_line <= ((macro_line == 29) &&
(write_counter_int == 7))? 0 :
            (write_counter_int == 7)? macro_line + 1 : macro_line;
        one_time <= 1;
    end
1:
    begin
        iteration_counter <= iteration_counter;
        write_counter_int <= write_counter_int;
        macro_line <= macro_line;
    end
endcase

end

else
    case (write_counter)
0:
        if (~one_write_cycle)
            begin
                write_enables <= 8'b10000000;
                one_write_cycle <= 1;
            end
        else
            write_enables <= 8'h00;
1:
        if (~one_write_cycle)
            begin
                write_enables <= 8'b01000000;
                one_write_cycle <= 1;
            end
        else
            write_enables <= 8'h00;
2:

```

```

        if (~one_write_cycle)
            begin
                write_enables <= 8'b00100000;
                one_write_cycle <= 1;
            end
        else
            write_enables <= 8'h00;
3:    if (~one_write_cycle)
            begin
                write_enables <= 8'b00010000;
                one_write_cycle <= 1;
            end
        else
            write_enables <= 8'h00;
4:    if (~one_write_cycle)
            begin
                write_enables <= 8'b00001000;
                one_write_cycle <= 1;
            end
        else
            write_enables <= 8'h00;
5:    if (~one_write_cycle)
            begin
                write_enables <= 8'b00000100;
                one_write_cycle <= 1;
            end
        else
            write_enables <= 8'h00;
6:    if (~one_write_cycle)
            begin
                write_enables <= 8'b00000010;
                one_write_cycle <= 1;
            end
        else
            write_enables <= 8'h00;
7:    if (~one_write_cycle)
            begin
                write_enables <= 8'b00000001;
                one_write_cycle <= 1;
            end
        else
            write_enables <= 8'h00;
default: write_enables <= 8'h00;
endcase

    else
        if ((data_counter == 8) && (data_counter2 == 29))
            halt <= (encoder_busy_sync2)? 1 : 0;
        else
            halt <= 1;
    else
        begin

```

```

        one_time <= 0;
        one_write_cycle <= 0;
        write_enables <= 0;
    end

/*      if ((data_counter2 == 29) && (data_counter == 8))
        begin
            case (one_time)
            0:
                begin
iteration_counter + 1;
write_counter_int + 1;
                    iteration_counter <= (iteration_counter == 239)? 0 :
                    write_counter_int <= (write_counter_int == 7)? 0 :
                    one_time <= 1;
                end
            1:
                begin
                    iteration_counter <= iteration_counter;
                    write_counter_int <= write_counter_int;
                end
            endcase
        end
    else if (data_counter == 8)
        case (write_counter)
        0:
            write_enables <= (write_enables == 8'h00)? 8'b10000000 :
8'h00;
        1:
            write_enables <= (write_enables == 8'h00)? 8'b01000000 :
8'h00;
        2:
            write_enables <= (write_enables == 8'h00)? 8'b00100000 :
8'h00;
        3:
            write_enables <= (write_enables == 8'h00)? 8'b00010000 :
8'h00;
        4:
            write_enables <= (write_enables == 8'h00)? 8'b00001000 :
8'h00;
        5:
            write_enables <= (write_enables == 8'h00)? 8'b00000100 :
8'h00;
        6:
            write_enables <= (write_enables == 8'h00)? 8'b00000010 :
8'h00;
        7:
            write_enables <= (write_enables == 8'h00)? 8'b00000001 :
8'h00;
        default: write_enables <= 8'h00;
        endcase
    else
        begin
            one_time <= 0;
            write_enables <= 0;
        end
    */

```

```
case (line_count_internal)
12:   base_addr <= 0;
20:   base_addr <= 30;
28:   base_addr <= 60;
36:   base_addr <= 90;
44:   base_addr <= 120;
52:   base_addr <= 150;
60:   base_addr <= 180;
68:   base_addr <= 210;
76:   base_addr <= 240;
84:   base_addr <= 270;
92:   base_addr <= 300;
100:  base_addr <= 330;
108:  base_addr <= 360;
116:  base_addr <= 390;
124:  base_addr <= 420;
132:  base_addr <= 450;
140:  base_addr <= 480;
148:  base_addr <= 510;
156:  base_addr <= 540;
164:  base_addr <= 570;
172:  base_addr <= 600;
180:  base_addr <= 630;
188:  base_addr <= 660;
196:  base_addr <= 690;
204:  base_addr <= 720;
212:  base_addr <= 750;
220:  base_addr <= 780;
228:  base_addr <= 810;
```

```
236:     base_addr <= 840;
244:     base_addr <= 870;
default: base_addr <= base_addr;
endcase

case (data_counter2)
0:     addr <= base_addr;
1:     addr <= base_addr + 1;
2:     addr <= base_addr + 2;
3:     addr <= base_addr + 3;
4:     addr <= base_addr + 4;
5:     addr <= base_addr + 5;
6:     addr <= base_addr + 6;
7:     addr <= base_addr + 7;
8:     addr <= base_addr + 8;
9:     addr <= base_addr + 9;
10:    addr <= base_addr + 10;
11:    addr <= base_addr + 11;
12:    addr <= base_addr + 12;
13:    addr <= base_addr + 13;
14:    addr <= base_addr + 14;
15:    addr <= base_addr + 15;
16:    addr <= base_addr + 16;
17:    addr <= base_addr + 17;
18:    addr <= base_addr + 18;
19:    addr <= base_addr + 19;
20:    addr <= base_addr + 20;
21:    addr <= base_addr + 21;
22:    addr <= base_addr + 22;
23:    addr <= base_addr + 23;
24:
```

```

        addr <= base_addr + 24;
25:    addr <= base_addr + 25;
26:    addr <= base_addr + 26;
27:    addr <= base_addr + 27;
28:    addr <= base_addr + 28;
29:    addr <= base_addr + 29;
default: addr <= addr;
endcase

    end
else
    begin
        addr <= 0;
        base_addr <= 0;
        write_enables <= 0;
        iteration_counter <= 0;
        write_counter_int <= 0;
    end
end

always @ (posedge clk)
begin
    write_counter <= write_counter_int;
end
endmodule

```



```

module store64(clk, reset, data_valid, Y, pixel_count_internal,
  data_out, data_counter, data_counter2, line_count_internal);
  input clk, reset, data_valid;
  input [7:0] Y;
  input [9:0] pixel_count_internal, line_count_internal;
  output [63:0] data_out;
  output [3:0] data_counter;
  output [4:0] data_counter2;

  wire [7:0] Y;
  wire [9:0] pixel_count_internal, line_count_internal;

  reg [63:0] data_out = 64'b0;
  reg [3:0] data_counter;
  reg [4:0] data_counter2;
  reg start, every_other;

  always @ (posedge clk)
  begin
    if (reset)
      begin
        data_out <= 0;
        data_counter <= 0;
        data_counter2 <= 0;
        start <= 0;
      end
    else if ((pixel_count_internal == 0) && (line_count_internal > 11) &&
(line_count_internal <= 251))
      begin
        start <= 1;
        data_counter <= 0; //changed from 0 to 1
        data_counter2 <= 0;
      end
    else if ((start) && (data_valid) && (every_other))
      case (data_counter)
        0:
          begin
            data_out[63:56] <= Y;
            data_counter <= data_counter + 1;
          end
        1:
          begin
            data_out[55:48] <= Y;
            data_counter <= data_counter + 1;
          end
        2:
          begin
            data_out[47:40] <= Y;
            data_counter <= data_counter + 1;
          end
        3:
          begin
            data_out[39:32] <= Y;
            data_counter <= data_counter + 1;
          end
        4:
          begin

```

```

        data_out[31:24] <= Y;
        data_counter <= data_counter + 1;
        end
    5:
        begin
            data_out[23:16] <= Y;
            data_counter <= data_counter + 1;
            end
    6:
        begin
            data_out[15:8] <= Y;
            data_counter <= data_counter + 1;
            end
    7:
        begin
            data_out[7:0] <= Y;
            data_counter <= data_counter + 1;
            end
    8:
        begin
            data_out[63:56] <= Y;
            //data_counter <= 0; //new line changed to 0 instead of
1
            data_counter <= (data_counter2 == 29)? 0: 1;
            data_counter2 <= (data_counter2 == 29)? 0 :
data_counter2 + 1;
            start <= (data_counter2 >= 29)? 0 : 1;
            end
        endcase
    else
        data_out <= data_out;
end

always @ (posedge clk)
begin
    if (reset)
        every_other <= 0;
    else if (data_valid)
        case (every_other)
        0:
            every_other <= 1;
        1:
            every_other <= 0;
        endcase
    else
        every_other <= every_other;
end
endmodule

```

```

module storeandsetaddr(clk, clk2, reset, data_valid, Y, pixel_count_internal,
line_count_internal,
    pixel_count, line_count, Y_out, encoder_busy, macro_line, write_enables,
data_in, addra);
    //addr);
    input clk, clk2, reset, data_valid, encoder_busy;
    input [7:0] Y;
    input [9:0] pixel_count_internal, line_count_internal;
    input [9:0] pixel_count, line_count; //addition
//    output [9:0] addr;
    output [7:0] Y_out;
    output [4:0] macro_line;
    output [63:0] data_in;
    output [7:0] write_enables;
    output [9:0] addra;

wire [63:0] data_in;
wire [63:0] data_out;
wire [63:0] dout0, dout1, dout2, dout3, dout4, dout5, dout6, dout7;
wire [3:0] data_counter;
wire [4:0] data_counter2;
wire [9:0] pixel_count_internal, line_count_internal;
wire [9:0] addra, addrb;
wire [7:0] iteration_counter, write_enables;
wire one_time;
wire [3:0] write_counter, write_counter_int;
wire [4:0] macro_line;

store64 store(clk, reset, data_valid, Y, pixel_count_internal,
    data_in, data_counter, data_counter2, line_count_internal);
//changed data_out to data_in
set_address addresser(clk, reset, data_counter, data_counter2, data_in,
pixel_count_internal,
    line_count_internal, addra, iteration_counter, one_time, write_enables,
write_counter, write_counter_int,
    encoder_busy, macro_line);

//new

    wire [3:0] read_counter;
    wire [4:0] read_counter2;
    wire [3:0] switch_dout_counter;
    wire [7:0] iteration_counter2;
    wire [63:0] first_eight, second_eight, dout;
    wire start;

/* -----\/\----- EXCLUDED -----\/\-----
encoder_memory memstructure(clk, clk2, addra, addrb, data_in, write_enables,
    dout0, dout1, dout2, dout3, dout4, dout5, dout6, dout7);
-----\/\----- EXCLUDED -----\/\----- */

read64 read(clk2, reset, Y_out, dout0, dout1, dout2, dout3, dout4, dout5, dout6,
dout7, addrb,
    pixel_count, line_count, read_counter, read_counter2, switch_dout_counter,
iteration_counter2,
    first_eight, second_eight, dout, start);

```

endmodule

```

/*****
**
** Module: ycrCb2rgb
**
** Generic Equations:
*****/

module YCrCb2RGB (R, G, B, clk, rst, Y, Cr, Cb);

output [7:0] R, G, B;

input clk,rst;
input[9:0] Y, Cr, Cb;

wire [7:0] R,G,B;
reg [20:0] R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
reg [9:0] const1,const2,const3,const4,const5;
reg[9:0] Y_reg, Cr_reg, Cb_reg;

//registering constants
always @ (posedge clk)
begin
    const1 = 10'b 0100101010; //1.164 = 01.00101010
    const2 = 10'b 0110011000; //1.596 = 01.10011000
    const3 = 10'b 0011010000; //0.813 = 00.11010000
    const4 = 10'b 0001100100; //0.392 = 00.01100100
    const5 = 10'b 1000000100; //2.017 = 10.00000100
end

always @ (posedge clk or posedge rst)
    if (rst)
        begin
            Y_reg <= 0;
            Cr_reg <= 0;
            Cb_reg <= 0;
        end
    else
        begin
            Y_reg <= Y;
            Cr_reg <= Cr;
            Cb_reg <= Cb;
        end
end

always @ (posedge clk or posedge rst)
    if (rst)
        begin
            A_int <= 0;
            B1_int <= 0;
            B2_int <= 0;
            C_int <= 0;
            X_int <= 0;
        end
    else
        begin
            X_int <= (const1 * (Y_reg - 'd64)) ;
            A_int <= (const2 * (Cr_reg - 'd512));
            B1_int <= (const3 * (Cr_reg - 'd512));

```

```

        B2_int <= (const4 * (Cb_reg - 'd512));
        C_int <= (const5 * (Cb_reg - 'd512));
    end

always @ (posedge clk or posedge rst)
    if (rst)
        begin
            R_int <= 0;
            G_int <= 0;
            B_int <= 0;
        end
    else
        begin
            R_int <= X_int + A_int;
            G_int <= X_int - B1_int - B2_int;
            B_int <= X_int + C_int;
        end

/*always @ (posedge clk or posedge rst)
    if (rst)
        begin
            R_int <= 0; G_int <= 0; B_int <= 0;
        end
    else
        begin
            X_int <= (const1 * (Y_reg - 'd64)) ;
            R_int <= X_int + (const2 * (Cr_reg - 'd512));
            G_int <= X_int - (const3 * (Cr_reg - 'd512)) - (const4 * (Cb_reg - 'd512));
            B_int <= X_int + (const5 * (Cb_reg - 'd512));
        end

*/
/* limit output to 0 - 4095, <0 equals 0 and >4095 equals 4095 */

assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] :
8'b11111111;
assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] :
8'b11111111;
assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] :
8'b11111111;

endmodule

```

```

module test(clk1, clk2, clk3, reset, tv_in_ycrCb, R, G, B, dout, doutb,
           addrb, start, line_count_sync, write, addra, pixel_count, line_count,
           RGB_out,
           Y, data_valid, Y_out, encoder_busy, macro_line, write_enables, data_in,
           gray_addra);
    input clk1, clk2, clk3, reset;
    input [19:0] tv_in_ycrCb;
    input [15:0] addrb;
    input [9:0] pixel_count, line_count;
    input encoder_busy;
    output start, write;
    output [7:0] R, G, B;
    output [9:0] line_count_sync;
    output [58:0] dout;
    output [23:0] doutb;
    output [15:0] addra;
    output [23:0] RGB_out;
    output [4:0] macro_line;
    output [63:0] data_in;
    output [7:0] write_enables;
    output [9:0] gray_addra;

    //new
    output [9:0] Y;
    output data_valid;
    output [7:0] Y_out;

wire [9:0] Y, Cr, Cb, line_count_sync, pixel_count_sync;
wire [23:0] RGB, dina, doutb;
wire start, write;

//new
wire [4:0] current_state;
wire data_valid;
wire [29:0] ycrCb;
wire [19:0] tv_in_sync;
wire [4:0] macro_line;

assign Y = ycrCb[29:20];
assign Cr = ycrCb[19:10];
assign Cb = ycrCb[9:0];

//assign tv_in_sync_19_10 = tv_in_sync [19:10];

/*    assign Y = dout [29:20];
    assign Cr = dout [19:10];
    assign Cb = dout [9:0]; */

assign RGB = {R, G, B};
assign dina = RGB;

reg wea, del_wea1, del_wea2, del_wea3;
reg [15:0] addra, del_addra1, del_addra2, del_addra3;
reg [9:0] pixel_count_internal, line_count_internal;
reg every_other;

wire almost_empty;

```

```

synchronizer sync(clk1, clk2, reset, tv_in_ycrCb, dout, write, line_count_sync,
pixel_count_sync,
    ycrCb, data_valid, current_state, tv_in_sync, almost_empty);

//synchronizer sync(clk1, clk2, reset, tv_in_ycrCb, dout, write,
line_count_test, pixel_count);
YCrCb2RGB converter(R, G, B, clk1, reset, Y, Cr, Cb);
//line_data lines(addrA, addrB, clk2, clk2, dina, doutB, wea);
line_data lines(del_addrA3, addrB, clk1, clk2, dina, doutB, del_wea3);

//new
wire [9:0] pixel_count, line_count;
wire [23:0] RGB_out;
assign RGB_out = {Y_out, Y_out, Y_out};
wire [9:0] gray_addrA;

storeandsetaddr readwrite(clk1, clk2, reset, data_valid, Y[9:2],
pixel_count_internal, line_count_internal,
    pixel_count, line_count, Y_out, encoder_busy, macro_line, write_enables,
data_in, gray_addrA);
/*
always @ (posedge clk1)
begin
if (reset)
    begin
        pixel_count_internal <= 0;
        line_count_internal <= 0;
        every_other <= 0;
    end
else if (~almost_empty)
    if ((current_state == 2) && (tv_in_sync[19:10] == 10'h2ac))
        line_count_internal <= 0;
    else if ((current_state == 2) && (tv_in_sync[19:10] == 10'h200))
        begin
            pixel_count_internal <= 0;
            line_count_internal <= line_count_internal + 1;
        end
    else if ((data_valid) && (pixel_count_internal <= 239))
        case (every_other)
        0:
            every_other <= 1;
        1:
            begin
                pixel_count_internal <= pixel_count_internal + 1;
                every_other <= 0;
            end
        endcase
    else
        begin
            pixel_count_internal <= pixel_count_internal;
            line_count_internal <= line_count_internal;
        end
else
    begin
        pixel_count_internal <= pixel_count_internal;
        line_count_internal <= line_count_internal;
    end

```



```

        end
end

always @ (posedge clk1)
begin
if (reset)
    begin
        wea <= 0;
        addra <= 0;
    end
else if ((pixel_count_internal == 0) && (line_count_internal == 1))
    addra <= 0;
else if (addra < 57600)
    case (data_valid)
    0:
        wea <= 0;
    1:
        if ((pixel_count_internal <= 239) && (line_count_internal > 11)
            && (line_count_internal <= 251) && (every_other == 0))
            begin
                wea <= 1;
                addra <= addra + 1;
            end
        else
            begin
                wea <= 0;
                addra <= addra;
            end
        endcase
    else
        begin
            wea <= 0;
            addra <= addra;
        end
    end
end
*/

always @ (posedge clk1)
begin
if (reset)
    begin
        pixel_count_internal <= 0;
        line_count_internal <= 0;
        every_other <= 0;
    end
else if ((current_state == 2) && (tv_in_ycrcb[19:10] == 10'h2ac)) //tv_in_ycrcb
used to be tv_in_sync
    line_count_internal <= 0;
else if ((current_state == 2) && (tv_in_ycrcb[19:10] == 10'h200))
    begin
        pixel_count_internal <= 0;
        line_count_internal <= line_count_internal + 1;
    end
else if ((data_valid) && (pixel_count_internal <= 239))
    case (every_other)
    0:
        every_other <= 1;

```

```

        1:
            begin
                pixel_count_internal <= pixel_count_internal + 1;
                every_other <= 0;
            end
        endcase
end

always @ (posedge clk1)
begin
    if (reset)
        begin
            wea <= 0;
            addra <= 0;
        end
    else if ((pixel_count_internal == 0) && (line_count_internal == 1))
        addra <= 0;
    else if (addra < 57600)
        case (data_valid)
            0:
                wea <= 0;
            1:
                if ((pixel_count_internal <= 239) && (line_count_internal > 11)
                    && (line_count_internal <= 251) && (every_other == 0))
                    begin
                        wea <= 1;
                        addra <= addra + 1;
                    end
                else
                    begin
                        wea <= 0;
                        addra <= addra;
                    end
                endcase
        else
            begin
                wea <= 0;
                addra <= addra;
            end
        end

always @ (posedge clk1)
begin
    del_wea1 <= wea;
    del_wea2 <= del_wea1;
    del_wea3 <= del_wea2;

    del_addra1 <= addra;
    del_addra2 <= del_addra1;
    del_addra3 <= del_addra2;
end

/*always @ (posedge clk2)
begin
    if (reset)
        begin
            wea <= 0;

```

```
        addra <= 0;
        end
    else if ((write) && (addra < 64000))
        begin
            wea <= 1;
            addra <= addra + 1;
        end
    else if ((pixel_count == 0) && (line_count_test == 1))
        begin
            wea <= 1;
            addra <= 0;
        end
    else
        begin
            wea <= 0;
            addra <= addra;
        end
    end
end */

endmodule
```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Encoder - Video Memory Module
// 05/13/2006
//
////////////////////////////////////

module video_memory(
    wr_clk,
    addra,
    dina,
    write_enables,
    r_clk,
    line_read,
    block_read,
    inter_row_cnt,
    addrb,
    output_row
);

input wr_clk, r_clk;
input [9:0] addra;
input [63:0] dina;
input [7:0] write_enables;

input [4:0] line_read, block_read;
input [2:0] inter_row_cnt;

output [9:0] addrb;
output [63:0] output_row;

reg [63:0] output_row;

wire [63:0] out0, out1, out2, out3, out4, out5, out6, out7;
wire [9:0] addrb;

assign addrb = 30*line_read + block_read;

wire wea0, wea1, wea2, wea3, wea4, wea5, wea6, wea7;

assign wea0 = write_enables[7];
assign wea1 = write_enables[6];
assign wea2 = write_enables[5];
assign wea3 = write_enables[4];
assign wea4 = write_enables[3];
assign wea5 = write_enables[2];
assign wea6 = write_enables[1];
assign wea7 = write_enables[0];

video_mem VIDEOMEM0(.clka(wr_clk), .clkb(r_clk), .wea(wea0), .addra(addra),
.addrb(addrb), .dina(dina), .doutb(out0));
video_mem VIDEOMEM1(.clka(wr_clk), .clkb(r_clk), .wea(wea1), .addra(addra),
.addrb(addrb), .dina(dina), .doutb(out1));

```

```

    video_mem VIDEOMEM2(.clka(wr_clk), .clkb(r_clk), .wea(wea2), .addra(addr),
    .addrb(addrb), .dina(dina), .doutb(out2));
    video_mem VIDEOMEM3(.clka(wr_clk), .clkb(r_clk), .wea(wea3), .addra(addr),
    .addrb(addrb), .dina(dina), .doutb(out3));
    video_mem VIDEOMEM4(.clka(wr_clk), .clkb(r_clk), .wea(wea4), .addra(addr),
    .addrb(addrb), .dina(dina), .doutb(out4));
    video_mem VIDEOMEM5(.clka(wr_clk), .clkb(r_clk), .wea(wea5), .addra(addr),
    .addrb(addrb), .dina(dina), .doutb(out5));
    video_mem VIDEOMEM6(.clka(wr_clk), .clkb(r_clk), .wea(wea6), .addra(addr),
    .addrb(addrb), .dina(dina), .doutb(out6));
    video_mem VIDEOMEM7(.clka(wr_clk), .clkb(r_clk), .wea(wea7), .addra(addr),
    .addrb(addrb), .dina(dina), .doutb(out7));

    always @(out0 or out1 or out2 or out3 or out4 or out5 or out6 or out7 or
inter_row_cnt)
        case (inter_row_cnt)
            0: output_row = out0;
            1: output_row = out1;
            2: output_row = out2;
            3: output_row = out3;
            4: output_row = out4;
            5: output_row = out5;
            6: output_row = out6;
            7: output_row = out7;
        endcase // case(inter_row_cnt)

endmodule // video_memory

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Encoder
// 05/05/2006
//
////////////////////////////////////

```

```

module encoder(
    clk,
    reset,
    resolution_select,
    max_lines,
    max_blocks,
    line_active,
    row,
    encode_busy,
    transmit_busy,
    inter_row_cnt,
    block_read,
    line_read,
    data_output,
    write_address,
    wen,
    mult_column,
    column_select,
    output_select,
    stagel_output,
    shift_reg_output
);

input clk, reset;
input [2:0] resolution_select;
input [4:0] max_lines, max_blocks, line_active;
input [71:0] row;
input transmit_busy;

output encode_busy;
output [4:0] block_read, line_read;
output [2:0] inter_row_cnt;
output [194:0] data_output;
output [9:0] write_address;
output wen;
output [103:0] mult_column;
output [2:0] column_select;
output [2:0] output_select;
output [17:0] stagel_output;
output [143:0] shift_reg_output;

wire [2:0] column_select, output_select;
wire [103:0] mult_column;
wire done;
wire [17:0] stagel_output;
wire [143:0] shift_reg_output;

```

```

// Instantiate Multiply Unit
dct_multiply DCTMULTIPLY1(
    .clk(clk),
    .reset(reset),
    .row(row),
    .column_select(column_select),
    .output_column(mult_column),
    .stage1_output(stage1_output),
    .shift_reg_output(shift_reg_output),
    .done(done)
);

// Instantiate Encoder FSM
encode_fsm ENCODEFSM1(
    .clk(clk),
    .reset(reset),
    .resolution_select(resolution_select),
    .max_blocks(max_blocks),
    .max_lines(max_lines),
    .transmit_busy(transmit_busy),
    .line_active(line_active),
    .line_read(line_read),
    .block_read(block_read),
    .column_select(column_select),
    .inter_row_cnt(inter_row_cnt),
    .output_select(output_select),
    .encode_busy(encode_busy),
    .state()
);

// Instantiate Variable Shift Register
encode_memory_register ENCODEMEMREG1(.clk(clk),
    .reset(reset),
    .load_select(output_select),
    .max_lines(max_lines),
    .max_blocks(max_blocks),
    .resolution_select(resolution_select),
    .coef7(mult_column[12:0] ),
    .coef6(mult_column[25:13]),
    .coef5(mult_column[38:26]),
    .coef4(mult_column[51:39]),
    .coef3(mult_column[64:52]),
    .coef2(mult_column[77:65]),
    .coef1(mult_column[90:78]),
    .coef0(mult_column[103:91]),
    .data_output(data_output),
    .address(write_address),
    .wen(wen));

endmodule // encoder

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// DCT Multiply Module - Encoder
// 04/24/2006
//
////////////////////////////////////

module dct_multiply(
    clk,
    reset,
    row,
    column_select,
    output_column,
    stagel_output,
    shift_reg_output,
    done
);

input clk, reset;
input [71:0] row;
input [2:0] column_select;

output [103:0] output_column;
output [17:0] stagel_output;
output [143:0] shift_reg_output;
output done;

wire [17:0] stagel_output;
wire [143:0] shift_reg_output;

wire [71:0] dct_row[7:0];
wire [71:0] dct_col[7:0];
reg [71:0] dct_row_final;

// Instantiate DCT Table
dct_table DCTTABLE0(.clk(clk),
    .row0(dct_row[0]),
    .row1(dct_row[1]),
    .row2(dct_row[2]),
    .row3(dct_row[3]),
    .row4(dct_row[4]),
    .row5(dct_row[5]),
    .row6(dct_row[6]),
    .row7(dct_row[7]),
    .col0(dct_col[0]),
    .col1(dct_col[1]),
    .col2(dct_col[2]),
    .col3(dct_col[3]),
    .col4(dct_col[4]),
    .col5(dct_col[5]),
    .col6(dct_col[6]),
    .col7(dct_col[7])
);

```



```

// Front set of multipliers and truncate-ors
dct_front DCTFRONT0(.clk(clk),
    .row(row),
    .column(dct_row_final),
    .output_coef(stagel_output));

// Shift Register
mult_shift_reg MULTSHFTREG1(
    .clk(clk),
    .reset(reset),
    .coef(stagel_output),
    .rdy(1'b1),
    .column(shift_reg_output),
    .done(done)
);

// Instantiate second set of multipliers
dct_back DCTBACK0(.clk(clk),
    .row0(dct_row[0]),
    .row1(dct_row[1]),
    .row2(dct_row[2]),
    .row3(dct_row[3]),
    .row4(dct_row[4]),
    .row5(dct_row[5]),
    .row6(dct_row[6]),
    .row7(dct_row[7]),
    .column(shift_reg_output),
    .output_column(output_column));

// Decides which DCT column to multiply against the input matrix
always @(column_select)
begin
    case (column_select)
        0: dct_row_final = dct_row[0];
        1: dct_row_final = dct_row[1];
        2: dct_row_final = dct_row[2];
        3: dct_row_final = dct_row[3];
        4: dct_row_final = dct_row[4];
        5: dct_row_final = dct_row[5];
        6: dct_row_final = dct_row[6];
        7: dct_row_final = dct_row[7];
    endcase // case(column_select)
end // always @ (column_select)

endmodule // dct_multiply

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// DCT Front - First stage of multiply module
// 04/24/2006
//
////////////////////////////////////

module dct_front(
    clk,
    row,
    column,
    output_trunc,
    output_coef
);

    input clk;
    input [71:0] row, column;

    output [17:0] output_coef;
    output [20:0] output_trunc;

    wire [20:0] coef;

    // Debugging vector
    assign output_trunc = coef;

    // Instantiate first set of multipliers
    matrix_naive MTX0(.clk(clk), .row(row), .column(column), .product(coef));

    // First bank of truncate-ors
    trunc_s1 TRUNCS100(.input_coef(coef), .output_coef(output_coef));

endmodule // dct_front

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Single Matrix Coefficient Generator - Naive Implementation
// 04/22/2006
//
////////////////////////////////////

module matrix_naive(
    clk,
    row,
    column,
    RX_0, RX_1, RX_2, RX_3, RX_4, RX_5, RX_6, RX_7,
    RI_0, RI_1, RI_2, RI_3,
    RM_0, RM_1,
    product);

input clk;
input [71:0] row, column;

output signed [17:0] RX_0, RX_1, RX_2, RX_3, RX_4, RX_5, RX_6, RX_7;
output signed [18:0] RI_0, RI_1, RI_2, RI_3;
output signed [19:0] RM_0, RM_1;
output signed [20:0] product;

wire signed [8:0] row_pixel [7:0];
wire signed [8:0] col_pixel [7:0];

wire signed [17:0] RX[7:0];
wire signed [18:0] RI[3:0];
wire signed [19:0] RM[1:0];
wire signed [20:0] product;

assign RX_0 = RX[0];
assign RX_1 = RX[1];
assign RX_2 = RX[2];
assign RX_3 = RX[3];
assign RX_4 = RX[4];
assign RX_5 = RX[5];
assign RX_6 = RX[6];
assign RX_7 = RX[7];

assign RI_0 = RI[0];
assign RI_1 = RI[1];
assign RI_2 = RI[2];
assign RI_3 = RI[3];

assign RM_0 = RM[0];
assign RM_1 = RM[1];

assign row_pixel[0] = row[8:0];
assign row_pixel[1] = row[17:9];
assign row_pixel[2] = row[26:18];
assign row_pixel[3] = row[35:27];

```

```

assign row_pixel[4] = row[44:36];
assign row_pixel[5] = row[53:45];
assign row_pixel[6] = row[62:54];
assign row_pixel[7] = row[71:63];

assign col_pixel[0] = column[8:0];
assign col_pixel[1] = column[17:9];
assign col_pixel[2] = column[26:18];
assign col_pixel[3] = column[35:27];
assign col_pixel[4] = column[44:36];
assign col_pixel[5] = column[53:45];
assign col_pixel[6] = column[62:54];
assign col_pixel[7] = column[71:63];

mult_sign_9 MULT0(.clk(clk), .a(row_pixel[0]), .b(col_pixel[0]), .q(RX[0]));
mult_sign_9 MULT1(.clk(clk), .a(row_pixel[1]), .b(col_pixel[1]), .q(RX[1]));
mult_sign_9 MULT2(.clk(clk), .a(row_pixel[2]), .b(col_pixel[2]), .q(RX[2]));
mult_sign_9 MULT3(.clk(clk), .a(row_pixel[3]), .b(col_pixel[3]), .q(RX[3]));
mult_sign_9 MULT4(.clk(clk), .a(row_pixel[4]), .b(col_pixel[4]), .q(RX[4]));
mult_sign_9 MULT5(.clk(clk), .a(row_pixel[5]), .b(col_pixel[5]), .q(RX[5]));
mult_sign_9 MULT6(.clk(clk), .a(row_pixel[6]), .b(col_pixel[6]), .q(RX[6]));
mult_sign_9 MULT7(.clk(clk), .a(row_pixel[7]), .b(col_pixel[7]), .q(RX[7]));

add_sign_18 ADD00(.A(RX[0]), .B(RX[1]), .S(RI[0]));
add_sign_18 ADD01(.A(RX[2]), .B(RX[3]), .S(RI[1]));
add_sign_18 ADD02(.A(RX[4]), .B(RX[5]), .S(RI[2]));
add_sign_18 ADD03(.A(RX[6]), .B(RX[7]), .S(RI[3]));

add_sign_19 ADD10(.A(RI[0]), .B(RI[1]), .S(RM[0]));
add_sign_19 ADD11(.A(RI[2]), .B(RI[3]), .S(RM[1]));

add_sign_20 ADD20(.A(RM[0]), .B(RM[1]), .S(product));

```

```
endmodule // matrix_naive
```

```
////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Truncate Stage 1 (21 bits -> 18 bits)
// 04/24/2006
//
////////////////////////////////////

module trunc_s1(
    input_coef,
    output_coef
);

    input [20:0] input_coef;
    output [17:0] output_coef;

    // Nature of input bits:
    // ( sign_bit integer_bits fractional_bits )
    // Input: (1 12 8)
    // Output: (1 10 7)
    assign output_coef = input_coef[18:1];

endmodule // trunc_s1
```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Encoder Internal Shift Register
// 05/12/2006
//
////////////////////////////////////

module mult_shift_reg(
    clk,
    reset,
    coef,
    rdy,
    column,
    done
);

input clk, reset;
input [17:0] coef;
input rdy;

output [143:0] column;
output done;

reg [2:0] reg_fill;
reg [143:0] column;
reg done;

parameter MAX_FILL = 8;

always @(posedge clk)
begin
    if (!reset)
    begin
        column <= 0;
        reg_fill <= 0;
        done <= 0;
    end
    else
    begin
        if (rdy)
        begin
            column <= {column[125:0], coef};
            if (reg_fill == MAX_FILL - 1)
                reg_fill <= 0;
            else
                reg_fill <= reg_fill + 1;
        end

        if (reg_fill == MAX_FILL - 1)
            done <= 1;
        else
            done <= 0;
    end
end

```

```
        end // else: !if(!reset)
    end // always @ (posedge clk)
endmodule // mult_shift_reg
```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// DCT Back - Second stage of multiply module
// 04/24/2006
//
////////////////////////////////////

module dct_back(
    clk,
    row0,
    row1,
    row2,
    row3,
    row4,
    row5,
    row6,
    row7,
    column,
    output_trunc,
    output_column
);

input clk;
input [71:0] row0, row1, row2, row3, row4, row5, row6, row7;
input [143:0] column;

output [103:0] output_column;
output [239:0] output_trunc;

wire [29:0] s2coef[7:0];

assign output_trunc[29:0]= s2coef[0];
assign output_trunc[59:30]= s2coef[1];
assign output_trunc[89:60]= s2coef[2];
assign output_trunc[119:90]= s2coef[3];
assign output_trunc[149:120]= s2coef[4];
assign output_trunc[179:150]= s2coef[5];
assign output_trunc[209:180]= s2coef[6];
assign output_trunc[239:210]= s2coef[7];

// Instantiate second set of multipliers
matrix_mult_s2 MTXMULTS2(.clk(clk),
    .row0(row0),
    .row1(row1),
    .row2(row2),
    .row3(row3),
    .row4(row4),
    .row5(row5),
    .row6(row6),
    .row7(row7),
    .column(column),
    .coef0(s2coef[0]),
    .coef1(s2coef[1]),

```



```

        .coef2(s2coef[2]),
        .coef3(s2coef[3]),
        .coef4(s2coef[4]),
        .coef5(s2coef[5]),
        .coef6(s2coef[6]),
        .coef7(s2coef[7]));

    // Second round of truncating
    trunc_s2 TRUNCS200(.input_coef(s2coef[0]), .output_coef(output_column[103:
91]));
    trunc_s2 TRUNCS201(.input_coef(s2coef[1]), .output_coef(output_column[ 90:
78]));
    trunc_s2 TRUNCS202(.input_coef(s2coef[2]), .output_coef(output_column[ 77:
65]));
    trunc_s2 TRUNCS203(.input_coef(s2coef[3]), .output_coef(output_column[ 64:
52]));
    trunc_s2 TRUNCS204(.input_coef(s2coef[4]), .output_coef(output_column[ 51:
39]));
    trunc_s2 TRUNCS205(.input_coef(s2coef[5]), .output_coef(output_column[ 38:
26]));
    trunc_s2 TRUNCS206(.input_coef(s2coef[6]), .output_coef(output_column[ 25:
13]));
    trunc_s2 TRUNCS207(.input_coef(s2coef[7]), .output_coef(output_column[ 12:
0]));

endmodule // dct_back

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Matrix Multiply -> Stage 2
// 04/22/2006
//
////////////////////////////////////

module matrix_mult_s2(
    clk,
    row0,
    row1,
    row2,
    row3,
    row4,
    row5,
    row6,
    row7,
    column,
    coef0,
    coef1,
    coef2,
    coef3,
    coef4,
    coef5,
    coef6,
    coef7
);

    input clk;
    input [71:0] row0, row1, row2, row3, row4, row5, row6, row7;
    input [143:0] column;

    output [29:0] coef0, coef1, coef2, coef3, coef4, coef5, coef6, coef7;

    matrix_naive_s2_trunc MTX0(.clk(clk), .row(row0), .column(column),
    .product(coef0));
    matrix_naive_s2_trunc MTX1(.clk(clk), .row(row1), .column(column),
    .product(coef1));
    matrix_naive_s2_trunc MTX2(.clk(clk), .row(row2), .column(column),
    .product(coef2));
    matrix_naive_s2_trunc MTX3(.clk(clk), .row(row3), .column(column),
    .product(coef3));
    matrix_naive_s2_trunc MTX4(.clk(clk), .row(row4), .column(column),
    .product(coef4));
    matrix_naive_s2_trunc MTX5(.clk(clk), .row(row5), .column(column),
    .product(coef5));
    matrix_naive_s2_trunc MTX6(.clk(clk), .row(row6), .column(column),
    .product(coef6));
    matrix_naive_s2_trunc MTX7(.clk(clk), .row(row7), .column(column),
    .product(coef7));

endmodule // matrix_mult_s2

```



```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Single Matrix Coefficient Generator - Naive Implementation Stage 2 truncated
// 04/22/2006
//
////////////////////////////////////

```

```

module matrix_naive_s2_trunc(
    clk,
    row,
    column,
    RX_0, RX_1, RX_2, RX_3, RX_4, RX_5, RX_6, RX_7,
    RI_0, RI_1, RI_2, RI_3,
    RM_0, RM_1,
    product);

input clk;
input [71:0] row;
input [143:0] column;

output signed [26:0] RX_0, RX_1, RX_2, RX_3, RX_4, RX_5, RX_6, RX_7;
output signed [27:0] RI_0, RI_1, RI_2, RI_3;
output signed [28:0] RM_0, RM_1;
output signed [29:0] product;

wire signed [8:0] row_pixel [7:0];
wire signed [17:0] col_pixel [7:0];

wire signed [26:0] RX[7:0];
wire signed [27:0] RI[3:0];
wire signed [28:0] RM[1:0];
wire signed [29:0] product;

assign RX_0 = RX[0];
assign RX_1 = RX[1];
assign RX_2 = RX[2];
assign RX_3 = RX[3];
assign RX_4 = RX[4];
assign RX_5 = RX[5];
assign RX_6 = RX[6];
assign RX_7 = RX[7];

assign RI_0 = RI[0];
assign RI_1 = RI[1];
assign RI_2 = RI[2];
assign RI_3 = RI[3];

assign RM_0 = RM[0];
assign RM_1 = RM[1];

assign row_pixel[0] = row[8:0];
assign row_pixel[1] = row[17:9];
assign row_pixel[2] = row[26:18];
assign row_pixel[3] = row[35:27];
assign row_pixel[4] = row[44:36];
assign row_pixel[5] = row[53:45];
assign row_pixel[6] = row[62:54];
assign row_pixel[7] = row[71:63];

```

```

assign    col_pixel[0] = column[17:0];
assign    col_pixel[1] = column[35:18];
assign    col_pixel[2] = column[53:36];
assign    col_pixel[3] = column[71:54];
assign    col_pixel[4] = column[89:72];
assign    col_pixel[5] = column[107:90];
assign    col_pixel[6] = column[125:108];
assign    col_pixel[7] = column[143:126];

mult_sign_18_9 MULT0(.a(col_pixel[0]), .b(row_pixel[0]), .o(RX[0]));
mult_sign_18_9 MULT1(.a(col_pixel[1]), .b(row_pixel[1]), .o(RX[1]));
mult_sign_18_9 MULT2(.a(col_pixel[2]), .b(row_pixel[2]), .o(RX[2]));
mult_sign_18_9 MULT3(.a(col_pixel[3]), .b(row_pixel[3]), .o(RX[3]));
mult_sign_18_9 MULT4(.a(col_pixel[4]), .b(row_pixel[4]), .o(RX[4]));
mult_sign_18_9 MULT5(.a(col_pixel[5]), .b(row_pixel[5]), .o(RX[5]));
mult_sign_18_9 MULT6(.a(col_pixel[6]), .b(row_pixel[6]), .o(RX[6]));
mult_sign_18_9 MULT7(.a(col_pixel[7]), .b(row_pixel[7]), .o(RX[7]));

adder_sign_27 ADD00(.A(RX[0]), .B(RX[1]), .S(RI[0]));
adder_sign_27 ADD01(.A(RX[2]), .B(RX[3]), .S(RI[1]));
adder_sign_27 ADD02(.A(RX[4]), .B(RX[5]), .S(RI[2]));
adder_sign_27 ADD03(.A(RX[6]), .B(RX[7]), .S(RI[3]));

adder_sign_28 ADD10(.A(RI[0]), .B(RI[1]), .S(RM[0]));
adder_sign_28 ADD11(.A(RI[2]), .B(RI[3]), .S(RM[1]));

adder_sign_29 ADD20(.A(RM[0]), .B(RM[1]), .S(product));

```

```
endmodule // matrix_naive
```

```
////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Truncate Stage 2 (21 bits -> 18 bits)
// 04/24/2006
//
////////////////////////////////////

module trunc_s2(
    input_coef,
    output_coef
);

    input [29:0] input_coef;
    output [12:0] output_coef;

    // Nature of input bits:
    // ( sign_bit integer_bits fractional_bits )
    // Input: (1 14 15)
    // Output: (1 12 0)
    assign output_coef = input_coef[27:15];

endmodule // trunc_s2
```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// DCT Matrix
// 04/22/2006
//
////////////////////////////////////

module dct_table(clk,
                row0, row1, row2, row3, row4, row5, row6, row7,
                col0, col1, col2, col3, col4, col5, col6, col7
                );

    input clk;

    output [71:0] row0, row1, row2, row3, row4, row5, row6, row7;
    output [71:0] col0, col1, col2, col3, col4, col5, col6, col7;

    reg          signed [8:0] dct_coef[63:0];
    wire [71:0]   row0, row1, row2, row3, row4, row5, row6, row7;
    wire [71:0]   col0, col1, col2, col3, col4, col5, col6, col7;

    // Assigning Coefficients to Rows and Columns
    assign row0 = {dct_coef[0],  dct_coef[1],  dct_coef[2],  dct_coef[3],
dct_coef[4],  dct_coef[5],  dct_coef[6],  dct_coef[7]};
    assign row1 = {dct_coef[8],  dct_coef[9],  dct_coef[10], dct_coef[11],
dct_coef[12], dct_coef[13], dct_coef[14], dct_coef[15]};
    assign row2 = {dct_coef[16], dct_coef[17], dct_coef[18], dct_coef[19],
dct_coef[20], dct_coef[21], dct_coef[22], dct_coef[23]};
    assign row3 = {dct_coef[24], dct_coef[25], dct_coef[26], dct_coef[27],
dct_coef[28], dct_coef[29], dct_coef[30], dct_coef[31]};
    assign row4 = {dct_coef[32], dct_coef[33], dct_coef[34], dct_coef[35],
dct_coef[36], dct_coef[37], dct_coef[38], dct_coef[39]};
    assign row5 = {dct_coef[40], dct_coef[41], dct_coef[42], dct_coef[43],
dct_coef[44], dct_coef[45], dct_coef[46], dct_coef[47]};
    assign row6 = {dct_coef[48], dct_coef[49], dct_coef[50], dct_coef[51],
dct_coef[52], dct_coef[53], dct_coef[54], dct_coef[55]};
    assign row7 = {dct_coef[56], dct_coef[57], dct_coef[58], dct_coef[59],
dct_coef[60], dct_coef[61], dct_coef[62], dct_coef[63]};

    assign col0 = {dct_coef[0],  dct_coef[8],  dct_coef[16], dct_coef[24],
dct_coef[32], dct_coef[40], dct_coef[48], dct_coef[56]};
    assign col1 = {dct_coef[1],  dct_coef[9],  dct_coef[17], dct_coef[25],
dct_coef[33], dct_coef[41], dct_coef[49], dct_coef[57]};
    assign col2 = {dct_coef[2],  dct_coef[10], dct_coef[18], dct_coef[26],
dct_coef[34], dct_coef[42], dct_coef[50], dct_coef[58]};
    assign col3 = {dct_coef[3],  dct_coef[11], dct_coef[19], dct_coef[27],
dct_coef[35], dct_coef[43], dct_coef[51], dct_coef[59]};
    assign col4 = {dct_coef[4],  dct_coef[12], dct_coef[20], dct_coef[28],
dct_coef[36], dct_coef[44], dct_coef[52], dct_coef[60]};
    assign col5 = {dct_coef[5],  dct_coef[13], dct_coef[21], dct_coef[29],
dct_coef[37], dct_coef[45], dct_coef[53], dct_coef[61]};

```

```
    assign    col6 = {dct_coef[6], dct_coef[14], dct_coef[22], dct_coef[30],  
dct_coef[38], dct_coef[46], dct_coef[54], dct_coef[62]};  
    assign    col7 = {dct_coef[7], dct_coef[15], dct_coef[23], dct_coef[31],  
dct_coef[39], dct_coef[47], dct_coef[55], dct_coef[63]};
```

```
always @(posedge clk)
```

```
begin
```

```
    // Default DCT Coefficients
```

```
    dct_coef[0] = 91;  
    dct_coef[1] = 91;  
    dct_coef[2] = 91;  
    dct_coef[3] = 91;  
    dct_coef[4] = 91;  
    dct_coef[5] = 91;  
    dct_coef[6] = 91;  
    dct_coef[7] = 91;  
    dct_coef[8] = 126;  
    dct_coef[9] = 106;  
    dct_coef[10] = 71;  
    dct_coef[11] = 25;  
    dct_coef[12] = -25;  
    dct_coef[13] = -71;  
    dct_coef[14] = -106;  
    dct_coef[15] = -126;  
    dct_coef[16] = 118;  
    dct_coef[17] = 49;  
    dct_coef[18] = -49;  
    dct_coef[19] = -118;  
    dct_coef[20] = -118;  
    dct_coef[21] = -49;  
    dct_coef[22] = 49;  
    dct_coef[23] = 118;  
    dct_coef[24] = 106;  
    dct_coef[25] = -25;  
    dct_coef[26] = -126;  
    dct_coef[27] = -71;  
    dct_coef[28] = 71;  
    dct_coef[29] = 126;  
    dct_coef[30] = 25;  
    dct_coef[31] = -106;  
    dct_coef[32] = 91;  
    dct_coef[33] = -91;  
    dct_coef[34] = -91;  
    dct_coef[35] = 91;  
    dct_coef[36] = 91;  
    dct_coef[37] = -91;  
    dct_coef[38] = -91;  
    dct_coef[39] = 91;  
    dct_coef[40] = 71;  
    dct_coef[41] = -126;  
    dct_coef[42] = 25;  
    dct_coef[43] = 106;  
    dct_coef[44] = -106;  
    dct_coef[45] = -25;  
    dct_coef[46] = 126;  
    dct_coef[47] = -71;  
    dct_coef[48] = 49;
```



```
dct_coef[49] = -118;
dct_coef[50] = 118;
dct_coef[51] = -49;
dct_coef[52] = -49;
dct_coef[53] = 118;
dct_coef[54] = -118;
dct_coef[55] = 49;
dct_coef[56] = 25;
dct_coef[57] = -71;
dct_coef[58] = 106;
dct_coef[59] = -126;
dct_coef[60] = 126;
dct_coef[61] = -106;
dct_coef[62] = 71;
dct_coef[63] = -25;
end // always @ (posedge clk)

endmodule // dct_table
```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Encoder Memory Register
// 05/04/2006
//
////////////////////////////////////

module encode_memory_register(
    clk,
    reset,
    load_select,
    max_lines,
    max_blocks,
    resolution_select,
    coef0,
    coef1,
    coef2,
    coef3,
    coef4,
    coef5,
    coef6,
    coef7,
    data_output,
    address,
    wen
);

parameter          DATA_LOC = 15;
parameter          DATA_SIZE = 13;
parameter          DATA_BUFFER = 194;

parameter          NORMAL = 0;
parameter          ZOOM = 1;
parameter          OTHER = 2;

input clk;
input reset;
input [2:0] load_select, resolution_select;
input [4:0] max_lines, max_blocks;
input [DATA_SIZE-1:0] coef0, coef1, coef2, coef3, coef4, coef5, coef6, coef7;
output [DATA_BUFFER:0] data_output;
output [9:0] address;
output wen;

reg [DATA_BUFFER:0] data_output;
reg [3:0] buffer_fill;
reg [4:0] block_size;
reg [9:0] address;
reg wen;

always @(posedge clk)
    if (!reset)
        begin

```

```

    buffer_fill <= 0;
    wen <= 0;
    address <= 0;
    data_output <= 0;
end
else
begin

    // Pick WEN
    if (buffer_fill >= block_size)
        begin
            wen <= 1;
            buffer_fill <= load_select;
        end
    else
        begin
            wen <= 0;
            buffer_fill <= buffer_fill + load_select;
        end

        // Shift data_output to reflect
        case (load_select)
            1: data_output <= {data_output[DATA_BUFFER-1*DATA_SIZE:0], coef0};
            2: data_output <= {data_output[DATA_BUFFER-2*DATA_SIZE:0], coef0,
coef1};
            3: data_output <= {data_output[DATA_BUFFER-3*DATA_SIZE:0], coef0,
coef1, coef2};
            4: data_output <= {data_output[DATA_BUFFER-4*DATA_SIZE:0], coef0,
coef1, coef2, coef3};
            5: data_output <= {data_output[DATA_BUFFER-5*DATA_SIZE:0], coef0,
coef1, coef2, coef3, coef4};
            6: data_output <= {data_output[DATA_BUFFER-6*DATA_SIZE:0], coef0,
coef1, coef2, coef3, coef4, coef5};
            7: data_output <= {data_output[DATA_BUFFER-7*DATA_SIZE:0], coef0,
coef1, coef2, coef3, coef4, coef5, coef6};
            default: data_output <= data_output;
        endcase // case(load_select)

        // Increment
        if (wen)
            address <= (address == 899) ? 0 : address + 1;

        // Choose max_resolution
        case (resolution_select)
            NORMAL: block_size = 6;
            ZOOM: block_size = 15;
            OTHER: block_size = 10;
            default: block_size = 6;
        endcase // case(resolution_select)
    end // else: !if(!reset)

endmodule // encode_memory_register

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Encoder Finite State Machine
// 05/01/2006
//
////////////////////////////////////

module encode_fsm(
    clk,
    reset,
    resolution_select,
    max_blocks,
    max_lines,
    transmit_busy,
    line_active,
    line_read,
    block_read,
    column_select,
    inter_row_cnt,
    output_select,
    encode_busy,
    state);

    input clk, reset;
    input transmit_busy;
    input [2:0] resolution_select;
    input [4:0] max_blocks, max_lines, line_active;

    output [4:0] line_read, block_read;
    output [2:0] column_select;
    output [2:0] inter_row_cnt;
    output [2:0] output_select;
    output encode_busy;
    output [1:0] state;

    // Variables for Registered Outputs
    reg [4:0] line_read, block_read;
    reg [2:0] column_select;
    reg [2:0] inter_row_cnt;
    reg [2:0] output_select;
    reg encode_busy;

    reg [4:0] line_read_int, block_read_int;
    reg [2:0] column_select_int, inter_row_cnt_int;

    // State
    reg [1:0] state, next;

    parameter IDLE = 0;
    parameter IDLE_ENCODE = 1;
    parameter ENCODE_BLOCK = 2;
    integer i;
    parameter PIPELINE = 1;

```

```

parameter        NORMAL = 0;
parameter        ZOOM = 1;
parameter        OTHER = 2;

// Pipelined output selector
reg [2:0]        output_pipe[PIPELINE-1:0], output_pipe_int;

always @(posedge clk)
begin
    if (!reset)
        begin
            line_read <= 0;
            block_read <= 0;
            column_select <= 0;
            inter_row_cnt <= 0;
            output_select <= 0;
            state <= IDLE;

            for (i = 0; i < PIPELINE; i = i + 1)
                output_pipe[i] <= 0;

        end
    else
        begin
            line_read <= line_read_int;
            block_read <= block_read_int;
            column_select <= column_select_int;
            inter_row_cnt <= inter_row_cnt_int;
            output_select <= output_pipe[PIPELINE-1];
            state <= next;

            for (i = 0; i < PIPELINE-1; i = i + 1)
                output_pipe[i+1] <= output_pipe[i];

            output_pipe[0] <= output_pipe_int;

        end // else: !if(!reset)
end // always @ (posedge clk)

always @(line_read or block_read or line_active or column_select or state or
inter_row_cnt or resolution_select or transmit_busy)
begin

    line_read_int = line_read;
    block_read_int = block_read;
    column_select_int = column_select;
    inter_row_cnt_int = inter_row_cnt;
    next = state;

    encode_busy = 1;

    case (state)
        //
        // IDLE -> waiting for TRANSMITTER to FINISH SENDING FRAME
        //
        IDLE:

```

```

begin
    encode_busy = 0;
    output_pipe_int = 0;

    if (transmit_busy)
        next = IDLE;
    else
        next = IDLE_ENCODE;

end

//
// IDLE_ENCODE -> waiting for Video Capture to finish writing lines
//
IDLE_ENCODE:
begin
    // NOTICE -> not registered
    encode_busy = 1;
    output_pipe_int = 0;
    // Do nothing, wait for line_active to move past current_line
    if (line_read == line_active)
        next = IDLE_ENCODE;
    else
        next = ENCODE_BLOCK;
end

//
// Encode Block -> Multiply current block by DCT rows
//
ENCODE_BLOCK:
begin
    // Reset block_dct
    if (inter_row_cnt == 7)
        inter_row_cnt_int = 0;
    else
        inter_row_cnt_int = inter_row_cnt + 1;

    if ((inter_row_cnt == 7) && (column_select == 7))
        column_select_int = 0;
    else if (inter_row_cnt == 7)
        column_select_int = column_select + 1;

    // Reset Blocks
    if ((inter_row_cnt == 6) && (column_select == 7) && (block_read ==
max_blocks))
        block_read_int = 0;
    else if ((inter_row_cnt == 6) && (column_select == 7))
        block_read_int = block_read + 1;

    // Reset Lines
    if ((inter_row_cnt == 6) && (column_select == 7) && (block_read ==
max_blocks) && (line_read == max_lines))
        line_read_int = 0;
    else if ((inter_row_cnt == 6) && (column_select == 7) &&
(block_read == max_blocks))
        line_read_int = line_read + 1;

    // Set next state

```

```

        if ((inter_row_cnt == 7) && (column_select == 7) && (block_read ==
max_blocks) && (line_read == max_lines))
            next = IDLE;
        else if ((inter_row_cnt == 7) && (column_select == 7) &&
(block_read == max_blocks))
            begin
                // If next line to be read is active -> then transition into
IDLE_ENCODE state
                if (line_read_int == line_active)
                    next = IDLE_ENCODE;
                else
                    next = ENCODE_BLOCK;
            end

// Select output_pipe element
if (inter_row_cnt == 7)
case (resolution_select)
    NORMAL:
        case (column_select)
            0: output_pipe_int = 3;
            1: output_pipe_int = 2;
            2: output_pipe_int = 1;
            default: output_pipe_int = 0;
        endcase // case(column_select_int)
    ZOOM:
        case (column_select)
            0: output_pipe_int = 5;
            1: output_pipe_int = 4;
            2: output_pipe_int = 3;
            3: output_pipe_int = 2;
            4: output_pipe_int = 1;
            default: output_pipe_int = 0;
        endcase // case(column_select)
    OTHER:
        case (column_select)
            0: output_pipe_int = 4;
            1: output_pipe_int = 3;
            2: output_pipe_int = 2;
            3: output_pipe_int = 1;
            default: output_pipe_int = 0;
        endcase // case(column_select)
    default:
        case (column_select)
            0: output_pipe_int = 3;
            1: output_pipe_int = 2;
            2: output_pipe_int = 1;
            default: output_pipe_int = 0;
        endcase // case(column_select_int)
    endcase // case(resolution_select)
else
    output_pipe_int = 0;

end // case: ENCODE_BLOCK

endcase // case(state)

end // always @ (*)

```

```
endmodule // encode_fsm
```



```
////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Encoder - Wireless Memory Module
// 05/14/2006
//
////////////////////////////////////

module wireless_memory(
    wr_clk,
    wr_addr,
    din,
    wen,
    r_clk,
    read_addr,
    dout
);

    input wr_clk, r_clk;
    input [9:0] wr_addr, read_addr;
    input [77:0] din;
    input wen;

    output [77:0] dout;

    wireless_mem WIREMEM0(.clka(wr_clk), .clkb(r_clk), .wea(wen),
    .addra(wr_addr), .addrb(read_addr), .dina(din), .doutb(dout));

endmodule // wireless_memory
```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:      Noel Campbell
//
// Create Date:   23:00:12 05/14/06
// Design Name:
// Module Name:   tx_control_unit
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module tx_control_unit(clk,
                      reset,
                      cts_b,
                      read_data,
                      encode_wr_addr,
                      rts_b,
                      txd,
                      read_addr,
                      tx_busy,
                      rs232_state,
                      state,
                      shift_once,
                      send,
                      ready,
                      enable
                      );

input clk, reset, cts_b;
input [9:0] encode_wr_addr;
input [77:0] read_data;
output [9:0] read_addr;
output [2:0] state;
output [3:0] rs232_state;
output txd, rts_b, tx_busy;
output shift_once, send, ready, enable;

reg          shift_once, tx_busy, tx_busy_int, shift_once_int;
reg [2:0]    state, next;
reg [9:0]    read_addr_int, read_addr;
reg [3:0]    shift_cnt, shift_cnt_int;
reg         load, load_int;
reg         send, send_int;

wire [79:0]  encoded_data;
wire        ready;
wire [7:0]  encoded_byte;

```

```

parameter      MAX_READ_ADDR = 10'd899;

// States
parameter      IDLE = 0;
parameter      LOAD = 1;
parameter      SEND = 2;
parameter      WAIT = 3;
parameter      SHIFT = 4;

always @ (posedge clk) begin
    if (!reset) begin
        read_addr <= 10'd0;
        tx_busy <= 0;
        shift_once <= 0;
        state <= IDLE;
        send <= 0;
        load <= 0;
        shift_cnt <=0;
    end
    else begin
        read_addr <= read_addr_int;
        tx_busy <= tx_busy_int;
        shift_once <= shift_once_int;
        send <= send_int;
        load <= load_int;
        state <= next;
        shift_cnt <= shift_cnt_int;
    end
end

always @(state or encode_wr_addr or read_addr or ready or shift_cnt)
begin
    send_int = 0;
    load_int = 0;
    read_addr_int = read_addr;
    shift_once_int = 0;
    shift_cnt_int = shift_cnt;
    tx_busy_int = 1;
    case (state)
        IDLE:
            begin
                tx_busy_int = 0;
                if (read_addr != encode_wr_addr)
                    next = LOAD;
                else
                    next = IDLE;
            end
        LOAD:
            begin
                load_int = 1;
                if (ready)
                    next = SEND;
                else
                    next = LOAD;
            end
        SEND:
            begin

```

```

        send_int = 1;
        if (!ready)
            next = WAIT;
        else
            next = SEND;
    end
WAIT:
    if (!ready)
        next = WAIT;
    else
        next = SHIFT;
SHIFT:
    if (read_addr == 899)
        begin
            next = IDLE;
            shift_cnt_int = 0;
            read_addr_int = 0;
        end
    else if (shift_cnt == 9)
        begin
            shift_cnt_int = 0;
            next = LOAD;
            read_addr_int = read_addr + 1;
        end
    else
        begin
            shift_once_int = 1;
            shift_cnt_int = shift_cnt + 1;
            next = SEND;
        end // else: !if(shift_cnt == 9)
    endcase // case(state)
end // always @ (state or ...)

assign encoded_data = {2'b0, read_data};

tx_shift_reg txsr(.clk(clk),
    .reset(reset),
    .load(load),
    .encoded_data(encoded_data),
    .shift_once(shift_once),
    .encoded_byte(encoded_byte)
);

rs232_senderFSM_new sender(.clk(clk),
    .reset(reset),
    .data(encoded_byte),
    .cts_b(cts_b),
    .send(send),
    .txd(txd),
    .rts_b(rts_b),
    .ready(ready),
    .state(rs232_state),
    .enable(enable));
endmodule // tx_control_unit

```

```

`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
// Company:
// Engineer:      Noel Campbell
//
// Create Date:   23:00:12 05/14/06
// Design Name:
// Module Name:   tx_shift_reg
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////
module tx_shift_reg(clk,
                    reset,
                    encoded_data,
                    load,
                    shift_once,
                    encoded_byte);

    input clk, reset, shift_once;
    input [79:0] encoded_data;
    input  load;
    output [7:0] encoded_byte;

    reg [79:0]  load_data;

    assign  encoded_byte = load_data[7:0];

    always @ (posedge clk)
        begin
            if (!reset)
                load_data <= 0;
            else
                begin
                    // Load data into registers upon LOAD signal
                    if (load)
                        load_data <= encoded_data;

                    if (shift_once)
                        load_data <= {8'b0, load_data[79:8]};

                    end // else: !if(!reset)

                end // always @ (posedge clk)
        end

endmodule // tx_shift_reg

```

```

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    19:08:54 05/02/06
// Design Name:
// Module Name:    rs232_senderFSM_new
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module rs232_senderFSM_new(clk,
                           reset,
                           data,
                           cts_b,
                           send,
                           txd,
                           rts_b,
                           ready,
                           state,
                           enable);

    input clk, reset, cts_b, send;
    input[7:0] data;
    output rts_b, ready, txd;
    output[3:0] state;
    output enable;
    reg[3:0] state, next;
    reg[11:0] count;
    reg txd, txd_int, ready, enable, ready_int, rts_int, rts_b;

    parameter BAUD_COUNTER = 12'd107;
    parameter IDLE = 4'd0;
    parameter SEND_START_BIT = 4'd1;
    parameter SEND_DATA_BIT_0 = 4'd2;
    parameter SEND_DATA_BIT_1 = 4'd3;
    parameter SEND_DATA_BIT_2 = 4'd4;
    parameter SEND_DATA_BIT_3 = 4'd5;
    parameter SEND_DATA_BIT_4 = 4'd6;
    parameter SEND_DATA_BIT_5 = 4'd7;
    parameter SEND_DATA_BIT_6 = 4'd8;
    parameter SEND_DATA_BIT_7 = 4'd9;
    parameter SEND_STOP_BIT = 4'd10;

    always @ (posedge clk) begin
        if (!reset) begin
            count <= 12'd0;
            state <= SEND_STOP_BIT;

```

```

        rts_b <= 1;
        txd <= 1;
        ready <= 1;
    end
    else begin
        if (count==BAUD_COUNTER) begin
            enable <= 1;
            count <= 12'd0;
        end
        else begin
            count <= count + 12'd1;
            enable <= 0;
        end
        state <= next;
        txd <= txd_int;
        ready <= ready_int;
        rts_b <= rts_int;
    end
end

always @ (state or enable or send or ready) begin
    rts_int = 1;
    case (state)
        IDLE:
            begin
                txd_int = 1;
                ready_int = 1;
                rts_int = 0;
                if ((send == 1) && (enable == 1) && (cts_b == 0))
                    next = SEND_START_BIT;
                end
            else begin
                next = IDLE;
            end
        end
    SEND_START_BIT:
        begin
            txd_int = 0;
            ready_int = 0;
            if (enable == 1) begin
                next = SEND_DATA_BIT_0;
            end
            else next = SEND_START_BIT;
        end
    SEND_DATA_BIT_0:
        begin
            txd_int = data[0];
            ready_int = 0;
            if (enable == 1) begin
                next = SEND_DATA_BIT_1;
            end
            else next = SEND_DATA_BIT_0;
        end
    SEND_DATA_BIT_1:
        begin
            txd_int = data[1];

```

```

        ready_int = 0;
        if (enable == 1) begin
            next = SEND_DATA_BIT_2;
        end
        else next = SEND_DATA_BIT_1;
    end
SEND_DATA_BIT_2:
    begin
        txd_int = data[2];
        ready_int = 0;
        if (enable == 1) begin
            next = SEND_DATA_BIT_3;
        end
        else next = SEND_DATA_BIT_2;
    end
SEND_DATA_BIT_3:
    begin
        txd_int = data[3];
        ready_int = 0;
        if (enable == 1) begin
            next = SEND_DATA_BIT_4;
        end
        else next = SEND_DATA_BIT_3;
    end
SEND_DATA_BIT_4:
    begin
        txd_int = data[4];
        ready_int = 0;
        if (enable == 1) begin
            next = SEND_DATA_BIT_5;
        end
        else next = SEND_DATA_BIT_4;
    end
SEND_DATA_BIT_5:
    begin
        txd_int = data[5];
        ready_int = 0;
        if (enable == 1) begin
            next = SEND_DATA_BIT_6;
        end
        else next = SEND_DATA_BIT_5;
    end
SEND_DATA_BIT_6:
    begin
        txd_int = data[6];
        ready_int = 0;
        if (enable == 1) begin
            next = SEND_DATA_BIT_7;
        end
        else next = SEND_DATA_BIT_6;
    end
SEND_DATA_BIT_7:
    begin
        txd_int = data[7];
        ready_int = 0;
        if (enable == 1) begin
            next = SEND_STOP_BIT;
        end
    end

```



```
        end
        else next = SEND_DATA_BIT_7;
    end
    SEND_STOP_BIT:
    begin
        txd_int = 1;
        rts_int = 0;
        if ((enable == 1) && (cts_b == 0)) begin
            ready_int = 1;
            next = IDLE;
        end
        else begin
            ready_int = ready; //0;
            next = SEND_STOP_BIT;
        end
    end
end
endcase
end
endmodule
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    02:32:34 05/16/2006
// Design Name:    tx_control_unit
// Module Name:    tb_control_unit_new.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: tx_control_unit
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_control_unit_new_v;

    // Inputs
    reg clk;
    reg reset;
    reg cts_b;
    reg [77:0] read_data;
    reg [9:0]  encode_wr_addr;

    // Outputs
    wire      rts_b;
    wire      txd;
    wire [9:0] read_addr;
    wire      tx_busy;
    wire [3:0] rs232_state;
    wire [2:0] state;
    wire      shift_once;
    wire      send;
    wire      ready;
    wire      enable;

    // Instantiate the Unit Under Test (UUT)
    tx_control_unit uut (
        .clk(clk),
        .reset(reset),
        .cts_b(cts_b),
        .read_data(read_data),
        .encode_wr_addr(encode_wr_addr),
        .rts_b(rts_b),
        .txd(txd),
        .read_addr(read_addr),
        .tx_busy(tx_busy),
        .rs232_state(rs232_state),

```

```

        .state(state),
        .shift_once(shift_once),
        .send(send),
        .ready(ready),
        .enable(enable)
    );

defparam uut.sender.BAUD_COUNTER = 12'd2;
always #5 clk = ~clk;
initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    cts_b = 0;
    read_data = 0;
    encode_wr_addr = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    reset = 1;

    read_data = 78'hF0F0F0F0F0F0F0F0F0;

    repeat (30)
    begin
        #10 encode_wr_addr = encode_wr_addr + 1;
    end
end // initial begin
endmodule // tb_control_unit_new_v

```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:13:50 04/25/2006
// Design Name:    dct_back
// Module Name:    tb_dct_back.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: dct_back
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_dct_back_v;

    // Inputs
    reg clk;
    reg [71:0] row0;
    reg [71:0] row1;
    reg [71:0] row2;
    reg [71:0] row3;
    reg [71:0] row4;
    reg [71:0] row5;
    reg [71:0] row6;
    reg [71:0] row7;
    reg [143:0] column;

    // Outputs
    wire [95:0] output_column;
    wire [239:0] output_trunc;

    wire [71:0]    dct_row[7:0];
    wire [71:0]    dct_col[7:0];
    wire [11:0]    coef[7:0];
    wire [15:0]    col_frac[7:0];
    wire [15:0]    col_int[7:0];
    wire [29:0]    output_trunc_seg[7:0];
    integer i;

    // Instantiate DCT Table
    dct_table DCTTABLE0(.clk(clk),
                        .row0(dct_row[0]),
                        .row1(dct_row[1]),
                        .row2(dct_row[2]),
                        .row3(dct_row[3]),
                        .row4(dct_row[4]),

```

```

        .row5(dct_row[5]),
        .row6(dct_row[6]),
        .row7(dct_row[7]),
        .col0(dct_col[0]),
        .col1(dct_col[1]),
        .col2(dct_col[2]),
        .col3(dct_col[3]),
        .col4(dct_col[4]),
        .col5(dct_col[5]),
        .col6(dct_col[6]),
        .col7(dct_col[7])
    );

// Instantiate the Unit Under Test (UUT)
dct_back uut (
    .clk(clk),
    .row0(row0),
    .row1(row1),
    .row2(row2),
    .row3(row3),
    .row4(row4),
    .row5(row5),
    .row6(row6),
    .row7(row7),
    .column(column),
    .output_trunc(output_trunc),
    .output_column(output_column)
);

assign coef[0] = output_column[11:0];
assign coef[1] = output_column[23:12];
assign coef[2] = output_column[35:24];
assign coef[3] = output_column[47:36];
assign coef[4] = output_column[59:48];
assign coef[5] = output_column[71:60];
assign coef[6] = output_column[83:72];
assign coef[7] = output_column[95:84];

assign output_trunc_seg[0]= output_trunc[29:0];
assign output_trunc_seg[1]= output_trunc[59:30];
assign output_trunc_seg[2]= output_trunc[89:60];
assign output_trunc_seg[3]= output_trunc[119:90];
assign output_trunc_seg[4]= output_trunc[149:120];
assign output_trunc_seg[5]= output_trunc[179:150];
assign output_trunc_seg[6]= output_trunc[209:180];
assign output_trunc_seg[7]= output_trunc[239:210];

assign col_frac[0] = output_trunc[14:0];
assign col_int[0] = output_trunc[29:15];

assign col_frac[1] = output_trunc[44:30];
assign col_int[1] = output_trunc[59:45];

assign col_frac[2] = output_trunc[74:60];
assign col_int[2] = output_trunc[89:75];

assign col_frac[3] = output_trunc[104:90];

```

```

assign col_int[3] = output_trunc[119:105];

assign col_frac[4] = output_trunc[134:120];
assign col_int[4] = output_trunc[149:135];

assign col_frac[5] = output_trunc[164:150];
assign col_int[5] = output_trunc[179:165];

assign col_frac[6] = output_trunc[194:180];
assign col_int[6] = output_trunc[209:195];

assign col_frac[7] = output_trunc[224:210];
assign col_int[7] = output_trunc[239:225];

always #2 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    row0 = dct_row[0];
    row1 = dct_row[1];
    row2 = dct_row[2];
    row3 = dct_row[3];
    row4 = dct_row[4];
    row5 = dct_row[5];
    row6 = dct_row[6];
    row7 = dct_row[7];
    column = 0;

    // Wait 100 ns for global reset to finish
    #50;
    row0 = dct_row[0];
    row1 = dct_row[1];
    row2 = dct_row[2];
    row3 = dct_row[3];
    row4 = dct_row[4];
    row5 = dct_row[5];
    row6 = dct_row[6];
    row7 = dct_row[7];

    // Add stimulus here
    #4 column = {8{18'd256}};
    #4 column = {8{18'd128}};
    #4 column = {8{18'd256}};
    #4 column = {8{18'd512}};
    #4 column = {8{18'd728}};
    #4 column = {8{18'd128}};
    #4 column = {8{18'h3FFFF}};
    #4 column = {8{18'h20000}};

end

endmodule

```

```

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:23:16 04/25/2006
// Design Name:    dct_front
// Module Name:    tb_dct_front.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: dct_front
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//`include "dct_table.v"

module tb_dct_front_v;

    // Inputs
    reg clk;
    reg [71:0] row;
    reg [71:0] column;

    // Outputs
    wire [17:0] output_coef;
    wire [20:0] output_trunc;

    wire [71:0]    dct_row[7:0];
    wire [71:0]    dct_col[7:0];
    //reg [71:0]    row[7:0];
    wire [10:0]    col_int;
    wire [6:0]     col_frac;
    integer       i;

    // Instantiate DCT Table
    dct_table DCTTABLE0(.clk(clk),
                        .row0(dct_row[0]),
                        .row1(dct_row[1]),
                        .row2(dct_row[2]),
                        .row3(dct_row[3]),
                        .row4(dct_row[4]),
                        .row5(dct_row[5]),
                        .row6(dct_row[6]),
                        .row7(dct_row[7]),
                        .col0(dct_col[0]),
                        .col1(dct_col[1]),
                        .col2(dct_col[2]),

```

```

        .col3(dct_col[3]),
        .col4(dct_col[4]),
        .col5(dct_col[5]),
        .col6(dct_col[6]),
        .col7(dct_col[7])
    );

// Instantiate the Unit Under Test (UUT)
dct_front uut (
    .clk(clk),
    .row(row),
    .column(column),
    .output_trunc(output_trunc),
    .output_coef(output_coef)
);

assign    col_frac = output_coef[6:0];
assign    col_int  = output_coef[17:7];

always #20 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    row = 0;
    column = 0;

    // Wait 100 ns for global reset to finish
    #170;

    // Add stimulus here

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};
            column = dct_row[i];
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd100};
            column = dct_row[i];
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {9'd127, 9'd127, 9'd127, 9'd127, 9'd127, 9'd127, 9'd127,
9'd127};
            column = dct_row[i];
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255};
            column = dct_row[i];
        end
end

```



```
    for (i = 0; i < 8; i = i + 1)
    repeat (8) begin
        #40 row = {9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255};
        column = dct_row[i];
    end

end // initial begin

endmodule // tb_dct_front_v
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:49:17 04/25/2006
// Design Name:    dct_multiply
// Module Name:    tb_dct_multiply.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: dct_multiply
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_dct_multiply_v;

    // Inputs
    reg clk;
    reg reset;
    reg [71:0] row;
    reg [2:0] column_select;
    reg [8:0] temp;

    // Outputs
    wire [103:0] output_column;
    wire [17:0]  stagel_output;
    wire [143:0] shift_reg_output;
    wire        done;

    wire [12:0] col[7:0];
    wire [17:0] sreg_seg[7:0];

    assign      col[0] =  output_column[12:0];
    assign      col[1] =  output_column[25:13];
    assign      col[2] =  output_column[38:26];
    assign      col[3] =  output_column[51:39];
    assign      col[4] =  output_column[64:52];
    assign      col[5] =  output_column[77:65];
    assign      col[6] =  output_column[90:78];
    assign      col[7] =  output_column[103:91];

    assign      sreg_seg[0] =  shift_reg_output[17:0];
    assign      sreg_seg[1] =  shift_reg_output[35:18];
    assign      sreg_seg[2] =  shift_reg_output[53:36];
    assign      sreg_seg[3] =  shift_reg_output[71:54];
    assign      sreg_seg[4] =  shift_reg_output[89:72];
    assign      sreg_seg[5] =  shift_reg_output[107:90];

```

```

assign      sreg_seg[6] = shift_reg_output[125:108];
assign      sreg_seg[7] = shift_reg_output[143:126];

// Instantiate the Unit Under Test (UUT)
dct_multiply uut (
    .clk(clk),
    .reset(reset),
    .row(row),
    .column_select(column_select),
    .output_column(output_column),
    .stapel_output(stapel_output),
    .shift_reg_output(shift_reg_output),
    .done(done)
);

integer     i;

always #20 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    row = 1;
    column_select = 0;
    i = 0;

    // Wait 100 ns for global reset to finish
    #200 reset = 1;
    i = 8;
    #10;

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255};
            column_select = i;
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {9'd127, 9'd127, 9'd127, 9'd127, 9'd127, 9'd127, 9'd127,
9'd127};
            column_select = i;
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63};
            column_select = i;
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};
            column_select = i;
        end
end

```

```
for (i = 0; i < 8; i = i + 1)
repeat (8) begin
    #40 row = {9'd1, 9'd100, 9'd1, 9'd100, 9'd1, 9'd100, 9'd1, 9'd100};
    column_select = i;
end

for (i = 0; i < 8; i = i + 1)
repeat (8) begin
    #40 column_select = i;
    temp = 1 << i;
    row = temp << 9*i;
    column_select = i;
end

end // initial begin

endmodule // tb_dct_multiply_v
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:09:28 04/24/2006
// Design Name:    dct_table
// Module Name:    tb_dct_table.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: dct_table
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_dct_table_v;

    // Inputs
    reg clk;

    // Outputs
    wire [71:0] row0;
    wire [71:0] row1;
    wire [71:0] row2;
    wire [71:0] row3;
    wire [71:0] row4;
    wire [71:0] row5;
    wire [71:0] row6;
    wire [71:0] row7;
    wire [71:0] col0;
    wire [71:0] col1;
    wire [71:0] col2;
    wire [71:0] col3;
    wire [71:0] col4;
    wire [71:0] col5;
    wire [71:0] col6;
    wire [71:0] col7;

    wire [8:0] row_0[7:0];
    wire [8:0] row_1[7:0];
    wire [8:0] row_2[7:0];
    wire [8:0] row_3[7:0];
    wire [8:0] row_4[7:0];
    wire [8:0] row_5[7:0];
    wire [8:0] row_6[7:0];
    wire [8:0] row_7[7:0];

    assign row_0[0] = row0[8:0];

```

```
assign    row_0[1] = row0[17:9];
assign    row_0[2] = row0[26:18];
assign    row_0[3] = row0[35:27];
assign    row_0[4] = row0[44:36];
assign    row_0[5] = row0[53:45];
assign    row_0[6] = row0[62:54];
assign    row_0[7] = row0[71:63];
```

```
assign    row_1[0] = row1[8:0];
assign    row_1[1] = row1[17:9];
assign    row_1[2] = row1[26:18];
assign    row_1[3] = row1[35:27];
assign    row_1[4] = row1[44:36];
assign    row_1[5] = row1[53:45];
assign    row_1[6] = row1[62:54];
assign    row_1[7] = row1[71:63];
```

```
assign    row_2[0] = row2[8:0];
assign    row_2[1] = row2[17:9];
assign    row_2[2] = row2[26:18];
assign    row_2[3] = row2[35:27];
assign    row_2[4] = row2[44:36];
assign    row_2[5] = row2[53:45];
assign    row_2[6] = row2[62:54];
assign    row_2[7] = row2[71:63];
```

```
assign    row_3[0] = row3[8:0];
assign    row_3[1] = row3[17:9];
assign    row_3[2] = row3[26:18];
assign    row_3[3] = row3[35:27];
assign    row_3[4] = row3[44:36];
assign    row_3[5] = row3[53:45];
assign    row_3[6] = row3[62:54];
assign    row_3[7] = row3[71:63];
```

```
assign    row_4[0] = row4[8:0];
assign    row_4[1] = row4[17:9];
assign    row_4[2] = row4[26:18];
assign    row_4[3] = row4[35:27];
assign    row_4[4] = row4[44:36];
assign    row_4[5] = row4[53:45];
assign    row_4[6] = row4[62:54];
assign    row_4[7] = row4[71:63];
```

```
assign    row_5[0] = row5[8:0];
assign    row_5[1] = row5[17:9];
assign    row_5[2] = row5[26:18];
assign    row_5[3] = row5[35:27];
assign    row_5[4] = row5[44:36];
assign    row_5[5] = row5[53:45];
assign    row_5[6] = row5[62:54];
assign    row_5[7] = row5[71:63];
```

```
assign    row_6[0] = row6[8:0];
assign    row_6[1] = row6[17:9];
assign    row_6[2] = row6[26:18];
assign    row_6[3] = row6[35:27];
```

```
assign    row_6[4] = row6[44:36];
assign    row_6[5] = row6[53:45];
assign    row_6[6] = row6[62:54];
assign    row_6[7] = row6[71:63];
```

```
assign    row_7[0] = row7[8:0];
assign    row_7[1] = row7[17:9];
assign    row_7[2] = row7[26:18];
assign    row_7[3] = row7[35:27];
assign    row_7[4] = row7[44:36];
assign    row_7[5] = row7[53:45];
assign    row_7[6] = row7[62:54];
assign    row_7[7] = row7[71:63];
```

```
wire [8:0] col_0[7:0];
wire [8:0] col_1[7:0];
wire [8:0] col_2[7:0];
wire [8:0] col_3[7:0];
wire [8:0] col_4[7:0];
wire [8:0] col_5[7:0];
wire [8:0] col_6[7:0];
wire [8:0] col_7[7:0];
```

```
assign    col_0[0] = col0[8:0];
assign    col_0[1] = col0[17:9];
assign    col_0[2] = col0[26:18];
assign    col_0[3] = col0[35:27];
assign    col_0[4] = col0[44:36];
assign    col_0[5] = col0[53:45];
assign    col_0[6] = col0[62:54];
assign    col_0[7] = col0[71:63];
```

```
assign    col_1[0] = col1[8:0];
assign    col_1[1] = col1[17:9];
assign    col_1[2] = col1[26:18];
assign    col_1[3] = col1[35:27];
assign    col_1[4] = col1[44:36];
assign    col_1[5] = col1[53:45];
assign    col_1[6] = col1[62:54];
assign    col_1[7] = col1[71:63];
```

```
assign    col_2[0] = col2[8:0];
assign    col_2[1] = col2[17:9];
assign    col_2[2] = col2[26:18];
assign    col_2[3] = col2[35:27];
assign    col_2[4] = col2[44:36];
assign    col_2[5] = col2[53:45];
assign    col_2[6] = col2[62:54];
assign    col_2[7] = col2[71:63];
```

```
assign    col_3[0] = col3[8:0];
assign    col_3[1] = col3[17:9];
assign    col_3[2] = col3[26:18];
assign    col_3[3] = col3[35:27];
assign    col_3[4] = col3[44:36];
assign    col_3[5] = col3[53:45];
assign    col_3[6] = col3[62:54];
```

```

assign    col_3[7] = col3[71:63];

assign    col_4[0] = col4[8:0];
assign    col_4[1] = col4[17:9];
assign    col_4[2] = col4[26:18];
assign    col_4[3] = col4[35:27];
assign    col_4[4] = col4[44:36];
assign    col_4[5] = col4[53:45];
assign    col_4[6] = col4[62:54];
assign    col_4[7] = col4[71:63];

assign    col_5[0] = col5[8:0];
assign    col_5[1] = col5[17:9];
assign    col_5[2] = col5[26:18];
assign    col_5[3] = col5[35:27];
assign    col_5[4] = col5[44:36];
assign    col_5[5] = col5[53:45];
assign    col_5[6] = col5[62:54];
assign    col_5[7] = col5[71:63];

assign    col_6[0] = col6[8:0];
assign    col_6[1] = col6[17:9];
assign    col_6[2] = col6[26:18];
assign    col_6[3] = col6[35:27];
assign    col_6[4] = col6[44:36];
assign    col_6[5] = col6[53:45];
assign    col_6[6] = col6[62:54];
assign    col_6[7] = col6[71:63];

assign    col_7[0] = col7[8:0];
assign    col_7[1] = col7[17:9];
assign    col_7[2] = col7[26:18];
assign    col_7[3] = col7[35:27];
assign    col_7[4] = col7[44:36];
assign    col_7[5] = col7[53:45];
assign    col_7[6] = col7[62:54];
assign    col_7[7] = col7[71:63];

// Instantiate the Unit Under Test (UUT)
dct_table uut (
    .clk(clk),
    .row0(row0),
    .row1(row1),
    .row2(row2),
    .row3(row3),
    .row4(row4),
    .row5(row5),
    .row6(row6),
    .row7(row7),
    .col0(col0),
    .col1(col1),
    .col2(col2),
    .col3(col3),
    .col4(col4),
    .col5(col5),
    .col6(col6),
    .col7(col7)

```



```
        );  
  
always #1 clk = ~clk;  
  
initial begin  
    // Initialize Inputs  
    clk = 0;  
  
    // Wait 100 ns for global reset to finish  
    #5;  
  
    // Add stimulus here  
  
end  
  
endmodule
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:07:37 05/04/2006
// Design Name:    encode_fsm
// Module Name:    tb_encode_fsm.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: encode_fsm
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_encode_fsm_v;

    // Inputs
    reg clk;
    reg reset;
    reg transmit_busy;
    reg [2:0] resolution_select;
    reg [4:0] max_blocks;
    reg [4:0] max_lines;
    reg [4:0] line_active;

    // Outputs
    wire [4:0] line_read;
    wire [4:0] block_read;
    wire [2:0] column_select;
    wire [2:0] inter_row_cnt;
    wire [2:0] output_select;
    wire      encode_busy;
    wire [1:0] state;

    // Instantiate the Unit Under Test (UUT)
    encode_fsm uut (
        .clk(clk),
        .reset(reset),
        .resolution_select(resolution_select),
        .max_blocks(max_blocks),
        .max_lines(max_lines),
        .transmit_busy(transmit_busy),
        .line_active(line_active),
        .line_read(line_read),
        .block_read(block_read),
        .column_select(column_select),
        .inter_row_cnt(inter_row_cnt),
    );

```

```

        .output_select(output_select),
        .encode_busy(encode_busy),
        .state(state)
    );

always #0.25 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    transmit_busy = 0;
    resolution_select = 0;
    max_blocks = 10;
    max_lines = 4;
    line_active = 1;

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

    #800 line_active = 2;

    repeat (15)
        #20 line_active = line_active + 1;

    #5 reset = 0;
    resolution_select = 0;
    #10 reset = 1;

    #20 transmit_busy = 1;

    #2000 transmit_busy = 0;

    // Add stimulus here

end // initial begin

endmodule // tb_encode_fsm_v

```

```

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:44:24 05/04/2006
// Design Name:    encode_memory_register
// Module Name:    tb_encode_mem_reg.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: encode_memory_register
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

module tb_encode_mem_reg_v;

    // Inputs
    reg clk;
    reg reset;
    reg [2:0] load_select;
    reg [4:0] max_lines;
    reg [4:0] max_blocks;
    reg [2:0] resolution_select;
    reg [12:0] coef0;
    reg [12:0] coef1;
    reg [12:0] coef2;
    reg [12:0] coef3;
    reg [12:0] coef4;
    reg [12:0] coef5;
    reg [12:0] coef6;
    reg [12:0] coef7;

    // Outputs
    wire [194:0] data_output;
    wire [9:0]    address;
    wire        wen;

    wire [12:0]    coef[7:0];

    // Instantiate the Unit Under Test (UUT)
    encode_memory_register uut (
        .clk(clk),
        .reset(reset),
        .load_select(load_select),
        .max_lines(max_lines),
        .max_blocks(max_blocks),
        .resolution_select(resolution_select),

```

```

        .coef0(coef0),
        .coef1(coef1),
        .coef2(coef2),
        .coef3(coef3),
        .coef4(coef4),
        .coef5(coef5),
        .coef6(coef6),
        .coef7(coef7),
        .data_output(data_output),
        .address(address),
        .wen(wen)
    );

always #1 clk = ~clk;

assign  coef[0] = data_output[12:0];
assign  coef[1] = data_output[25:13];
assign  coef[2] = data_output[38:26];
assign  coef[3] = data_output[51:39];
assign  coef[4] = data_output[64:52];
assign  coef[5] = data_output[77:65];
assign  coef[6] = data_output[90:78];
assign  coef[7] = data_output[103:91];

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    load_select = 0;
    max_lines = 3;
    max_blocks = 10;
    resolution_select = 0;
    coef0 = 1;
    coef1 = 2;
    coef2 = 3;
    coef3 = 4;
    coef4 = 5;
    coef5 = 6;
    coef6 = 7;
    coef7 = 8;

    // Wait 100 ns for global reset to finish
    #15 reset = 1;

    #2 load_select = 1;
    #2 load_select = 2;
    #2 load_select = 3;
    #2 load_select = 0;
    #4 load_select = 1;
    #2 load_select = 2;
    #2 load_select = 3;
    #2 load_select = 1;
    #2 load_select = 2;
    #2 load_select = 3;
    #2 load_select = 0;
    #8 load_select = 1;
    #2 load_select = 2;

```

```
#2 load_select = 0;  
// Add stimulus here  
end  
endmodule
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:43:25 05/05/2006
// Design Name:    encoder
// Module Name:    tb_encoder.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: encoder
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_encoder_v;

    // Inputs
    reg clk;
    reg reset;
    reg [2:0] resolution_select;
    reg [4:0] max_lines;
    reg [4:0] max_blocks;
    reg [4:0] line_active;
    reg [71:0] row;
    reg          transmit_busy;

    // Outputs
    wire          encode_busy;
    wire [2:0] inter_row_cnt;
    wire [4:0] block_read;
    wire [4:0] line_read;
    wire [194:0] data_output;
    wire [9:0]   write_address;
    wire          wen;
    wire [103:0] mult_column;
    wire [12:0]   mult[7:0];
    wire [12:0]   output_seg[7:0];
    wire [2:0]   column_select;
    wire [2:0]   output_select;
    wire [17:0]  stage1_output;
    wire [143:0] shift_reg_output;
    wire [17:0]  sreg_seg[7:0];

    assign mult[0] = mult_column[12:0];
    assign mult[1] = mult_column[25:13];
    assign mult[2] = mult_column[38:26];
    assign mult[3] = mult_column[51:39];

```

```

assign mult[4] = mult_column[64:52];
assign mult[5] = mult_column[77:65];
assign mult[6] = mult_column[90:78];
assign mult[7] = mult_column[103:91];

assign output_seg[0] = data_output[12:0];
assign output_seg[1] = data_output[25:13];
assign output_seg[2] = data_output[38:26];
assign output_seg[3] = data_output[51:39];
assign output_seg[4] = data_output[64:52];
assign output_seg[5] = data_output[77:65];
assign output_seg[6] = data_output[90:78];
assign output_seg[7] = data_output[103:91];

assign sreg_seg[0] = shift_reg_output[17:0];
assign sreg_seg[1] = shift_reg_output[35:18];
assign sreg_seg[2] = shift_reg_output[53:36];
assign sreg_seg[3] = shift_reg_output[71:54];
assign sreg_seg[4] = shift_reg_output[89:72];
assign sreg_seg[5] = shift_reg_output[107:90];
assign sreg_seg[6] = shift_reg_output[125:108];
assign sreg_seg[7] = shift_reg_output[143:126];

// Instantiate the Unit Under Test (UUT)
encoder uut (
    .clk(clk),
    .reset(reset),
    .resolution_select(resolution_select),
    .max_lines(max_lines),
    .max_blocks(max_blocks),
    .line_active(line_active),
    .row(row),
    .encode_busy(encode_busy),
    .transmit_busy(transmit_busy),
    .inter_row_cnt(inter_row_cnt),
    .block_read(block_read),
    .line_read(line_read),
    .data_output(data_output),
    .write_address(write_address),
    .wen(wen),
    .mult_column(mult_column),
    .column_select(column_select),
    .output_select(output_select),
    .stagel_output(stagel_output),
    .shift_reg_output(shift_reg_output)
);

always #20 clk = ~clk;

reg [8:0] temp;
integer i;

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;

```



```

resolution_select = 0;
max_lines = 4;
max_blocks = 10;
line_active = 5;
row = 0;
transmit_busy = 0;

// Wait 100 ns for global reset to finish
#200 reset = 1;
#10;

// Add stimulus here
for (i = 0; i < 8; i = i + 1)
repeat (8) begin
    #40 row = {9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255, 9'd255,
9'd255};

    end

    for (i = 0; i < 8; i = i + 1)
repeat (8) begin
    #40 row = {9'd127, 9'd127, 9'd127, 9'd127, 9'd127, 9'd127, 9'd127,
9'd127};

    end

    for (i = 0; i < 8; i = i + 1)
repeat (8) begin
    #40 row = {9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63, 9'd63};

    end

    for (i = 0; i < 8; i = i + 1)
repeat (8) begin
    #40 row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};

    end

    for (i = 0; i < 8; i = i + 1)
repeat (8) begin
    #40 row = {9'd1, 9'd100, 9'd1, 9'd100, 9'd1, 9'd100, 9'd1, 9'd100};

    end

    for (i = 0; i < 8; i = i + 1)
repeat (8) begin
    #40
        temp = 1 << i;
        row = temp << 9*i;

    end

    #2000 transmit_busy = 1;

end // initial begin
endmodule // tb_encoder_v

```



```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:06:50 04/23/2006
// Design Name:    matrix_naive
// Module Name:    tb_mtx.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: matrix_naive
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_mtx_v;

    // Inputs
    reg clk;
    reg [71:0] row;
    reg [71:0] column;

    // Outputs
    wire [20:0] product;

    wire    signed [8:0] row_pixel [7:0];
    wire    signed [8:0] col_pixel [7:0];
    wire    signed [17:0] RX[7:0];
    wire    signed [18:0] RI[3:0];
    wire    signed [19:0] RM[1:0];

    assign    row_pixel[0] = row[8:0];
    assign    row_pixel[1] = row[17:9];
    assign    row_pixel[2] = row[26:18];
    assign    row_pixel[3] = row[35:27];
    assign    row_pixel[4] = row[44:36];
    assign    row_pixel[5] = row[53:45];
    assign    row_pixel[6] = row[62:54];
    assign    row_pixel[7] = row[71:63];

    assign    col_pixel[0] = column[8:0];
    assign    col_pixel[1] = column[17:9];
    assign    col_pixel[2] = column[26:18];
    assign    col_pixel[3] = column[35:27];
    assign    col_pixel[4] = column[44:36];
    assign    col_pixel[5] = column[53:45];
    assign    col_pixel[6] = column[62:54];
    assign    col_pixel[7] = column[71:63];

```

```

// Instantiate the Unit Under Test (UUT)
matrix_naive uut (
    .clk(clk),
    .row(row),
    .column(column),
    .RX_0(RX[0]), .RX_1(RX[1]), .RX_2(RX[2]), .RX_3(RX[3]),
    .RX_4(RX[4]), .RX_5(RX[5]), .RX_6(RX[6]), .RX_7(RX[7]),
    .RI_0(RI[0]), .RI_1(RI[1]), .RI_2(RI[2]), .RI_3(RI[3]),
    .RM_0(RM[0]), .RM_1(RM[1]),
    .product(product)
);

always #2 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    row = 0;
    column = 0;

    #5;

    // Add stimulus here
    #4 row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};
    column = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};

    #4 row = {9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2};
    column = {9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2};

    #4 row = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};
    column = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};

    #4 row = {9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF};
    column = {9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF};

    #4 row = {9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0};
    column = {9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0};

    #4 row = {9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF};
    column = {9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0, 9'hF0};

    #4 row = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};
    column = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};

    #4 row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};
    column = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};

    #4 row = {9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0};
    column = {9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0};

end

endmodule

```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:52:18 04/24/2006
// Design Name:    matrix_mult_s1
// Module Name:    tb_mtx_mult_s1.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: matrix_mult_s1
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_mtx_mult_s1_v;

    // Inputs
    reg clk;
    reg [71:0] row0;
    reg [71:0] row1;
    reg [71:0] row2;
    reg [71:0] row3;
    reg [71:0] row4;
    reg [71:0] row5;
    reg [71:0] row6;
    reg [71:0] row7;
    reg [71:0] column;

    // Outputs
    wire [20:0] coef0;
    wire [20:0] coef1;
    wire [20:0] coef2;
    wire [20:0] coef3;
    wire [20:0] coef4;
    wire [20:0] coef5;
    wire [20:0] coef6;
    wire [20:0] coef7;

    // Instantiate the Unit Under Test (UUT)
    matrix_mult_s1 uut (
        .clk(clk),
        .row0(row0),
        .row1(row1),
        .row2(row2),
        .row3(row3),
        .row4(row4),
        .row5(row5),

```

```

        .row6(row6),
        .row7(row7),
        .column(column),
        .coef0(coef0),
        .coef1(coef1),
        .coef2(coef2),
        .coef3(coef3),
        .coef4(coef4),
        .coef5(coef5),
        .coef6(coef6),
        .coef7(coef7)
    );

always #2 clk = ~clk;

reg [71:0] row[7:0];
integer i;

initial begin
    // Initialize Inputs
    clk = 0;
    row0 = 0;
    row1 = 0;
    row2 = 0;
    row3 = 0;
    row4 = 0;
    row5 = 0;
    row6 = 0;
    row7 = 0;
    column = 0;

    for (i = 0; i < 8; i = i +1)
        row[i] = 1 << 9*i;

    // Wait 100 ns for global reset to finish
    #5;

    // Add stimulus here

    // 1 * 1 in all rows (shifted to represent different pixels)
    #4 row0 = row[0];
    row1 = row[1];
    row2 = row[2];
    row3 = row[3];
    row4 = row[4];
    row5 = row[5];
    row6 = row[6];
    row7 = row[7];
    column = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};

    // 1 * 2 in all rows (shifted to represent different pixels)
    for (i = 0; i < 8; i = i +1)
        row[i] = 2'b10 << 9*i;

    #4 row0 = row[0];
    row1 = row[1];

```

```

row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};

// (2*i) * 1 ( should show 0-16)
for (i = 0; i < 8; i = i +1)
row[i] = 2*i;

#4 row0 = row[0];
row1 = row[1];
row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};

// (2*i) * -1 (should be 0 - (-16))
#4 row0 = row[0];
row1 = row[1];
row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF};

// 255 * 255 * 8 or (520200) in all rows (most positive number)
for (i = 0; i < 8; i = i +1)
row[i] = {9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF};

#4 row0 = row[0];
row1 = row[1];
row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF};

// -256 * -256 * 8 or (524288) in all rows (most negative numbers)
for (i = 0; i < 8; i = i +1)
row[i] = {9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100};

#4 row0 = row[0];
row1 = row[1];
row2 = row[2];

```

```

row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100};

// -256 * 255 * 8 or (522240) in all rows (most positive * most negative)
for (i = 0; i < 8; i = i +1)
row[i] = {9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100};

#4 row0 = row[0];
row1 = row[1];
row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF};

// 1, 4, 9, 16 ...
for (i = 0; i < 8; i = i +1)
row[i] = i + 1 << 9*(7-i);

#4 row0 = row[0];
row1 = row[1];
row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};

end

endmodule

```



```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:19:54 04/24/2006
// Design Name:    matrix_mult_s2
// Module Name:    tb_mtx_mult_s2.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: matrix_mult_s2
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_mtx_mult_s2_v;

    // Inputs
    reg clk;
    reg [71:0] row0;
    reg [71:0] row1;
    reg [71:0] row2;
    reg [71:0] row3;
    reg [71:0] row4;
    reg [71:0] row5;
    reg [71:0] row6;
    reg [71:0] row7;
    reg [143:0] column;

    // Outputs
    wire [29:0] coef0;
    wire [29:0] coef1;
    wire [29:0] coef2;
    wire [29:0] coef3;
    wire [29:0] coef4;
    wire [29:0] coef5;
    wire [29:0] coef6;
    wire [29:0] coef7;

    // Instantiate the Unit Under Test (UUT)
    matrix_mult_s2 uut (
        .clk(clk),
        .row0(row0),
        .row1(row1),
        .row2(row2),
        .row3(row3),
        .row4(row4),
        .row5(row5),

```

```

        .row6(row6),
        .row7(row7),
        .column(column),
        .coef0(coef0),
        .coef1(coef1),
        .coef2(coef2),
        .coef3(coef3),
        .coef4(coef4),
        .coef5(coef5),
        .coef6(coef6),
        .coef7(coef7)
    );
always #2 clk = ~clk;

reg [71:0] row[7:0];
integer i;

initial begin
    // Initialize Inputs
    clk = 0;
    row0 = 0;
    row1 = 0;
    row2 = 0;
    row3 = 0;
    row4 = 0;
    row5 = 0;
    row6 = 0;
    row7 = 0;
    column = 0;

    for (i = 0; i < 8; i = i +1)
        row[i] = 1 << 9*i;

    // Wait 100 ns for global reset to finish
    #5;

    // Add stimulus here

    // 1 * 1 in all rows (shifted to represent different pixels)
    #4 row0 = row[0];
    row1 = row[1];
    row2 = row[2];
    row3 = row[3];
    row4 = row[4];
    row5 = row[5];
    row6 = row[6];
    row7 = row[7];
    column = {18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1};

    // 1 * 2 in all rows (shifted to represent different pixels)
    for (i = 0; i < 8; i = i +1)
        row[i] = 2'b10 << 9*i;

    #4 row0 = row[0];
    row1 = row[1];
    row2 = row[2];

```

```

row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1};

// (2*i) * 1 ( should show 0-16)
for (i = 0; i < 8; i = i +1)
row[i] = 2*i;

#4 row0 = row[0];
row1 = row[1];
row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1};

// (2*i) * -1 (should be 0 - (-16))
#4 row0 = row[0];
row1 = row[1];
row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {18'h3FFFF, 18'h3FFFF, 18'h3FFFF, 18'h3FFFF, 18'h3FFFF,
18'h3FFFF, 18'h3FFFF, 18'h3FFFF};

// 255 * 131071 * 8 or (33423105) in all rows (most positive number)
for (i = 0; i < 8; i = i +1)
row[i] = {9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF};

#4 row0 = row[0];
row1 = row[1];
row2 = row[2];
row3 = row[3];
row4 = row[4];
row5 = row[5];
row6 = row[6];
row7 = row[7];
column = {18'h1FFFF, 18'h1FFFF, 18'h1FFFF, 18'h1FFFF, 18'h1FFFF,
18'h1FFFF, 18'h1FFFF, 18'h1FFFF};

// -256 * -131072 * 8 or (33554432) in all rows (most negative numbers)
for (i = 0; i < 8; i = i +1)
row[i] = {9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100};

#4 row0 = row[0];

```

```

    row1 = row[1];
    row2 = row[2];
    row3 = row[3];
    row4 = row[4];
    row5 = row[5];
    row6 = row[6];
    row7 = row[7];
    column = {18'h20000, 18'h20000, 18'h20000, 18'h20000, 18'h20000,
18'h20000, 18'h20000, 18'h20000};

    // -256 * 131071 * 8 or (33554176) in all rows (most positive * most
negative)
    for (i = 0; i < 8; i = i +1)
        row[i] = {9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100};

    #4 row0 = row[0];
    row1 = row[1];
    row2 = row[2];
    row3 = row[3];
    row4 = row[4];
    row5 = row[5];
    row6 = row[6];
    row7 = row[7];
    column = {18'h1FFFF, 18'h1FFFF, 18'h1FFFF, 18'h1FFFF, 18'h1FFFF,
18'h1FFFF, 18'h1FFFF, 18'h1FFFF};

    // 1, 4, 9, 16 ...
    for (i = 0; i < 8; i = i +1)
        row[i] = i + 1 << 9*(7-i);

    #4 row0 = row[0];
    row1 = row[1];
    row2 = row[2];
    row3 = row[3];
    row4 = row[4];
    row5 = row[5];
    row6 = row[6];
    row7 = row[7];
    column = {18'd1, 18'd2, 18'd3, 18'd4, 18'd5, 18'd6, 18'd7, 18'd8};

end

endmodule

```

```

// Verilog Test Fixture Template

`timescale 1 ns / 1 ps

module TEST_gate;
    reg <signal1>;
    reg [2:0] <signal2>;
    wire [3:0] <signal3>;
    wire <signal4>;

    <module_name> <instance_name> (
        <port1>,
        <port2>
    );

    integer <name1>;
    integer <name2>;

    // The following code initializes the Global Set Reset (GSR) and Global
    // Three-State (GTS) nets
    // Refer to the Synthesis and Simulation Design Guide for more information on
    // this process
    reg GSR;
    assign glbl.GSR = GSR;
    reg GTS;
    assign glbl.GTS = GTS;

    initial begin
        GSR = 1;
        GTS = 0; // GTS is not activated by default
        #100; // GSR is set for 100 ns
        GSR = 0;
    end

    // Initialize Inputs
    `ifdef auto_init
        initial begin
            end

        `endif
endmodule

```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:09:49 04/23/2006
// Design Name:    matrix_naive_s2
// Module Name:    tb_mtx_s2.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: matrix_naive_s2
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_mtx_s2_v;

    // Inputs
    reg clk;
    reg [167:0] row;
    reg [167:0] column;

    // Outputs
    wire [41:0] RX_0;
    wire [41:0] RX_1;
    wire [41:0] RX_2;
    wire [41:0] RX_3;
    wire [41:0] RX_4;
    wire [41:0] RX_5;
    wire [41:0] RX_6;
    wire [41:0] RX_7;
    wire [42:0] RI_0;
    wire [42:0] RI_1;
    wire [42:0] RI_2;
    wire [42:0] RI_3;
    wire [43:0] RM_0;
    wire [43:0] RM_1;
    wire [44:0] product;

    // Instantiate the Unit Under Test (UUT)
    matrix_naive_s2 uut (
        .clk(clk),
        .row(row),
        .column(column),
        .RX_0(RX_0),
        .RX_1(RX_1),
        .RX_2(RX_2),
        .RX_3(RX_3),

```

```

        .RX_4(RX_4),
        .RX_5(RX_5),
        .RX_6(RX_6),
        .RX_7(RX_7),
        .RI_0(RI_0),
        .RI_1(RI_1),
        .RI_2(RI_2),
        .RI_3(RI_3),
        .RM_0(RM_0),
        .RM_1(RM_1),
        .product(product)
    );

always #20 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    row = 0;
    column = 0;
    #100;

    // Add stimulus here
    #80 row = {21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1};
    column = {21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1};

    #80 row = {21'd2, 21'd2, 21'd2, 21'd2, 21'd2, 21'd2, 21'd2, 21'd2};
    column = {21'd2, 21'd2, 21'd2, 21'd2, 21'd2, 21'd2, 21'd2, 21'd2};

    #80 row = {21'd1, 21'd2, 21'd3, 21'd4, 21'd5, 21'd6, 21'd7, 21'd8};
    column = {21'd1, 21'd2, 21'd3, 21'd4, 21'd5, 21'd6, 21'd7, 21'd8};

    #80 row = {21'hFFFF, 21'hFFFF, 21'hFFFF, 21'hFFFF, 21'hFFFF, 21'hFFFF,
21'hFFFF, 21'hFFFF};
    column = {21'hFFFF, 21'hFFFF, 21'hFFFF, 21'hFFFF, 21'hFFFF, 21'hFFFF,
21'hFFFF, 21'hFFFF};

    #80 row = {21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0,
21'hFFF0, 21'hFFF0};
    column = {21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0,
21'hFFF0, 21'hFFF0};

    #80 row = {21'h1FFFF, 21'h1FFFF, 21'h1FFFF, 21'h1FFFF, 21'h1FFFF,
21'h1FFFF, 21'h1FFFF, 21'h1FFFF};
    column = {21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0, 21'hFFF0,
21'hFFF0, 21'hFFF0};

    #80 row = {21'd1, 21'd2, 21'd3, 21'd4, 21'd5, 21'd6, 21'd7, 21'd8};
    column = {21'd1, 21'd2, 21'd3, 21'd4, 21'd5, 21'd6, 21'd7, 21'd8};

    #80 row = {21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1};
    column = {21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1, 21'd1};

    #80 row = {21'd0, 21'd0, 21'd0, 21'd0, 21'd0, 21'd0, 21'd0, 21'd0};
    column = {21'd0, 21'd0, 21'd0, 21'd0, 21'd0, 21'd0, 21'd0, 21'd0};

end

```

endmodule


```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:02:14 04/23/2006
// Design Name:    matrix_naive_s2_trunc
// Module Name:    tb_mtx_s2_trunc.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: matrix_naive_s2_trunc
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_mtx_s2_trunc_v;

    // Inputs
    reg clk;
    reg [143:0] row;
    reg [143:0] column;

    // Outputs
    wire [35:0] RX_0;
    wire [35:0] RX_1;
    wire [35:0] RX_2;
    wire [35:0] RX_3;
    wire [35:0] RX_4;
    wire [35:0] RX_5;
    wire [35:0] RX_6;
    wire [35:0] RX_7;
    wire [36:0] RI_0;
    wire [36:0] RI_1;
    wire [36:0] RI_2;
    wire [36:0] RI_3;
    wire [37:0] RM_0;
    wire [37:0] RM_1;
    wire [38:0] product;

    // Instantiate the Unit Under Test (UUT)
    matrix_naive_s2_trunc uut (
        .clk(clk),
        .row(row),
        .column(column),
        .RX_0(RX_0),
        .RX_1(RX_1),
        .RX_2(RX_2),
        .RX_3(RX_3),

```

```

        .RX_4(RX_4),
        .RX_5(RX_5),
        .RX_6(RX_6),
        .RX_7(RX_7),
        .RI_0(RI_0),
        .RI_1(RI_1),
        .RI_2(RI_2),
        .RI_3(RI_3),
        .RM_0(RM_0),
        .RM_1(RM_1),
        .product(product)
    );

always #2 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    row = 0;
    column = 0;

    // Wait 100 ns for global reset to finish
    #5;

    // Add stimulus here
    #4 row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};
    column = {18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1};

    #4 row = {9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2};
    column = {18'd2, 18'd2, 18'd2, 18'd2, 18'd2, 18'd2, 18'd2, 18'd2};

    #4 row = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};
    column = {18'd1, 18'd2, 18'd3, 18'd4, 18'd5, 18'd6, 18'd7, 18'd8};

    #4 row = {9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF};
    column = {18'hFF, 18'hFF, 18'hFF, 18'hFF, 18'hFF, 18'hFF, 18'hFF, 18'hFF};

    #4 row = {9'hFF0, 9'hFF0, 9'hFF0, 9'hFF0, 9'hFF0, 9'hFF0, 9'hFF0, 9'hFF0};
    column = {18'h3FFF0, 18'h3FFF0, 18'h3FFF0, 18'h3FFF0, 18'h3FFF0, 18'h3FFF0,
18'h3FFF0, 18'h3FFF0, 18'h3FFF0};

    #4 row = {9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF, 9'h1FF};
    column = {18'hFFF0, 18'hFFF0, 18'hFFF0, 18'hFFF0, 18'hFFF0, 18'hFFF0, 18'hFFF0,
18'hFFF0, 18'hFFF0};

    #4 row = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};
    column = {18'd1, 18'd2, 18'd3, 18'd4, 18'd5, 18'd6, 18'd7, 18'd8};

    #4 row = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};
    column = {18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1, 18'd1};

    #4 row = {9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0};
    column = {18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0};

end

endmodule

```



```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:48:11 05/13/2006
// Design Name:    mult_shift_reg
// Module Name:    tb_mult_shift_reg.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: mult_shift_reg
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_mult_shift_reg_v;

    // Inputs
    reg clk;
    reg reset;
    reg [17:0] coef;
    reg          rdy;

    // Outputs
    wire [143:0] column;
    wire          done;

    wire [17:0]    sreg_seg[7:0];

    // Instantiate the Unit Under Test (UUT)
    mult_shift_reg uut (
        .clk(clk),
        .reset(reset),
        .coef(coef),
        .rdy(rdy),
        .column(column),
        .done(done)
    );

    always #1 clk = ~clk;

    assign    sreg_seg[0] = column[17:0];
    assign    sreg_seg[1] = column[35:18];
    assign    sreg_seg[2] = column[53:36];
    assign    sreg_seg[3] = column[71:54];
    assign    sreg_seg[4] = column[89:72];
    assign    sreg_seg[5] = column[107:90];
    assign    sreg_seg[6] = column[125:108];

```

```
assign      sreg_seg[7] = column[143:126];

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    coef = 0;
    rdy = 1;

    // Wait 100 ns for global reset to finish
    #98 reset = 1;

    // Add stimulus here
    repeat (15) begin
        #2 coef = coef + 1;
    end

    #5 rdy = 0;

    repeat (15) begin
        #2 coef = coef + 1;
    end

    #10 reset = 0;
    #10 reset = 1;

end // initial begin

endmodule // tb_mult_shift_reg_v
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:05:21 05/02/2006
// Design Name:    rs232_senderFSM_new
// Module Name:    tb_senderFSM_new.v
// Project Name:   final_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: rs232_senderFSM_new
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_senderFSM_new_v;

    // Inputs
    reg clk;
    reg reset;
    reg [7:0] data;
    reg cts;
    reg send;

    // Outputs
    wire txd;
    wire rts;
    wire ready;
    wire [3:0] state;

    // Instantiate the Unit Under Test (UUT)
    rs232_senderFSM_new uut (
        .clk(clk),
        .reset(reset),
        .data(data),
        .cts(cts),
        .send(send),
        .txd(txd),
        .rts(rts),
        .ready(ready),
        .state(state)
    );

    defparam uut.BAUD_COUNTER = 5;
    always #5 clk = ~clk;
    initial begin
        // Initialize Inputs
        clk = 0;

```

```
reset = 0;
data = 8'b00000001;
cts = 1;
send = 0;

// Wait 100 ns for global reset to finish
#100;

// Add stimulus here
reset = 1;
send = 1;
#20;
cts = 0;
#10;
cts = 1;
```

```
end
```

```
endmodule
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    02:18:44 05/15/2006
// Design Name:    tx_shift_reg
// Module Name:    tb_tx_shift_reg.v
// Project Name:   final_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: tx_shift_reg
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_tx_shift_reg_v;

    // Inputs
    reg clk;
    reg reset;
    reg [79:0] encoded_data;
    reg ready;
    reg send_block;

    // Outputs
    wire send;
    wire done;
    wire [7:0] encoded_byte;
    wire [1:0] state;

    // Instantiate the Unit Under Test (UUT)
    tx_shift_reg uut (
        .clk(clk),
        .reset(reset),
        .encoded_data(encoded_data),
        .ready(ready),
        .send_block(send_block),
        .send(send),
        .done(done),
        .encoded_byte(encoded_byte),
        .state(state)
    );

    always #5 clk = ~clk;
    initial begin
        // Initialize Inputs
        clk = 0;

```


endmodule

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    19:50:31 05/13/2006
// Design Name:    video_memory
// Module Name:    tb_video_memory.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: video_memory
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_video_memory_v;

    // Inputs
    reg wr_clk;
    reg [9:0] addra;
    reg [63:0] dina;
    reg          wea;
    reg          r_clk;
    reg [4:0] line_read;
    reg [4:0] block_read;
    reg [2:0] inter_row_cnt;

    // Outputs
    wire [9:0] addrb;
    wire [63:0] output_row;

    // Instantiate the Unit Under Test (UUT)
    video_memory uut (
        .wr_clk(wr_clk),
        .addra(addra),
        .dina(dina),
        .wea(wea),
        .r_clk(r_clk),
        .line_read(line_read),
        .block_read(block_read),
        .inter_row_cnt(inter_row_cnt),
        .addrb(addrb),
        .output_row(output_row)
    );

    always #20 r_clk = ~r_clk;

    initial begin

```

```
// Initialize Inputs
wr_clk = 0;
addra = 0;
dina = 0;
wea = 0;
r_clk = 0;
line_read = 0;
block_read = 0;
inter_row_cnt = 0;

// Wait 100 ns for global reset to finish
#90;

// Add stimulus here
repeat (30) begin
  repeat (30) begin
    repeat (7) begin
      #40 inter_row_cnt = inter_row_cnt + 1;
    end
    block_read = block_read + 1;
    inter_row_cnt = 0;
  end
  line_read = line_read + 1;
  block_read = 0;
end
end // initial begin

endmodule // tb_video_memory_v
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:52:37 05/14/2006
// Design Name:    wireless_memory
// Module Name:    tb_wireless_memory.v
// Project Name:   finalproject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: wireless_memory
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_wireless_memory_v;

    // Inputs
    reg wr_clk;
    reg [9:0] wr_addr;
    reg [77:0] din;
    reg          wen;
    reg          r_clk;
    reg [9:0] read_addr;

    // Outputs
    wire [77:0] dout;

    // Instantiate the Unit Under Test (UUT)
    wireless_memory uut (
        .wr_clk(wr_clk),
        .wr_addr(wr_addr),
        .din(din),
        .wen(wen),
        .r_clk(r_clk),
        .read_addr(read_addr),
        .dout(dout)
    );

    initial begin
        // Initialize Inputs
        wr_clk = 0;
        wr_addr = 0;
        din = 0;
        wen = 0;
        r_clk = 0;
        read_addr = 0;
    end
endmodule

```

```
        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end

endmodule
```

```

/*****
*
*          *****
*          *****
*          ***      ***
*          ***   +++   ***
*          ***   + +   ***
*          ***     +
*          ***   + +   ***
*          ***   +++   ***
*          ***
*          *****
*          *****
*
*****
* This program demonstrates the use of the CC2420DB library, including the basic RF library. The
* packet protocol being used is a small subset of the IEEE 802.15.4 standard. It uses an 802.15.4 MAC
* compatible frame format, but does not implement any other MAC functions/mechanisms (e.g. CSMA-CA).
* The basic RF library can thus not be used to communicate with compliant 802.15.4 networks.
*
* A pair of CC2420DBs running this program will establish a point-to-point RF link on channel 26,
* using the following node addresses:
* - PAN ID: 0x2420 (both nodes)
* - Short address:
*           0x1234 if the joystick is moved in any direction at startup
*           0x5678 if the joystick button is pressed down at startup
*
* Please note that there is no so-called (PAN) coordinator.
*
* INSTRUCTIONS:
* Data packets containing a 5-byte payload will be transmitted when the pot meter is turned, or S2 is
* held down. The first byte of the payload contains the pot meter value, which is used to control the
* PWM duty cycle on the receiving node. The other bytes are random (never initialized).
*
* LED indicators:
* - Red:    Transmission failed (acknowledgment not received)
* - Yellow: Transmission OK (acknowledgment received)
* - Orange: Remote controlled dimmer
*
* - Green:  Packet received
*****
* Compiler: AVR-GCC
* Target platform: CC2420DB (can easily be ported to other platforms)
*****
* Revision history:
* $Log: rf_blink_led.c,v $
* Revision 1.5  2004/07/26 11:18:13  mbr
* Changed PANID from 0xDEAD to 0x2420
*
* Revision 1.4  2004/04/05 08:25:52  mbr
* Comments changed in header
*
* Revision 1.3  2004/03/30 14:58:27  mbr
* Release for web
*
*
*
*****/
#include <include.h>

//-----
// Basic RF transmission and reception structures
BASIC_RF_RX_INFO rfrxInfo;
BASIC_RF_TX_INFO rftxInfo;
BYTE pTxBuffer[BASIC_RF_MAX_PAYLOAD_SIZE];
BYTE pRxBuffer[BASIC_RF_MAX_PAYLOAD_SIZE];
BOOL senderReceiver;
//-----

```

```

//-----
// BASIC_RF_RX_INFO* basicRfReceivePacket(BASIC_RF_RX_INFO *pRRI)
//
// DESCRIPTION:
//     This function is a part of the basic RF library, but must be declared by the application. Once
//     the application has turned on the receiver, using basicRfReceiveOn(), all incoming packets will
//     be received by the FIFOP interrupt service routine. When finished, the ISR will call the
//     basicRfReceivePacket() function. Please note that this function must return quickly, since the
//     next received packet will overwrite the active BASIC_RF_RX_INFO structure (pointed to by pRRI).
//
// ARGUMENTS:
//     BASIC_RF_RX_INFO *pRRI
//     The reception structure, which contains all relevant info about the received packet.
//
// RETURN VALUE:
//     BASIC_RF_RX_INFO*
//     The pointer to the next BASIC_RF_RX_INFO structure to be used by the FIFOP ISR. If there
//     is
//     only one buffer, then return pRRI.
//-----
BASIC_RF_RX_INFO* basicRfReceivePacket(BASIC_RF_RX_INFO *pRRI) {
    UINT8 n;

    // Adjust the led brightness
    //PWM0_SET_DUTY_CYCLE(pRRI->pPayload[0]);

    if (senderReceiver == FALSE) {

        for (n = 0; n < 10; n++) { //BASIC_RF_MAX_PAYLOAD_SIZE-1; n++) {
            // Send a request to send to the RS232 receiver module
            CLR_RTS();

            // Wait until CTS is low signaling that it is okay to send a byte to the receiver FPGA
            // if CTS is high (inactive) disable packet reception until CTS goes back low
            while (TRUE) {
                if (BM(UART1_CTS) & PIND) {
                    basicRfReceiveOff();
                    // Set the green led while the receiver is waiting on CTS to go low
                    SET_GLED();
                }
                else {
                    basicRfReceiveOn();
                    CLR_GLED();
                    break;
                }
            }

            // Clear the request to send signal
            SET_RTS();

            // Send the byte to the RS232 receiver
            if (n>=1) {
                UART1_WAIT_AND_SEND(pRRI->pPayload[n+1]);
            }
            else {
                UART1_WAIT_AND_SEND(pRRI->pPayload[n]);
            }
        }

    }

    // Continue using the (one and only) reception structure
    return pRRI;
} // basicRfReceivePacket

```



```

//-----
// void main (void)
//
// DESCRIPTION:
// Startup routine and main loop
//-----
int main (void) {
    //UIN16 ledDutyCycle, dimmerDifference;
    UIN16 n, test;
    BOOL success, firstByte; //, senderReceiver;

    // Initialize ports for communication with CC2420 and other peripheral units
    PORT_INIT();
    SPI_INIT();

    // Initialize PWM0 with a period of CLK/1024
    PWM0_INIT(TIMER_CLK_DIV1024);

    // Initialize and enable the ADC for reading the pot meter
    ADC_INIT();
    ADC_SET_CHANNEL(ADC_INPUT_0_POT_METER);
    ADC_ENABLE();

    // Wait for the user to select node address, and initialize for basic RF operation
    while (TRUE) {
        if (JOYSTICK_CENTER_PRESSED()) {
            senderReceiver = TRUE;
            basicRfInit(&rfrxInfo, 26, 0x2420, 0x1234);
            rfTxInfo.destAddr = 0x5678;
            break;
        } else if (JOYSTICK_UP_PRESSED()
            || JOYSTICK_DOWN_PRESSED()
            || JOYSTICK_LEFT_PRESSED()
            || JOYSTICK_RIGHT_PRESSED()) {

            senderReceiver = FALSE;
            basicRfInit(&rfrxInfo, 26, 0x2420, 0x5678);
            rfTxInfo.destAddr = 0x1234;
            break;
        }
    }

    // Initialize common protocol parameters
    rfTxInfo.length = 11; //BASIC_RF_MAX_PAYLOAD_SIZE - 1;
    rfTxInfo.ackRequest = TRUE;
    rfTxInfo.pPayload = pTxBuffer;
    rfrxInfo.pPayload = pRxBuffer;

    // Turn on RX mode
    basicRfReceiveOn();

//TEST CODE

    ENABLE_UART1();
    INIT_UART1(UART_BAUDRATE_250K, UART_OPT_8_BITS_PER_CHAR);

    firstByte = TRUE;
    while (senderReceiver) {
        // Wait for sender to issue a request to send
        while (TRUE) {
            if ((BM(UART1_CTS) & PIND)==0b00000000) {
                break;
            }
        }

        // Tell fpga to start sending data
        CLR_RTS();
    }
}

```

```

// fill transmit buffer with incoming serial data
if (firstByte) {
    UART1_RECEIVE(pTxBuffer[0]);
    firstByte = FALSE;
}
else {
    UART1_WAIT_AND_RECEIVE(pTxBuffer[0]);
}
//pTxBuffer[0] = 0x44;
for (n = 1; n < 11; n++) { //BASIC_RF_MAX_PAYLOAD_SIZE - 1; n++) {
    UART1_WAIT_AND_RECEIVE(pTxBuffer[n]);
    //pTxBuffer[n] = 0x45;
}

// Tell fpga to stop sending data
SET_RTS();

// Send packet and wait for ACK (keep resending packet if no ACK is received)
do {
    success = basicRfSendPacket(&rftxInfo);
    success = TRUE;
    //basicRfSendPacket(&rftxInfo);
    SET_YLED();
}
while (success == FALSE);

CLR_YLED();

//halWait(65535);

//while (TRUE) {
//}
}

return 0;
} // main

```

```

////////////////////////////////////
//
// Lab Kit FINAL - Receive-Decoder-Display
//
////////////////////////////////////

wire  reset;
wire [2:0] resolution_select;
wire [4:0] max_lines, max_blocks;
wire [9:0] addr_active;
wire [103:0] row;
wire [63:0] data_output;
wire      decode_busy;
wire [2:0]   inter_row_cnt;
wire [4:0]   block_read, line_read;
wire [4:0]   block_write, line_write;
wire [9:0]   write_address, read_addr;
wire      wen;
wire [2:0]   input_select;
wire [2:0]   column_select, column_select_write;
wire [143:0] shift_reg_output;
wire [17:0]  stagel_output;

reg [12:0]   coef, analyzer_out, mult_out;
reg [12:0]   row_out;
reg [7:0]    data_in_out;
reg [17:0]   sr_out;

// Ray's wires
wire [63:0]  dout0, dout1, dout2, dout3, dout4, dout5, dout6, dout7;
wire [9:0]   addr;
wire [23:0]  vga_out;

// Noel's wires
wire [639:0] dots;
  wire [77:0] encoded_data;
wire [7:0]   encoded_byte;
wire [3:0]   rs232_state;
  wire [2:0] state;
  wire[1:0] shift_state;
wire      enable, data_ready, wen_wireless, done;
  wire [3:0] counter;

reg      rs232_rxd1, rs232_rxd2, rs232_rxd_sync;
reg      rs232_cts1, rs232_cts2, rs232_cts_sync;

assign   led = ~encoded_byte;

assign   analyzer1_clock = clock_27mhz;
assign   analyzer2_clock = clock_27mhz;
assign   analyzer3_clock = clock_27mhz;
assign   analyzer4_clock = clock_27mhz;

assign   analyzer1_data[12:0] = row_out;
assign   analyzer1_data[15:13] = input_select;

```



```

        .encoded_byte(encoded_byte),

        .rs232_state(rs232_state),

        .shift_state(shift_state),

        .write_addr(write_address),

        .wen(wen_wireless),

        .state(state),
        .done(done),
        .counter(counter),

        .data_ready(data_ready));
/*
  toplevel_test2 top(.clk(clock_27mhz),
                    .reset(reset),
                    .cts_b(rs232_cts_sync),
                    .rxd(rs232_rxd_sync),
                    .rts_b(rs232_rts),
                    .data(encoded_byte),
                    .state(state),
                    .enable(enable),
                    .data_ready(data_ready));
*/
alpha_display alpha_dots(.clock(clock_27mhz),
                        .ascii(1'b1),
                        .bits(encoded_byte),
                        .dots(dots[39:0]));

alphanumeric_displays a_disp(.global_clock(clock_27mhz),
                              .manual_reset(~reset),
                              .disp_test(switch[7]),
                              .disp_blank(disp_blank),
                              .disp_clock(disp_clock),
                              .disp_rs(disp_rs),
                              .disp_ce_b(disp_ce_b),
                              .disp_reset_b(disp_reset_b),
                              .disp_data_out(disp_data_out),
                              .dots(dots));

wireless_memory WIREMEMDECODE001(
    .wr_clk(clock_27mhz),
    .wr_addr(write_address),
    .din(encoded_data),
    .wen(wen_wireless),
    .r_clk(clock_27mhz),
    .line_read(line_read),
    .block_read(block_read),
    .inter_row_cnt(inter_row_cnt),
    .dout(row)
);

decoder DECODE001(
    .clk(clock_27mhz),
    .reset(reset),
    .resolution_select(resolution_select),

```

```

.max_lines(max_lines),
.max_blocks(max_blocks),
.addr_active(write_address),
.row(row),
.decode_busy(decode_busy),
.inter_row_cnt(inter_row_cnt),
.block_read(block_read),
.line_read(line_read),
.data_output(data_output),
.wen(wen),
.mult_column(mult_column),
.column_select(column_select),
.input_select(input_select),
.stagel_output(stagel_output),
.shift_reg_output(shift_reg_output),
.line_write(line_write),
.block_write(block_write),
.column_select_write(column_select_write)
);

```

```

video_memory VIDMEMMOD001D(
    .wr_clk(clock_27mhz),
    .r_clk(clock_27mhz),
    .wen(wen),
    .line_write(line_write),
    .block_write(block_write),
    .column_select_write(column_select_write),
    .column_data(data_output),
    .addrb(addr),
    .dout0(dout0),
    .dout1(dout1),
    .dout2(dout2),
    .dout3(dout3),
    .dout4(dout4),
    .dout5(dout5),
    .dout6(dout6),
    .dout7(dout7),
    .addra()
);

```

```

video_blackwhite VIDBW001(
    .clk(clock_27mhz),
    .reset(~reset),
    .dout0(dout0),
    .dout1(dout1),
    .dout2(dout2),
    .dout3(dout3),
    .dout4(dout4),
    .dout5(dout5),
    .dout6(dout6),
    .dout7(dout7),
    .addr(addr),
    .vga_out_sync_b(vga_out_sync_b),
    .vga_out_blank_b(vga_out_blank_b),
    .vga_out_hsync(vga_out_hsync),
    .vga_out_vsync(vga_out_vsync),
    .vga_out(vga_out)
);

```

```

        );

assign    vga_out_pixel_clock = ~clock_27mhz;
assign    vga_out_red = vga_out[23:16];
assign    vga_out_green = vga_out[15:8];
assign    vga_out_blue = vga_out[7:0];

always @(switch[2:0])
    case (switch[2:0])
        0: analyzer_out[12:0] = data_output[7:0];
        1: analyzer_out[12:0] = data_output[15:8];
        2: analyzer_out[12:0] = data_output[23:16];
        3: analyzer_out[12:0] = data_output[31:24];
        4: analyzer_out[12:0] = data_output[39:32];
        5: analyzer_out[12:0] = data_output[47:40];
        6: analyzer_out[12:0] = data_output[55:48];
        7: analyzer_out[12:0] = data_output[63:56];
    endcase // case(switch[2:0])

always @(switch[2:0])
    case (switch[2:0])
        0: sr_out[17:0] = shift_reg_output[17:0];
        1: sr_out[17:0] = shift_reg_output[35:18];
        2: sr_out[17:0] = shift_reg_output[53:36];
        3: sr_out[17:0] = shift_reg_output[71:54];
        4: sr_out[17:0] = shift_reg_output[89:72];
        5: sr_out[17:0] = shift_reg_output[107:90];
        6: sr_out[17:0] = shift_reg_output[125:108];
        7: sr_out[17:0] = shift_reg_output[143:126];
    endcase // case(switch[2:0])

always @(switch[2:0])
    case (switch[2:0])
        0: row_out[12:0] = row[12:0];
        1: row_out[12:0] = row[25:13];
        2: row_out[12:0] = row[38:26];
        3: row_out[12:0] = row[51:39];
        4: row_out[12:0] = row[64:52];
        5: row_out[12:0] = row[77:65];
        6: row_out[12:0] = row[90:78];
        7: row_out[12:0] = row[103:91];
    endcase // case(switch[1:0])

always @(switch[2:0])
    case (switch[2:0])
        0: data_in_out[7:0] = dout0;
        1: data_in_out[7:0] = dout1;
        2: data_in_out[7:0] = dout2;
        3: data_in_out[7:0] = dout3;
        4: data_in_out[7:0] = dout4;
        5: data_in_out[7:0] = dout5;
        6: data_in_out[7:0] = dout6;
        7: data_in_out[7:0] = dout7;
    endcase // case(switch[1:0])

endmodule // labkit

```



```

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    09:29:20 05/15/06
// Design Name:
// Module Name:    rx_control_unit
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module rx_control_unit(clk,
                      reset,
                      cts_b,
                      rxd,
                      decode_busy,
                      rts_b,
                      enable,
                      encoded_data,
                      encoded_byte,
                      rs232_state,
                      shift_state,
                      wen,
                      write_addr,
                      state,
                      done,
                      counter,
                      data_ready);

    input clk, reset, cts_b, rxd, decode_busy;
    output rts_b, enable, wen;
    output[77:0] encoded_data;
    output[9:0] write_addr;
    output[7:0] encoded_byte;
    output[3:0] rs232_state;
    output[2:0] state;
    output[1:0] shift_state;
    output      data_ready;
    reg[9:0] write_addr;
    reg[2:0] state, next;
    reg wen_int, write_addr_inc;
    output done;
    output [3:0] counter;

    wire wen;

    wire data_ready, rx_ready;

```

```

parameter IDLE = 3'd0;
parameter WAIT = 3'd1;
parameter WRITE1 = 3'd2;
parameter WRITE2 = 3'd3;
parameter WRITE3 = 3'd4;

assign rx_ready = ~(decode_busy && (write_addr == 10'd0));
assign wen = (reset) ? done : 0;

always @ (posedge clk) begin
    if (!reset) begin
        write_addr <= 10'd0;
        state <= IDLE;
    end
    else if (done)
        begin
            if (write_addr == 10'd899) write_addr <= 0;
            else write_addr <= write_addr + 1;
        end
    else
        write_addr <= write_addr;
end

rx_shift_reg rxsr(.clk(clk),
                  .reset(reset),
                  .data_ready(data_ready),
                  .encoded_byte(encoded_byte),
                  .done(done),
                  .encoded_data(encoded_data),
                  .state(shift_state),
                  .counter(counter));

rs232_receiverFSM rx(.clk(clk),
                    .reset(reset),
                    .cts_b(cts_b),
                    .rxd(rxd),
                    .data(encoded_byte),
                    .data_ready(data_ready),
                    .rts_b(rts_b),
                    .state(rs232_state),
                    .enable(enable),
                    .rx_ready(rx_ready));

endmodule

```

```

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:17:46 05/13/06
// Design Name:
// Module Name:    rs232_receiverFSM
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module rs232_receiverFSM(clk,
                        reset,
                        cts_b,
                        rxd,
                        rx_ready,
                        data,
                        data_ready,
                        rts_b,
                        state,
                        enable);

    input clk, reset, cts_b, rxd, rx_ready;
    output[7:0] data;
    output[3:0] state;
    output data_ready, rts_b, enable;
    reg[3:0] state, next, index, index_int;
    reg rts_b, rts_int, data_ready_int, data_ready, counting,
        counting_int, first_count, first_count_int, enable, data_int;
    reg[11:0] count;
    reg[9:0] data_and_flags;
    reg[7:0] data;

    parameter BAUD_COUNTER = 12'd107;
    parameter HALF_BAUD_COUNTER = 12'd53;
    parameter RECEIVE_STOP_BIT = 4'd0;
    parameter RECEIVE_START_BIT = 4'd1;
    parameter RECEIVE_DATA_BIT_0 = 4'd2;
    parameter RECEIVE_DATA_BIT_1 = 4'd3;
    parameter RECEIVE_DATA_BIT_2 = 4'd4;
    parameter RECEIVE_DATA_BIT_3 = 4'd5;
    parameter RECEIVE_DATA_BIT_4 = 4'd6;
    parameter RECEIVE_DATA_BIT_5 = 4'd7;
    parameter RECEIVE_DATA_BIT_6 = 4'd8;
    parameter RECEIVE_DATA_BIT_7 = 4'd9;
    parameter IDLE = 4'd10;

    always @ (posedge clk) begin

```

```

    if (!reset) begin
        data_ready <= 0;
        rts_b <= 0;
        count <= 0;
        counting <= 0;
        first_count <= 0;
data <= 8'hFF;
        state <= RECEIVE_STOP_BIT;
    end
    else begin
        if ((counting && first_count && (count==HALF_BAUD_COUNTER)) ||
            (counting && !first_count && (count==BAUD_COUNTER))) begin
            enable <= 1;
            count <= 0;
        end
        else if (counting) begin
            enable <= 0;
            count <= count + 12'd1;
        end
        else begin
            enable <= 0;
            count <= 0;
        end

        if (data_ready) begin
data <= data_and_flags[8:1];
        end

        counting <= counting_int;
        first_count <= first_count_int;
        data_ready <= data_ready_int;
        rts_b <= rts_int;
        state <= next;
    end
end

always @ (state or enable or rxd) begin
    data_ready_int = 0;
    //rts_int = ~rx_ready; // ready to receive if RX shift reg is ready
    rts_int = 1;
    case (state)
        RECEIVE_STOP_BIT:
            begin
                rts_int = 0;
                first_count_int = 0;
                counting_int = 0;
                if (rxd == 0) begin
                    data_and_flags[9] = 1;
                    next = RECEIVE_START_BIT;
                end
                else begin
                    next = RECEIVE_STOP_BIT;
                end
            end
        RECEIVE_START_BIT:
            begin
                first_count_int = 1;
            end
    end
end

```

```

        counting_int = 1;
        if (enable) begin
            data_and_flags[0] = 0;
            next = RECEIVE_DATA_BIT_0;
        end
        else next = RECEIVE_START_BIT;
    end
end
RECEIVE_DATA_BIT_0:
begin
    first_count_int = 0;
    counting_int = 1;
    if (enable) begin
        data_and_flags[1] = rxd;
        next = RECEIVE_DATA_BIT_1;
    end
    else begin
        next = RECEIVE_DATA_BIT_0;
    end
end
end

RECEIVE_DATA_BIT_1:
begin
    first_count_int = 0;
    counting_int = 1;
    if (enable) begin
        data_and_flags[2] = rxd;

        next = RECEIVE_DATA_BIT_2;
    end
    else begin
        next = RECEIVE_DATA_BIT_1;
    end
end
end

RECEIVE_DATA_BIT_2:
begin
    first_count_int = 0;
    counting_int = 1;
    if (enable) begin
        data_and_flags[3] = rxd;

        next = RECEIVE_DATA_BIT_3;
    end
    else begin
        next = RECEIVE_DATA_BIT_2;
    end
end
end

RECEIVE_DATA_BIT_3:
begin
    first_count_int = 0;
    counting_int = 1;
    if (enable) begin
        data_and_flags[4] = rxd;

        next = RECEIVE_DATA_BIT_4;
    end
    else begin
        next = RECEIVE_DATA_BIT_3;
    end
end
end

```

```

        end
    end
RECEIVE_DATA_BIT_4:
    begin
        first_count_int = 0;
        counting_int = 1;
        if (enable) begin
            data_and_flags[5] = rxd;

            next = RECEIVE_DATA_BIT_5;
        end
        else begin
            next = RECEIVE_DATA_BIT_4;
        end
    end
end
RECEIVE_DATA_BIT_5:
    begin
        first_count_int = 0;
        counting_int = 1;
        if (enable) begin
            data_and_flags[6] = rxd;

            next = RECEIVE_DATA_BIT_6;
        end
        else begin
            next = RECEIVE_DATA_BIT_5;
        end
    end
end
RECEIVE_DATA_BIT_6:
    begin
        first_count_int = 0;
        counting_int = 1;
        if (enable) begin
            data_and_flags[7] = rxd;

            next = RECEIVE_DATA_BIT_7;
        end
        else begin
            next = RECEIVE_DATA_BIT_6;
        end
    end
end
RECEIVE_DATA_BIT_7:
    begin
        first_count_int = 0;
        counting_int = 1;
        if (enable) begin
            data_and_flags[8] = rxd;

            //data_ready_int = 1;
            next = IDLE;
        end
        else begin
            next = RECEIVE_DATA_BIT_7;
        end
    end
end
IDLE:
    begin

```

```
        first_count_int = 0;
        counting_int = 0;
        if (rxd) begin
            data_and_flags[9] = 1;
            data_ready_int = 1;

            next = RECEIVE_STOP_BIT;
        end
        else begin
            next = IDLE;
        end
    end
end
endcase
end
endmodule
```

```

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:44:43 05/07/06
// Design Name:
// Module Name:    rx_shift_reg
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module rx_shift_reg(clk,
                    reset,
                    data_ready,
                    encoded_byte,
                    done,
                    encoded_data,
                    state,
                    counter);

    input clk, reset, data_ready;
    input [7:0] encoded_byte;
    output      done, state;
    output [77:0] encoded_data;
    output [3:0] counter;
    reg [79:0]    encoded_data_r;
    reg          done, done_int, data_sample, data_sample_int;
    reg [1:0]    state, next;
    reg          index_inc;
    reg [6:0]    index;
    reg [3:0]    counter;

    parameter    IDLE = 0;
    parameter    WAIT = 1;
    parameter    SAMPLE_AND_SHIFT = 2;

    assign encoded_data = encoded_data_r[77:0];

    always @ (posedge clk) begin
        if (!reset)
            begin
                done <= 0;
                data_sample <= 0;
                index <= 7'd7;
                encoded_data_r <= 80'h0;
                state <= IDLE;
                counter <= 13;
            end
    end
end

```



```
else
  //begin
  if (data_ready)
    begin
      encoded_data_r <= {encoded_byte, encoded_data_r[79:8]};
      counter <= counter + 1;
      //done <= (counter == 9)? 1 : 0;
    end
  else if (counter == 10)
    begin
      done <= 1;
      counter <= counter + 1;
    end
  else if (counter == 11)
    begin
      done <= 0;
      counter <= counter + 1;
    end
  else
    counter <= (counter == 12)? 0 : counter;
end
endmodule
```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Decoder - Wireless Memory Module
// 05/14/2006
//
////////////////////////////////////

module wireless_memory(
    wr_clk,
    wr_addr,
    din,
    wen,
    r_clk,
    line_read,
    block_read,
    inter_row_cnt,
    dout
);

input wr_clk, r_clk;
input [9:0] wr_addr;
input [77:0] din;
input wen;
input [4:0] line_read, block_read;
input [2:0] inter_row_cnt;

output [103:0] dout;

wire [9:0] addrb;

reg [103:0] dout;
wire [77:0] memory_out;
wire [12:0] coef[5:0];

assign addrb = 30*line_read + block_read;

assign coef[5] = memory_out[12:0];
assign coef[4] = memory_out[25:13];
assign coef[3] = memory_out[38:26];
assign coef[2] = memory_out[51:39];
assign coef[1] = memory_out[64:52];
assign coef[0] = memory_out[77:65];

always @(inter_row_cnt)
    case (inter_row_cnt)
        0: dout = {coef[0], coef[3], coef[5], 65'b0};
        1: dout = {coef[1], coef[4], 78'b0};
        2: dout = {coef[2], 91'b0};
        default: dout = 104'b0;
    endcase // case(inter_row_cnt)

```

```
wireless_mem WIREMEM0(.clka(wr_clk), .clkb(r_clk), .wea(wen),  
.addra(wr_addr), .addrb(addrb), .dina(din), .doutb(memory_out));  
  
endmodule // wireless_memory
```

```
////////////////////////////////////  
//  
// Vivek Shah  
// Final Project  
//  
//  
// Decoder  
// 05/05/2006  
//  
////////////////////////////////////
```

```
module decoder(  
    clk,  
    reset,  
    resolution_select,  
    max_lines,  
    max_blocks,  
    addr_active,  
    row,  
    decode_busy,  
    inter_row_cnt,  
    block_read,  
    line_read,  
    data_output,  
    wen,  
    mult_column,  
    column_select,  
    input_select,  
    stagel_output,  
    shift_reg_output,  
    line_write,  
    block_write,  
    column_select_write  
);  
  
input clk, reset;  
input [2:0] resolution_select;  
input [4:0] max_lines, max_blocks;  
input [9:0] addr_active;  
input [103:0] row;  
  
output decode_busy;  
output [4:0] block_read, line_read;  
output [2:0] inter_row_cnt;  
output [63:0] data_output;  
output [103:0] mult_column;  
output [2:0] column_select;  
output [2:0] input_select;  
output [4:0] line_write, block_write;  
output [2:0] column_select_write;  
output wen;  
  
output [17:0] stagel_output;  
output [143:0] shift_reg_output;  
  
wire [2:0] column_select;  
wire [103:0] mult_column;
```

```

wire [17:0]      stagel_output;
wire [143:0]    shift_reg_output;

// Instantiate Multiply Unit
dct_multiply_decode DCTMULTIPLY1D(
    .clk(clk),
    .reset(reset),
    .row(row),
    .column_select(column_select),
    .output_column(data_output),
    .stagel_output(stagel_output),
    .shift_reg_output(shift_reg_output)
);

// Instantiate Decoder FSM
decoder_fsm DEOCDEFSM1(
    .clk(clk),
    .reset(reset),
    .resolution_select(resolution_select),
    .max_blocks(max_blocks),
    .max_lines(max_lines),
    .addr_active(addr_active),
    .line_read(line_read),
    .block_read(block_read),
    .line_write(line_write),
    .block_write(block_write),
    .column_select(column_select),
    .column_select_write(column_select_write),
    .inter_row_cnt(inter_row_cnt),
    .input_select(input_select),
    .decode_busy(decode_busy),
    .state(state),
    .wen(wen)
);

// Instantiate Variable Shift Register
/* -----\/------ EXCLUDED -----\/------
decode_memory_register DECODEMEMREG1(.clk(clk),
    .reset(reset),
    .load_select(output_select),
    .max_lines(max_lines),
    .max_blocks(max_blocks),
    .resolution_select(resolution_select),
    .coef7(mult_column[12:0] ),
    .coef6(mult_column[25:13]),
    .coef5(mult_column[38:26]),
    .coef4(mult_column[51:39]),
    .coef3(mult_column[64:52]),
    .coef2(mult_column[77:65]),
    .coef1(mult_column[90:78]),
    .coef0(mult_column[103:91]),
    .data_output(data_output),
    .address(write_address),
    .wen(wen));
-----/\----- EXCLUDED -----/\----- */

endmodule // decoder

```



```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// DCT Multiply Module - Decoder
// 05/12/2006
//
////////////////////////////////////

module dct_multiply_decode(
    clk,
    reset,
    row,
    column_select,
    output_column,
    output_trunc,
    stagel_output,
    shift_reg_output
);

input clk;
input reset;
input [103:0] row;
input [2:0] column_select;

output [63:0] output_column;
output [239:0] output_trunc;
output [17:0] stagel_output;
output [143:0] shift_reg_output;

wire [17:0] stagel_output;
wire [143:0] shift_reg_output;
wire [239:0] output_trunc;

wire [71:0] dct_row[7:0];
wire [71:0] dct_col[7:0];
reg [71:0] dct_col_final;

// Instantiate DCT Table
dct_table DCTTABLE0D(.clk(clk),
    .row0(dct_row[0]),
    .row1(dct_row[1]),
    .row2(dct_row[2]),
    .row3(dct_row[3]),
    .row4(dct_row[4]),
    .row5(dct_row[5]),
    .row6(dct_row[6]),
    .row7(dct_row[7]),
    .col0(dct_col[0]),
    .col1(dct_col[1]),
    .col2(dct_col[2]),
    .col3(dct_col[3]),
    .col4(dct_col[4]),
    .col5(dct_col[5]),
    .col6(dct_col[6]),

```

```

        .col7(dct_col[7])
    );

// Front set of multipliers and truncate-ors
dct_front_decode DCTFRONT0D(.clk(clk),
    .row(row),
    .column(dct_col_final),
    .output_coef(stagel_output));

// Shift Register
mult_shift_reg MULTSHFTREG1D(
    .clk(clk),
    .reset(reset),
    .coef(stagel_output),
    .rdy(1'b1),
    .column(shift_reg_output),
    .done()
);

// Instantiate second set of multipliers
dct_back_decode DCTBACK0D(.clk(clk),
    .row0(dct_col[0]),
    .row1(dct_col[1]),
    .row2(dct_col[2]),
    .row3(dct_col[3]),
    .row4(dct_col[4]),
    .row5(dct_col[5]),
    .row6(dct_col[6]),
    .row7(dct_col[7]),
    .column(shift_reg_output),
    .output_trunc(output_trunc),
    .output_column(output_column));

// Decides which DCT column to multiply against the input matrix
always @(column_select)
begin
    case (column_select)
        0: dct_col_final = dct_col[0];
        1: dct_col_final = dct_col[1];
        2: dct_col_final = dct_col[2];
        3: dct_col_final = dct_col[3];
        4: dct_col_final = dct_col[4];
        5: dct_col_final = dct_col[5];
        6: dct_col_final = dct_col[6];
        7: dct_col_final = dct_col[7];
    endcase // case(column_select)
end // always @ (column_select)

endmodule // dct_multiply_decode

```



```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// DCT Front - First stage of multiply module - decode
// 05/12/2006
//
////////////////////////////////////

module dct_front_decode(
    clk,
    row,
    column,
    output_trunc,
    output_coef
);

    input clk;
    input [103:0] row;
    input [71:0] column;

    output [17:0] output_coef;
    output [24:0] output_trunc;

    wire [24:0]    coef;

    // Debugging vector
    assign    output_trunc = coef;

    // Instantiate first set of multipliers
    matrix_naive_decode MTX0D(.clk(clk), .row(row), .column(column),
    .product(coef));

    // First bank of truncate-ors
    trunc_sl_decode TRUNCS100D(.input_coef(coef), .output_coef(output_coef));

endmodule // dct_front_decode

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Single Matrix Coefficient Generator - Naive Implementation - Decoder
// 05/12/2006
//
////////////////////////////////////

module matrix_naive_decode(
    clk,
    row,
    column,
    RX_0, RX_1, RX_2, RX_3, RX_4, RX_5, RX_6, RX_7,
    RI_0, RI_1, RI_2, RI_3,
    RM_0, RM_1,
    product);

input clk;
input [103:0] row;
input [71:0] column;

output signed [21:0] RX_0, RX_1, RX_2, RX_3, RX_4, RX_5, RX_6, RX_7;
output signed [22:0] RI_0, RI_1, RI_2, RI_3;
output signed [23:0] RM_0, RM_1;
output signed [24:0] product;

wire signed [12:0] row_pixel [7:0];
wire signed [8:0] col_pixel [7:0];

wire signed [21:0] RX[7:0];
wire signed [22:0] RI[3:0];
wire signed [23:0] RM[1:0];
wire signed [24:0] product;

assign RX_0 = RX[0];
assign RX_1 = RX[1];
assign RX_2 = RX[2];
assign RX_3 = RX[3];
assign RX_4 = RX[4];
assign RX_5 = RX[5];
assign RX_6 = RX[6];
assign RX_7 = RX[7];

assign RI_0 = RI[0];
assign RI_1 = RI[1];
assign RI_2 = RI[2];
assign RI_3 = RI[3];

assign RM_0 = RM[0];
assign RM_1 = RM[1];

assign row_pixel[0] = row[12:0];
assign row_pixel[1] = row[25:13];
assign row_pixel[2] = row[38:26];

```

```

assign    row_pixel[3] = row[51:39];
assign    row_pixel[4] = row[64:52];
assign    row_pixel[5] = row[77:65];
assign    row_pixel[6] = row[90:78];
assign    row_pixel[7] = row[103:91];

assign    col_pixel[0] = column[8:0];
assign    col_pixel[1] = column[17:9];
assign    col_pixel[2] = column[26:18];
assign    col_pixel[3] = column[35:27];
assign    col_pixel[4] = column[44:36];
assign    col_pixel[5] = column[53:45];
assign    col_pixel[6] = column[62:54];
assign    col_pixel[7] = column[71:63];

mult_sign_13_9 MULT0(.clk(clk), .a(row_pixel[0]), .b(col_pixel[0]),
.q(RX[0]));
mult_sign_13_9 MULT1(.clk(clk), .a(row_pixel[1]), .b(col_pixel[1]),
.q(RX[1]));
mult_sign_13_9 MULT2(.clk(clk), .a(row_pixel[2]), .b(col_pixel[2]),
.q(RX[2]));
mult_sign_13_9 MULT3(.clk(clk), .a(row_pixel[3]), .b(col_pixel[3]),
.q(RX[3]));
mult_sign_13_9 MULT4(.clk(clk), .a(row_pixel[4]), .b(col_pixel[4]),
.q(RX[4]));
mult_sign_13_9 MULT5(.clk(clk), .a(row_pixel[5]), .b(col_pixel[5]),
.q(RX[5]));
mult_sign_13_9 MULT6(.clk(clk), .a(row_pixel[6]), .b(col_pixel[6]),
.q(RX[6]));
mult_sign_13_9 MULT7(.clk(clk), .a(row_pixel[7]), .b(col_pixel[7]),
.q(RX[7]));

adder_sign_22 ADD00(.A(RX[0]), .B(RX[1]), .S(RI[0]));
adder_sign_22 ADD01(.A(RX[2]), .B(RX[3]), .S(RI[1]));
adder_sign_22 ADD02(.A(RX[4]), .B(RX[5]), .S(RI[2]));
adder_sign_22 ADD03(.A(RX[6]), .B(RX[7]), .S(RI[3]));

adder_sign_23 ADD10(.A(RI[0]), .B(RI[1]), .S(RM[0]));
adder_sign_23 ADD11(.A(RI[2]), .B(RI[3]), .S(RM[1]));

adder_sign_24 ADD20(.A(RM[0]), .B(RM[1]), .S(product));

endmodule // matrix_naive_decode

```

```
////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Truncate Stage 1 (25 bits -> 18 bits) - Deocode
// 05/12/2006
//
////////////////////////////////////

module trunc_s1_decode(
    input_coef,
    output_coef
);

    input [24:0] input_coef;
    output [17:0] output_coef;

    // Nature of input bits:
    // ( sign_bit integer_bits fractional_bits )
    // Input: (1 16 8)
    // Output: (1 10 7)
    assign output_coef = input_coef[18:1];

endmodule // trunc_s1_decode
```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// DCT Back - Second stage of multiply module - Decode
// 05/12/2006
//
////////////////////////////////////

module dct_back_decode(
    clk,
    row0,
    row1,
    row2,
    row3,
    row4,
    row5,
    row6,
    row7,
    column,
    output_trunc,
    output_column
);

input clk;
input [71:0] row0, row1, row2, row3, row4, row5, row6, row7;
input [143:0] column;

output [63:0] output_column;
output [239:0] output_trunc;

wire [29:0] s2coef[7:0];

assign output_trunc[29:0]= s2coef[0];
assign output_trunc[59:30]= s2coef[1];
assign output_trunc[89:60]= s2coef[2];
assign output_trunc[119:90]= s2coef[3];
assign output_trunc[149:120]= s2coef[4];
assign output_trunc[179:150]= s2coef[5];
assign output_trunc[209:180]= s2coef[6];
assign output_trunc[239:210]= s2coef[7];

// Instantiate second set of multipliers
matrix_mult_s2 MTXMULTS2(.clk(clk),
    .row0(row0),
    .row1(row1),
    .row2(row2),
    .row3(row3),
    .row4(row4),
    .row5(row5),
    .row6(row6),
    .row7(row7),
    .column(column),
    .coef0(s2coef[0]),
    .coef1(s2coef[1]),

```

```

        .coef2(s2coef[2]),
        .coef3(s2coef[3]),
        .coef4(s2coef[4]),
        .coef5(s2coef[5]),
        .coef6(s2coef[6]),
        .coef7(s2coef[7]));

    // Second round of truncating - TODO change these columns
    trunc_s2_decode TRUNCS200D(.input_coef(s2coef[0]),
    .output_coef(output_column[63:56]));
    trunc_s2_decode TRUNCS201D(.input_coef(s2coef[1]),
    .output_coef(output_column[55:48]));
    trunc_s2_decode TRUNCS202D(.input_coef(s2coef[2]),
    .output_coef(output_column[47:40]));
    trunc_s2_decode TRUNCS203D(.input_coef(s2coef[3]),
    .output_coef(output_column[39:32]));
    trunc_s2_decode TRUNCS204D(.input_coef(s2coef[4]),
    .output_coef(output_column[31:24]));
    trunc_s2_decode TRUNCS205D(.input_coef(s2coef[5]),
    .output_coef(output_column[23:16]));
    trunc_s2_decode TRUNCS206D(.input_coef(s2coef[6]),
    .output_coef(output_column[15: 8]));
    trunc_s2_decode TRUNCS207D(.input_coef(s2coef[7]),
    .output_coef(output_column[ 7: 0]));

endmodule // dct_back_decode

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Truncate Stage 2 (30 bits -> 10 bits) - Decode
// 04/24/2006
//
////////////////////////////////////

module trunc_s2_decode(
    input_coef,
    output_coef
);

    input [29:0] input_coef;
    output [7:0] output_coef;

    reg          signed [9:0] interm;
    reg [7:0]    output_coef;

    // Nature of input bits:
    // ( sign_bit integer_bits fractional_bits )
    // Input:  (1 14 15)
    // Output: (1 9 0)
    //
    // Check to see if INTERM > 255 or < 0
    // IF so, assign to border values
    always @(input_coef)
        begin
            interm = input_coef[25:15];
            if (interm[9]) // negative
                output_coef = 0;
            else if (interm >= 255) // too large
                output_coef = 255;
            else
                output_coef = interm[7:0] + input_coef[14];
        end

endmodule // trunc_s2_decode

```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
// Decoder Finite State Machine
// 05/01/2006
//
////////////////////////////////////

module decoder_fsm(
    clk,
    reset,
    resolution_select,
    max_blocks,
    max_lines,
    addr_active,
    line_read,
    block_read,
    line_write,
    block_write,
    column_select,
    column_select_write,
    inter_row_cnt,
    input_select,
    decode_busy,
    wen,
    state);

    input clk, reset;
    input [2:0] resolution_select;
    input [4:0] max_blocks, max_lines;
    input [9:0] addr_active;

    output [4:0] line_read, block_read;
    output [4:0] line_write, block_write;
    output [2:0] column_select;
    output [2:0] column_select_write;
    output [2:0] inter_row_cnt;
    output [2:0] input_select;
    output wen;
    output decode_busy;
    output state;

    // Variables for Registered Outputs
    reg [4:0] line_read, block_read;
    reg [4:0] line_write, block_write;
    reg [2:0] column_select, column_select_write;
    reg [2:0] inter_row_cnt;
    reg [2:0] input_select;
    reg wen;
    reg decode_busy;

    reg [4:0] line_read_int, block_read_int;
    reg [2:0] column_select_int, inter_row_cnt_int;
    reg wen_int;

```



```

// State
reg          state, next;

parameter    IDLE = 0;
parameter    DECODE_BLOCK = 1;
integer      i;
parameter    PIPELINE = 3;

parameter    NORMAL = 0;
parameter    ZOOM = 1;
parameter    OTHER = 2;

// Pipelined output selector
reg [2:0]     input_pipe[PIPELINE-1:0], input_pipe_int;
reg [4:0]     block_write_pipe[PIPELINE-1:0];
reg [4:0]     line_write_pipe[PIPELINE-1:0];
reg [2:0]     column_select_write_pipe[PIPELINE-1:0];
reg          wen_pipe[PIPELINE-1:0];

always @(posedge clk)
begin
    if (!reset)
        begin
            line_read <= 0;
            block_read <= 0;
            column_select <= 0;
            inter_row_cnt <= 0;
            input_select <= 0;
            state <= IDLE;

            line_write <= 0;
            block_write <= 0;
            column_select_write <= 0;
            wen <= 0;

            for (i = 0; i < PIPELINE; i = i + 1)
                begin
                    block_write_pipe[i] <= 0;
                    line_write_pipe[i] <= 0;
                    column_select_write_pipe[i] <= 0;
                    wen_pipe[i] <= 0;
                end
        end
    else
        begin
            line_read <= line_read_int;
            block_read <= block_read_int;
            column_select <= column_select_int;
            inter_row_cnt <= inter_row_cnt_int;
            input_select <= input_pipe_int;

            line_write <= line_write_pipe[PIPELINE-1];
            block_write <= block_write_pipe[PIPELINE-1];
            column_select_write <= column_select_write_pipe[PIPELINE-1];
            wen <= wen_pipe[PIPELINE-1];
        end
end

```

```

state <= next;

for (i = 0; i < PIPELINE-1; i = i + 1)
begin
    block_write_pipe[i+1] <= block_write_pipe[i];
    line_write_pipe[i+1] <= line_write_pipe[i];
    column_select_write_pipe[i+1] <= column_select_write_pipe[i];
    wen_pipe[i+1] <= wen_pipe[i];
end

block_write_pipe[0] <= block_read_int;
line_write_pipe[0] <= line_read_int;
column_select_write_pipe[0] <= column_select_int;
wen_pipe[0] <= wen_int;

end // else: !if(!reset)
end // always @ (posedge clk)

always @(line_read or block_read or addr_active or column_select or state or
inter_row_cnt or resolution_select)
begin

    line_read_int = line_read;
    block_read_int = block_read;
    column_select_int = column_select;
    inter_row_cnt_int = inter_row_cnt;
    next = state;

    decode_busy = 1;
    wen_int = 0;

    case (state)
        //
        // IDLE -> waiting for Video Capture to finish writing lines
        //
        IDLE:
            begin
                // NOTICE -> not registered
                decode_busy = 0;
                input_pipe_int = 0;
                // Do nothing, wait for addr_active to move past current_line
                if (line_read*30 + block_read == addr_active)
                    next = IDLE;
                else
                    next = DECODE_BLOCK;
            end
        //
        // Decode Block -> Multiply current block by DCT rows
        //
        DECODE_BLOCK:
            begin
                // Reset block_dct
                if (inter_row_cnt == 7)
                    inter_row_cnt_int = 0;
                else
                    inter_row_cnt_int = inter_row_cnt + 1;
            end
    end
end

```

```

        if ((inter_row_cnt == 7) && (column_select == 7))
            column_select_int = 0;
        else if (inter_row_cnt == 7)
            column_select_int = column_select + 1;

        // Reset Blocks
        if ((inter_row_cnt == 6) && (column_select == 7) && (block_read ==
max_blocks))
            block_read_int = 0;
        else if ((inter_row_cnt == 6) && (column_select == 7))
            block_read_int = block_read + 1;

        // Reset Lines
        if ((inter_row_cnt == 6) && (column_select == 7) && (block_read ==
max_blocks) && (line_read == max_lines))
            line_read_int = 0;
        else if ((inter_row_cnt == 6) && (column_select == 7) &&
(block_read == max_blocks))
            line_read_int = line_read + 1;

        // Set next state
        if ((inter_row_cnt == 7) && (column_select == 7) && (block_read ==
max_blocks) && (line_read == max_lines))
            next = IDLE;
        // If next line + block to be read is address active -> then
transition into IDLE state
        else if (line_read_int*30 + block_read_int == addr_active)
            next = IDLE;
        else
            next = DECODE_BLOCK;

        // Set Write Enable
        if (inter_row_cnt == 5)
            wen_int = 1;
        else
            wen_int = 0;

        // Select input_pipe element
        if (inter_row_cnt == 7)
            case (resolution_select)
                NORMAL:
                    case (column_select)
                        0: input_pipe_int = 3;
                        1: input_pipe_int = 2;
                        2: input_pipe_int = 1;
                        default: input_pipe_int = 0;
                    endcase // case(column_select_int)
                ZOOM:
                    case (column_select)
                        0: input_pipe_int = 5;
                        1: input_pipe_int = 4;
                        2: input_pipe_int = 3;
                        3: input_pipe_int = 2;
                        4: input_pipe_int = 1;
                        default: input_pipe_int = 0;
                    endcase // case(column_select)
            endcase

```

```
OTHER:
  case (column_select)
    0: input_pipe_int = 4;
    1: input_pipe_int = 3;
    2: input_pipe_int = 2;
    3: input_pipe_int = 1;
    default: input_pipe_int = 0;
  endcase // case(column_select)
default:
  case (column_select)
    0: input_pipe_int = 3;
    1: input_pipe_int = 2;
    2: input_pipe_int = 1;
    default: input_pipe_int = 0;
  endcase // case(column_select_int)
endcase // case(resolution_select)
else
  input_pipe_int = 0;

  end // case: DECODE_BLOCK

endcase // case(state)

end // always @ (*)

endmodule // decode_fsm
```

```

////////////////////////////////////
//
// Vivek Shah
// Final Project
//
//
// Decoder - Video Memory Module
// 05/15/2006
//
////////////////////////////////////

module video_memory(
    wr_clk,
    r_clk,
    wen,
    line_write,
    block_write,
    column_select_write,
    column_data,
    addrb,
    dout0,
    dout1,
    dout2,
    dout3,
    dout4,
    dout5,
    dout6,
    dout7,
    addra
);

input wr_clk, r_clk;
input wen;
input [4:0] line_write, block_write;
input [2:0] column_select_write;
input [9:0] addrb;
input [63:0] column_data;

output [63:0] dout0, dout1, dout2, dout3, dout4, dout5, dout6, dout7;
output [12:0] addra;

wire [12:0]    addra;

assign    addra = 240*line_write + 8*block_write + column_select_write;

video_mem VIDMEM000D(.addra(addra), .addrb(addrb), .clka(wr_clk),
.clkb(r_clk), .dina(column_data[ 7: 0]), .doutb(dout0), .wea(wen));
video_mem VIDMEM001D(.addra(addra), .addrb(addrb), .clka(wr_clk),
.clkb(r_clk), .dina(column_data[15: 8]), .doutb(dout1), .wea(wen));
video_mem VIDMEM002D(.addra(addra), .addrb(addrb), .clka(wr_clk),
.clkb(r_clk), .dina(column_data[23:16]), .doutb(dout2), .wea(wen));
video_mem VIDMEM003D(.addra(addra), .addrb(addrb), .clka(wr_clk),
.clkb(r_clk), .dina(column_data[31:24]), .doutb(dout3), .wea(wen));
video_mem VIDMEM004D(.addra(addra), .addrb(addrb), .clka(wr_clk),
.clkb(r_clk), .dina(column_data[39:32]), .doutb(dout4), .wea(wen));
video_mem VIDMEM005D(.addra(addra), .addrb(addrb), .clka(wr_clk),
.clkb(r_clk), .dina(column_data[47:40]), .doutb(dout5), .wea(wen));

```

```
    video_mem VIDMEM006D(.addra(addr_a), .addrb(addr_b), .clka(wr_clk),  
.clkb(r_clk), .dina(column_data[55:48]), .doutb(dout6), .wea(wen));  
    video_mem VIDMEM007D(.addra(addr_a), .addrb(addr_b), .clka(wr_clk),  
.clkb(r_clk), .dina(column_data[63:56]), .doutb(dout7), .wea(wen));  
  
endmodule // video_memory
```

```

module video_blackwhite(clk, reset, dout0, dout1, dout2, dout3, dout4, dout5,
dout6, dout7, addr,
    vga_out_sync_b, vga_out_blank_b, vga_out_hsync, vga_out_vsync, vga_out);
    input clk, reset;
    input [63:0] dout0, dout1, dout2, dout3, dout4, dout5, dout6, dout7;
    output [9:0] addr;
    output vga_out_sync_b, vga_out_blank_b, vga_out_hsync, vga_out_vsync;
    output [23:0] vga_out;

wire [7:0] Y;
wire [9:0] pixel_count, line_count;

read64 read_decode_mem(clk, reset, Y, dout0, dout1, dout2, dout3, dout4, dout5,
dout6, dout7, addr,
    pixel_count, line_count, read_counter, read_counter2, switch_dout_counter,
iteration_counter,
    first_eight, second_eight, dout, start);

// Instantiate VGA Controller

VGA_controller Controller_display(clk, reset, hsync, vsync, pixel_count,
    line_count, vga_out_sync_b, vga_out_blank_b, hblank, vblank, dout);

// Instantiate delay

delay sync_delay_display(clk, hsync, vsync, vga_out_hsync, vga_out_vsync);

// Instantiate display

wire [23:0] RGB_out;
wire [38:0] doutb;
wire [7:0] R, G, B;
assign R = 8'b0;
assign G = 8'b0;
assign B = 8'b0;
assign doutb = 38'b0;
assign RGB_out = {Y, Y, Y};

display top_display_display(clk, R, G, B, pixel_count,
    line_count, doutb, addrb, vga_out, RGB_out);

endmodule

/*
    video_blackwhite(
        .clk(clk),
        .reset(reset),
        .dout0(dout0),
        .dout1(dout1),
        .dout2(dout2),
        .dout3(dout3),
        .dout4(dout4),
        .dout5(dout5),
        .dout6(dout6),
        .dout7(dout7),
        .addr(addr),
        .vga_out_sync_b(vga_out_sync_b),

```

```
.vga_out_blank_b(vga_out_blank_b),  
.vga_out_hsync(vga_out_hsync),  
.vga_out_vsync(vga_out_vsync),  
.vga_out(vga_out)  
);  
*/
```



```

module read64(clk, reset, Y, dout0, dout1, dout2, dout3, dout4, dout5, dout6,
dout7, addr,
    pixel_count, line_count, read_counter, read_counter2, switch_dout_counter,
iteration_counter,
    first_eight, second_eight, dout, start);
    input clk, reset;
    input [9:0] pixel_count, line_count;
    input [63:0] dout0, dout1, dout2, dout3, dout4, dout5, dout6, dout7;
    output [9:0] addr;
    output [7:0] Y;

    //new
    output [3:0] read_counter;
    output [4:0] read_counter2;
    output [3:0] switch_dout_counter;
    output [7:0] iteration_counter;
    output [63:0] first_eight, second_eight, dout;
    output start;

wire [63:0] dout0, dout1, dout2, dout3, dout4, dout5, dout6, dout7;

reg [7:0] Y, iteration_counter;
reg [63:0] data_in = 64'b0;
reg [3:0] read_counter;
reg [4:0] read_counter2;
reg [3:0] switch_dout_counter;

reg [63:0] first_eight, second_eight, dout;
reg being_read, start;

reg [9:0] addr;
reg [9:0] base_addr = 0;
reg [9:0] base_addr_int;

parameter x = 300;

//current
always @ (posedge clk)
begin
    if (reset)
        begin
            read_counter <= 1;
            read_counter2 <= 0;
            being_read <= 0;
            iteration_counter <= 0;
            base_addr <= 0;
            addr <= 0;
            //
            switch_dout_counter <= 0;
        end
    else if (pixel_count == x - 5)
        begin

            case (line_count)
            0:
                base_addr <= 0;
            8:
                base_addr <= 30;
            endcase
        end
end

```

```
16:      base_addr <= 60;
24:      base_addr <= 90;
32:      base_addr <= 120;
40:      base_addr <= 150;
48:      base_addr <= 180;
56:      base_addr <= 210;
64:      base_addr <= 240;
72:      base_addr <= 270;
80:      base_addr <= 300;
88:      base_addr <= 330;
96:      base_addr <= 360;
104:     base_addr <= 390;
112:     base_addr <= 420;
120:     base_addr <= 450;
128:     base_addr <= 480;
136:     base_addr <= 510;
144:     base_addr <= 540;
152:     base_addr <= 570;
160:     base_addr <= 600;
168:     base_addr <= 630;
176:     base_addr <= 660;
184:     base_addr <= 690;
192:     base_addr <= 720;
200:     base_addr <= 750;
208:     base_addr <= 780;
216:     base_addr <= 810;
224:     base_addr <= 840;
232:     base_addr <= 870;
240:
```

```

        base_addr <= 600;
default: base_addr <= base_addr;
endcase
end
/*
if (line_count == 0)
    switch_dout_counter <= 0;
else
    switch_dout_counter <= switch_dout_counter + 1;

case (switch_dout_counter)
0:
    dout <= dout7;
1:
    dout <= dout6;
2:
    dout <= dout5;
3:
    dout <= dout4;
4:
    dout <= dout3;
5:
    dout <= dout2;
6:
    dout <= dout1;
7:
    dout <= dout0;
endcase
end
*/
else if (pixel_count == x - 3)
    addr <= base_addr;
else if (pixel_count == x)
    if (line_count == 0)
        begin
            read_counter <= 1;
            read_counter2 <= 0;
            being_read <= 0;
            iteration_counter <= 0;
        end
    else if (line_count < 240)
        begin
            read_counter <= 1;
            read_counter2 <= 0;
            being_read <= 0;
        end
    else
        begin
            read_counter <= read_counter;
            read_counter2 <= read_counter2;
        end
    else if ((pixel_count == x + 1) && ((line_count == 0) || (line_count <
240)))
        begin
            start <= 1;

            case (switch_dout_counter)

```

```

0:      first_eight <= dout7;
1:      first_eight <= dout6;
2:      first_eight <= dout5;
3:      first_eight <= dout4;
4:      first_eight <= dout3;
5:      first_eight <= dout2;
6:      first_eight <= dout1;
7:      first_eight <= dout0;
default: first_eight <= dout7;
endcase

end

/*
if (line_count == 0)
    begin
        first_eight <= dout7;
        start <= 1;
    end
else
    begin
        first_eight <= dout;
        start <= 1;
    end

/*
case (line_count)
0:
    first_eight <= dout7;
default: first_eight <= dout;
endcase

*/

else if (start)
    case (being_read)
0:
    case(read_counter)
/*
0:
    begin
        Y <= first_eight[63:56];
        read_counter <= read_counter + 1;
    end */
1:
    begin
        Y <= first_eight[63:56];
        read_counter <= read_counter + 1;
    end
2:
    begin
        Y <= first_eight[55:48];
        read_counter <= read_counter + 1;
    end

```

```

3:
    begin
        Y <= first_eight[47:40];
        read_counter <= read_counter + 1;
        addr <= addr + 1;
    end
4:
    begin
        Y <= first_eight[39:32];
        read_counter <= read_counter + 1;
    end
5:
    begin
        Y <= first_eight[31:24];
        read_counter <= read_counter + 1;
    end
6:
    begin
        Y <= first_eight[23:16];
        read_counter <= read_counter + 1;
    end
7:
    begin
        Y <= first_eight[15:8];
        read_counter <= read_counter + 1;
    end
8:
    begin
        Y <= first_eight[7:0];
        read_counter <= 1;
        being_read <= (being_read)? 0 : 1;
        read_counter2 <= (read_counter2 == 29)? 0 :
read_counter2 + 1;
        second_eight <= dout;
    end
default: Y <= Y;
endcase
1:
    case(read_counter)
0:
        begin
            Y <= second_eight[63:56];
            read_counter <= read_counter + 1;
        end
1:
        begin
            Y <= second_eight[63:56];
            read_counter <= read_counter + 1;
        end
2:
        begin
            Y <= second_eight[55:48];
            read_counter <= read_counter + 1;
        end
3:
        begin
            Y <= second_eight[47:40];

```

```

        read_counter <= read_counter + 1;
        addr <= (read_counter2 == 29)? base_addr : addr + 1;
    end
4:
    begin
        Y <= second_eight[39:32];
        read_counter <= read_counter + 1;
/*
    if (read_counter2 == 29)
        switch_dout_counter <= (switch_dout_counter == 7)?
0 :
        switch_dout_counter + 1;
    else
        switch_dout_counter <= switch_dout_counter;

    if (read_counter2 == 29)
        if (line_count == 0)
            switch_dout_counter <= 0;
        else
            switch_dout_counter <= switch_dout_counter +
1;
*/
    end
5:
    begin
        Y <= second_eight[31:24];
        read_counter <= read_counter + 1;
    end
6:
    begin
        Y <= second_eight[23:16];
        read_counter <= read_counter + 1;
    end
7:
    begin
        Y <= second_eight[15:8];
        read_counter <= read_counter + 1;
    end
8:
    begin
        Y <= second_eight[7:0];
        read_counter <= 1;
        being_read <= (being_read)? 0 : 1;
        read_counter2 <= (read_counter2 == 29)? 0 :
read_counter2 + 1;
        iteration_counter <= ((read_counter2 == 29) &&
(iteration_counter == 239))?
0 :
(read_counter2 == 29)? iteration_counter + 1 : iteration_counter;
        first_eight <= dout;
        start <= (read_counter2 == 29)? 0 : 1;
    end
    default: Y <= Y;
    endcase
endcase
else

```

```

        begin
            Y <= 8'h11;
            start <= 0;
        end
end

always @ (posedge clk)
begin
    if (reset)
        switch_dout_counter <= 0;
    else if ((read_counter == 4) && (read_counter2 == 29))
        switch_dout_counter <= (switch_dout_counter == 7)? 0 :

        switch_dout_counter + 1;
    else
        switch_dout_counter <= switch_dout_counter;
end

always @ (posedge clk)
begin
    case (switch_dout_counter)
    0:
        dout <= dout7;
    1:
        dout <= dout6;
    2:
        dout <= dout5;
    3:
        dout <= dout4;
    4:
        dout <= dout3;
    5:
        dout <= dout2;
    6:
        dout <= dout1;
    7:
        dout <= dout0;
    endcase
end

/*
always @ (posedge clk)
begin
    if (reset)
        begin
            base_addr <= 600;
        end
    else
        begin
            base_addr <= 600;
        end
end
*/

/*
always @ (posedge clk)

```

```
begin
  if (reset)
    base_addr <= 300;
  else if (pixel_count == x - 20)
    case (line_count)
      0:
        base_addr <= 600;
      default: base_addr <= base_addr;
    /*
      0:
        base_addr <= 0;
      8:
        base_addr <= 30;
      16:
        base_addr <= 60;
      24:
        base_addr <= 90;
      32:
        base_addr <= 120;
      40:
        base_addr <= 150;
      48:
        base_addr <= 180;
      56:
        base_addr <= 210;
      64:
        base_addr <= 240;
      72:
        base_addr <= 270;
      80:
        base_addr <= 300;
      88:
        base_addr <= 330;
      96:
        base_addr <= 360;
      104:
        base_addr <= 390;
      112:
        base_addr <= 420;
      120:
        base_addr <= 450;
      128:
        base_addr <= 480;
      136:
        base_addr <= 510;
      144:
        base_addr <= 540;
      152:
        base_addr <= 570;
      160:
        base_addr <= 600;
      168:
        base_addr <= 630;
      176:
        base_addr <= 660;
      184:
        base_addr <= 690;
      192:
```



```
        base_addr <= 720;
200:    base_addr <= 750;
208:    base_addr <= 780;
216:    base_addr <= 810;
224:    base_addr <= 840;
232:    base_addr <= 870;
240:    base_addr <= 600;
default: base_addr <= base_addr; */

/*
0:    base_addr_int <= 0;
8:    base_addr_int <= 30;
16:   base_addr_int <= 60;
24:   base_addr_int <= 90;
32:   base_addr_int <= 120;
40:   base_addr_int <= 150;
48:   base_addr_int <= 180;
56:   base_addr_int <= 210;
64:   base_addr_int <= 240;
72:   base_addr_int <= 270;
80:   base_addr_int <= 300;
88:   base_addr_int <= 330;
96:   base_addr_int <= 360;
104:  base_addr_int <= 390;
112:  base_addr_int <= 420;
120:  base_addr_int <= 450;
128:  base_addr_int <= 480;
136:  base_addr_int <= 510;
144:  base_addr_int <= 540;
152:  base_addr_int <= 570;
160:  base_addr_int <= 600;
```

```

168:     base_addr_int <= 630;
176:     base_addr_int <= 660;
184:     base_addr_int <= 690;
192:     base_addr_int <= 720;
200:     base_addr_int <= 750;
208:     base_addr_int <= 780;
216:     base_addr_int <= 810;
224:     base_addr_int <= 840;
232:     base_addr_int <= 870;
240:     base_addr_int <= 600;
default: base_addr_int <= base_addr_int;
endcase
else
    base_addr_int <= base_addr_int;
end
*/
/*
    else if ((read_counter == 1) && (read_counter2 == 29) &&
(switch_dout_counter == 7))
        case (line_count)
            7:
                base_addr <= 30;
            207:
                base_addr <= 330;
            default: base_addr <= base_addr;
            endcase
        */
/*
always @ (posedge clk)
begin
    if (reset)
        base_addr_int <= 0;
    else
        base_addr <= 0;

/*
    case (line_count)
        0:
            base_addr <= 0;
        8:
            base_addr <= 30;
        16:
            base_addr <= 60;
        24:
            base_addr <= 90;
        32:
            base_addr <= 120;
        40:
            base_addr <= 150;

```

```
48:     base_addr <= 180;
56:     base_addr <= 210;
64:     base_addr <= 240;
72:     base_addr <= 270;
80:     base_addr <= 300;
88:     base_addr <= 330;
96:     base_addr <= 360;
104:    base_addr <= 390;
112:    base_addr <= 420;
120:    base_addr <= 450;
128:    base_addr <= 480;
136:    base_addr <= 510;
144:    base_addr <= 540;
152:    base_addr <= 570;
160:    base_addr <= 600;
168:    base_addr <= 630;
176:    base_addr <= 660;
184:    base_addr <= 690;
192:    base_addr <= 720;
200:    base_addr <= 750;
208:    base_addr <= 780;
216:    base_addr <= 810;
224:    base_addr <= 840;
232:    base_addr <= 870;
default: base_addr <= base_addr;
endcase    */
```

```
//end
```

```
endmodule
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:39:55 05/14/2006
// Design Name:    dct_back_decode
// Module Name:    tb_dct_back_decode.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: dct_back_decode
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_dct_back_decode_v;

    // Inputs
    reg clk;
    reg [71:0] row0;
    reg [71:0] row1;
    reg [71:0] row2;
    reg [71:0] row3;
    reg [71:0] row4;
    reg [71:0] row5;
    reg [71:0] row6;
    reg [71:0] row7;
    reg [143:0] column;

    // Outputs
    wire [239:0] output_trunc;
    wire [63:0]   output_column;

    wire [71:0]   dct_row[7:0];
    wire [71:0]   dct_col[7:0];
    wire [11:0]   coef[7:0];
    wire [29:0]   output_trunc_seg[7:0];
    integer i;

    // Instantiate DCT Table
    dct_table DCTTABLE0(.clk(clk),
                        .row0(dct_row[0]),
                        .row1(dct_row[1]),
                        .row2(dct_row[2]),
                        .row3(dct_row[3]),
                        .row4(dct_row[4]),
                        .row5(dct_row[5]),
                        .row6(dct_row[6]),

```

```

        .row7(dct_row[7
1),
        .col0(dct_col[0]),
        .col1(dct_col[1]),
        .col2(dct_col[2]),
        .col3(dct_col[3]),
        .col4(dct_col[4]),
        .col5(dct_col[5]),
        .col6(dct_col[6]),
        .col7(dct_col[7])
    );

// Instantiate the Unit Under Test (UUT)
dct_back_decode uut (
    .clk(clk),
    .row0(row0),
    .row1(row1),
    .row2(row2),
    .row3(row3),
    .row4(row4),
    .row5(row5),
    .row6(row6),
    .row7(row7),
    .column(column),
    .output_trunc(output_trunc),
    .output_column(output_column)
);

assign  coef[0] = output_column[7:0];
assign  coef[1] = output_column[15:8];
assign  coef[2] = output_column[23:16];
assign  coef[3] = output_column[31:24];
assign  coef[4] = output_column[39:32];
assign  coef[5] = output_column[47:40];
assign  coef[6] = output_column[55:48];
assign  coef[7] = output_column[63:56];

assign  output_trunc_seg[0] = output_trunc[29:0];
assign  output_trunc_seg[1] = output_trunc[59:30];
assign  output_trunc_seg[2] = output_trunc[89:60];
assign  output_trunc_seg[3] = output_trunc[119:90];
assign  output_trunc_seg[4] = output_trunc[149:120];
assign  output_trunc_seg[5] = output_trunc[179:150];
assign  output_trunc_seg[6] = output_trunc[209:180];
assign  output_trunc_seg[7] = output_trunc[239:210];

always #20 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    row0 = dct_col[0];
    row1 = dct_col[1];
    row2 = dct_col[2];
    row3 = dct_col[3];
    row4 = dct_col[4];
    row5 = dct_col[5];

```

```

row6 = dct_col[6];
row7 = dct_col[7];
column = 0;

// Wait 100 ns for global reset to finish
#100;
row0 = dct_col[0];
row1 = dct_col[1];
row2 = dct_col[2];
row3 = dct_col[3];
row4 = dct_col[4];
row5 = dct_col[5];
row6 = dct_col[6];
row7 = dct_col[7];

// Add stimulus here
#40 column = {18'd2062, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0};
#40 column = {18'd255, 18'd255, 18'd255, 18'd255, 18'd255, 18'd255,
18'd255, 18'd255};
#40 column = {18'd137, 18'd137, 18'd137, 18'd137, 18'd137, 18'd137,
18'd137, 18'd137};
#40 column = {18'd127, 18'd127, 18'd127, 18'd127, 18'd127, 18'd127,
18'd127, 18'd127};
#40 column = {18'd1000, 18'd1000, 18'd1000, 18'd1000, 18'd1000, 18'd1000,
18'd1000, 18'd1000};
#40 column = {18'd500, 18'd500, 18'd500, 18'd500, 18'd500, 18'd500,
18'd500, 18'd500};
#40 column = {18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0};
#40 column = {18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0};
#40 column = {18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0};
#40 column = {18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0};
#40 column = {18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0};
#40 column = {18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0, 18'd0};

end // initial begin

endmodule // tb_dct_back_decode_v

```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:40:34 05/14/2006
// Design Name:    dct_front_decode
// Module Name:    tb_dct_front_decode.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: dct_front_decode
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_dct_front_decode_v;

    // Inputs
    reg clk;
    reg [103:0] row;
    reg [71:0] column;

    // Outputs
    wire [24:0] output_trunc;
    wire [17:0] output_coef;

    wire [71:0]    dct_row[7:0];
    wire [71:0]    dct_col[7:0];
    //reg [71:0]    row[7:0];
    wire [10:0]    col_int;
    wire [6:0]     col_frac;
    integer    i;

    // Instantiate the Unit Under Test (UUT)
    dct_front uut (
        .clk(clk),
        .row(row),
        .column(column),
        .output_trunc(output_trunc),
        .output_coef(output_coef)
    );

    // Instantiate DCT Table
    dct_table DCTTABLE0(.clk(clk),
        .row0(dct_row[0]),
        .row1(dct_row[1]),
        .row2(dct_row[2]),
        .row3(dct_row[3]),

```

```

        .row4(dct_row[4]),
        .row5(dct_row[5]),
        .row6(dct_row[6]),
        .row7(dct_row[7]),
        .col0(dct_col[0]),
        .col1(dct_col[1]),
        .col2(dct_col[2]),
        .col3(dct_col[3]),
        .col4(dct_col[4]),
        .col5(dct_col[5]),
        .col6(dct_col[6]),
        .col7(dct_col[7])
    );

assign    col_frac = output_coef[6:0];
assign    col_int  = output_coef[17:7];

always #20 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    row = 0;
    column = 0;

    // Wait 100 ns for global reset to finish
    #170;

    // Add stimulus here

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1};
            column = dct_row[i];
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd100};
            column = dct_row[i];
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {13'd127, 13'd127, 13'd127, 13'd127, 13'd127, 13'd127,
13'd127, 13'd127};
            column = dct_row[i];
        end

    for (i = 0; i < 8; i = i + 1)
        repeat (8) begin
            #40 row = {13'd4095, 13'd4095, 13'd4095, 13'd4095, 13'd4095, 13'd4095,
13'd4095, 13'd4095};
            column = dct_row[i];
        end

    for (i = 0; i < 8; i = i + 1)

```



```
        repeat (8) begin
            #40 row = {13'd2062, 13'd2062, 13'd2062, 13'd2062, 13'd2062, 13'd2062,
13'd2062, 13'd2062};
            column = dct_row[i];
        end

    end // initial begin
endmodule // tb_dct_front_decode_v
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:40:11 05/14/2006
// Design Name:    dct_multiply_decode
// Module Name:    tb_dct_multiply_decode.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: dct_multiply_decode
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_dct_multiply_decode_v;

    // Inputs
    reg clk;
    reg reset;
    reg [103:0] row;
    reg [2:0] column_select;
    reg [12:0] temp;

    // Outputs
    wire [63:0] output_column;
    wire [239:0] output_trunc;
    wire [17:0] stager1_output;
    wire [143:0] shift_reg_output;
    wire [29:0] output_trunc_seg[7:0];

    wire [7:0] col[7:0];
    wire [17:0] sreg_seg[7:0];

    assign col[0] = output_column[7:0];
    assign col[1] = output_column[15:8];
    assign col[2] = output_column[23:16];
    assign col[3] = output_column[31:24];
    assign col[4] = output_column[39:32];
    assign col[5] = output_column[47:40];
    assign col[6] = output_column[55:48];
    assign col[7] = output_column[63:56];

    assign sreg_seg[0] = shift_reg_output[17:0];
    assign sreg_seg[1] = shift_reg_output[35:18];
    assign sreg_seg[2] = shift_reg_output[53:36];
    assign sreg_seg[3] = shift_reg_output[71:54];
    assign sreg_seg[4] = shift_reg_output[89:72];

```

```

assign  sreg_seg[5] =  shift_reg_output[107:90];
assign  sreg_seg[6] =  shift_reg_output[125:108];
assign  sreg_seg[7] =  shift_reg_output[143:126];

assign  output_trunc_seg[0] = output_trunc[29:0];
assign  output_trunc_seg[1] = output_trunc[59:30];
assign  output_trunc_seg[2] = output_trunc[89:60];
assign  output_trunc_seg[3] = output_trunc[119:90];
assign  output_trunc_seg[4] = output_trunc[149:120];
assign  output_trunc_seg[5] = output_trunc[179:150];
assign  output_trunc_seg[6] = output_trunc[209:180];
assign  output_trunc_seg[7] = output_trunc[239:210];

// Instantiate the Unit Under Test (UUT)
dct_multiply_decode uut (
    .clk(clk),
    .reset(reset),
    .row(row),
    .column_select(column_select),
    .output_column(output_column),
    .output_trunc(output_trunc),
    .stage1_output(stage1_output),
    .shift_reg_output(shift_reg_output)
);

integer  i;

always #20 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    row = 1;
    column_select = 0;
    i = 0;

    // Wait 100 ns for global reset to finish
    #200 reset = 1;
    i = 8;
    #10;

    #40 row = {13'd2062, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    column_select = 0;

    for (i = 1; i < 8; i = i + 1)
        repeat (7) begin
            #40 row = {13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
            column_select = i;
        end

    #40 row = {13'd2062, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    column_select = 0;

    for (i = 1; i < 8; i = i + 1)
        repeat (7) begin
            #40 row = {13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
            column_select = i;
        end

```

```

end

#40 row = {13'd127, 13'd127, 13'd127, 13'd127, 13'd127, 13'd127, 13'd127,
13'd127};
column_select = 0;

for (i = 1; i < 8; i = i + 1)
repeat (7) begin
#40 row = {13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
column_select = i;
end

for (i = 0; i < 8; i = i + 1)
repeat (8) begin
#40 row = {13'd63, 13'd63, 13'd63, 13'd63, 13'd63, 13'd63, 13'd63,
13'd63};
column_select = i;
end

for (i = 0; i < 8; i = i + 1)
repeat (8) begin
#40 row = {13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1};
column_select = i;
end

for (i = 0; i < 8; i = i + 1)
repeat (8) begin
#40 row = {13'd1, 13'd100, 13'd1, 13'd100, 13'd1, 13'd100, 13'd1,
13'd100};
column_select = i;
end

for (i = 0; i < 8; i = i + 1)
repeat (8) begin
#40 column_select = i;
temp = 1 << (i+1);
row = temp << 13*i;
column_select = i;
end

end // initial begin

endmodule // tb_dct_multiply_decode_v

```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:28:16 05/14/2006
// Design Name:    decode_fsm
// Module Name:    tb_decode_fsm.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: decode_fsm
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_decode_fsm_v;

    // Inputs
    reg clk;
    reg reset;
    reg [2:0] resolution_select;
    reg [4:0] max_blocks;
    reg [4:0] max_lines;
    reg [9:0] addr_active;

    // Outputs
    wire [4:0] line_read;
    wire [4:0] block_read;
    wire [2:0] column_select;
    wire [2:0] inter_row_cnt;
    wire [2:0] output_select;
    wire      decode_busy;
    wire      state;

    // Instantiate the Unit Under Test (UUT)
    decode_fsm uut (
        .clk(clk),
        .reset(reset),
        .resolution_select(resolution_select),
        .max_blocks(max_blocks),
        .max_lines(max_lines),
        .addr_active(addr_active),
        .line_read(line_read),
        .block_read(block_read),
        .column_select(column_select),
        .inter_row_cnt(inter_row_cnt),
        .output_select(output_select),
        .decode_busy(decode_busy),

```

```
        .state(state)
    );

always #0.25 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    resolution_select = 0;
    max_blocks = 29;
    max_lines = 29;
    addr_active = 80;

    // Wait 100 ns for global reset to finish
    #10 reset = 1;

    // Add stimulus here
    #3000 addr_active = 81;
    #100 addr_active = 100;

end // initial begin

endmodule // tb_decode_fsm_v
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    03:04:08 05/15/2006
// Design Name:    decoder
// Module Name:    tb_decoder.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: decoder
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_decoder_v;

    // Inputs
    reg clk;
    reg reset;
    reg [2:0] resolution_select;
    reg [4:0] max_lines;
    reg [4:0] max_blocks;
    reg [9:0] addr_active;
    reg [103:0] row;

    // Outputs
    wire decode_busy;
    wire [2:0] inter_row_cnt;
    wire [4:0] block_read;
    wire [4:0] line_read;
    wire [63:0] data_output;
    wire wen;
    wire [103:0] mult_column;
    wire [2:0] column_select;
    wire [2:0] input_select;
    wire [17:0] stagel_output;
    wire [143:0] shift_reg_output;
    wire [4:0] line_write;
    wire [4:0] block_write;
    wire [2:0] column_select_write;

    integer i;
    wire [7:0] coef[7:0];

    assign coef[0] = data_output[7:0];
    assign coef[1] = data_output[15:8];
    assign coef[2] = data_output[23:16];

```



```
for (i = 0; i < 8; i = i + 1)
begin
    #40 row = {13'd1027, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    repeat (7) begin
        #40 row = {13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    end
end

for (i = 0; i < 8; i = i + 1)
begin
    #40 row = {13'd8, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    repeat (7) begin
        #40 row = {13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    end
end

for (i = 0; i < 8; i = i + 1)
begin
    #40 row = {13'd100, 13'd10, 13'd1, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    #40 row = {13'd10, 13'd1, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    #40 row = {13'd1, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    repeat (5) begin
        #40 row = {13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    end
end

end // initial begin

endmodule // tb_decoder_v
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    02:53:04 05/15/2006
// Design Name:    decoder_fsm
// Module Name:    tb_decoder_fsm.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: decoder_fsm
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_decoder_fsm_v;

    // Inputs
    reg clk;
    reg reset;
    reg [2:0] resolution_select;
    reg [4:0] max_blocks;
    reg [4:0] max_lines;
    reg [9:0] addr_active;

    // Outputs
    wire [4:0] line_read;
    wire [4:0] block_read;
    wire [4:0] line_write;
    wire [4:0] block_write;
    wire [2:0] column_select;
    wire [2:0] column_select_write;
    wire [2:0] inter_row_cnt;
    wire [2:0] input_select;
    wire      decode_busy;
    wire      state;
    wire      wen;

    // Instantiate the Unit Under Test (UUT)
    decoder_fsm uut (
        .clk(clk),
        .reset(reset),
        .resolution_select(resolution_select),
        .max_blocks(max_blocks),
        .max_lines(max_lines),
        .addr_active(addr_active),
        .line_read(line_read),

```

```

        .block_read(block_read),
        .line_write(line_write),
        .block_write(block_write),
        .column_select(column_select),
        .column_select_write(column_select_write),
        .inter_row_cnt(inter_row_cnt),
        .input_select(input_select),
        .decode_busy(decode_busy),
        .state(state),
        .wen(wen)
    );

always #0.25 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    resolution_select = 0;
    max_blocks = 29;
    max_lines = 29;
    addr_active = 80;

    // Wait 100 ns for global reset to finish
    #10 reset = 1;

    // Add stimulus here
    #3000 addr_active = 81;
    #100 addr_active = 100;

end // initial begin

endmodule // tb_decoder_fsm_v

```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:40:51 05/14/2006
// Design Name:    matrix_naive_decode
// Module Name:    tb_matrix_naive_decode.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: matrix_naive_decode
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_matrix_naive_decode_v;

    // Inputs
    reg clk;
    reg [103:0] row;
    reg [71:0] column;

    // Outputs
    wire [21:0] RX_0;
    wire [21:0] RX_1;
    wire [21:0] RX_2;
    wire [21:0] RX_3;
    wire [21:0] RX_4;
    wire [21:0] RX_5;
    wire [21:0] RX_6;
    wire [21:0] RX_7;
    wire [22:0] RI_0;
    wire [22:0] RI_1;
    wire [22:0] RI_2;
    wire [22:0] RI_3;
    wire [23:0] RM_0;
    wire [23:0] RM_1;
    wire [24:0] product;

    wire    signed [8:0] row_pixel [7:0];
    wire    signed [8:0] col_pixel [7:0];

    assign    row_pixel[0] = row[8:0];
    assign    row_pixel[1] = row[17:9];
    assign    row_pixel[2] = row[26:18];
    assign    row_pixel[3] = row[35:27];
    assign    row_pixel[4] = row[44:36];
    assign    row_pixel[5] = row[53:45];

```

```

assign      row_pixel[6] = row[62:54];
assign      row_pixel[7] = row[71:63];

assign      col_pixel[0] = column[8:0];
assign      col_pixel[1] = column[17:9];
assign      col_pixel[2] = column[26:18];
assign      col_pixel[3] = column[35:27];
assign      col_pixel[4] = column[44:36];
assign      col_pixel[5] = column[53:45];
assign      col_pixel[6] = column[62:54];
assign      col_pixel[7] = column[71:63];

// Instantiate the Unit Under Test (UUT)
matrix_naive_decode uut (
    .clk(clk),
    .row(row),
    .column(column),
    .RX_0(RX_0),
    .RX_1(RX_1),
    .RX_2(RX_2),
    .RX_3(RX_3),
    .RX_4(RX_4),
    .RX_5(RX_5),
    .RX_6(RX_6),
    .RX_7(RX_7),
    .RI_0(RI_0),
    .RI_1(RI_1),
    .RI_2(RI_2),
    .RI_3(RI_3),
    .RM_0(RM_0),
    .RM_1(RM_1),
    .product(product)
);

always #2 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    row = 0;
    column = 0;

    #5;

    // Add stimulus here
    #4 row = {13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1};
    column = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};

    #4 row = {13'd2, 13'd2, 13'd2, 13'd2, 13'd2, 13'd2, 13'd2, 13'd2};
    column = {9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2, 9'd2};

    #4 row = {13'd1, 13'd2, 13'd3, 13'd4, 13'd5, 13'd6, 13'd7, 13'd8};
    column = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};

    #4 row = {13'h0FFF, 13'h0FFF, 13'h0FFF, 13'h0FFF, 13'h0FFF, 13'h0FFF,
13'h0FFF, 13'h0FFF};
    column = {9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF, 9'hFF};

```

```
    #4 row = {13'h100, 13'h100, 13'h100, 13'h100, 13'h100, 13'h100, 13'h100,
13'h100};
    column = {9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100};

    #4 row = {13'h0FFF, 13'h0FFF, 13'h0FFF, 13'h0FFF, 13'h0FFF, 13'h0FFF,
13'h0FFF, 13'h0FFF};
    column = {9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100, 9'h100};

    #4 row = {13'd1, 13'd2, 13'd3, 13'd4, 13'd5, 13'd6, 13'd7, 13'd8};
    column = {9'd1, 9'd2, 9'd3, 9'd4, 9'd5, 9'd6, 9'd7, 9'd8};

    #4 row = {13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1, 13'd1};
    column = {9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1, 9'd1};

    #4 row = {13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0, 13'd0};
    column = {9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0, 9'd0};

end // initial begin

endmodule // tb_matrix_naive_decode_v
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:23:41 05/15/2006
// Design Name:    rx_control_unit
// Module Name:    tb_rx_control_unit.v
// Project Name:   final_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: rx_control_unit
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_rx_control_unit_v;

    // Inputs
    reg clk;
    reg reset;
    reg cts_b;
    reg rxd;
    reg decode_busy;

    // Outputs
    wire rts_b;
    wire enable;
    wire [77:0] encoded_data;
    wire [7:0] encoded_byte;
    wire [3:0] rs232_state;
    wire shift_state;
    wire wen;
    wire [9:0] write_addr;
    wire[2:0] state;

    // Instantiate the Unit Under Test (UUT)
    rx_control_unit uut (
        .clk(clk),
        .reset(reset),
        .cts_b(cts_b),
        .rxd(rxd),
        .decode_busy(decode_busy),
        .rts_b(rts_b),
        .enable(enable),
        .encoded_data(encoded_data),
        .encoded_byte(encoded_byte),
        .rs232_state(rs232_state),
        .shift_state(shift_state),

```

```
        .wen(wen),
        .write_addr(write_addr),
        .state(state)
    );

defparam uut.rx.BAUD_COUNTER = 3;
defparam uut.rx.HALF_BAUD_COUNTER = 1;
initial begin
    // Initialize Inputs
    clk = 0;
    reset = 0;
    cts_b = 0;
    rxd = 1;
    decode_busy = 1;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    #30;
    decode_busy = 0;

    repeat (20)
    begin
        rxd = 0;
        #40;
        rxd = 1;
        #360;
    end

end

endmodule
```



```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:11:54 05/15/2006
// Design Name:    video_memory
// Module Name:    tb_video_memory.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: video_memory
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_video_memory_v;

    // Inputs
    reg wr_clk;
    reg r_clk;
    reg wen;
    reg [4:0] line_write;
    reg [4:0] block_write;
    reg [2:0] column_select_write;
    reg [63:0] column_data;
    reg [9:0] addrb;

    // Outputs
    wire [63:0] dout0;
    wire [63:0] dout1;
    wire [63:0] dout2;
    wire [63:0] dout3;
    wire [63:0] dout4;
    wire [63:0] dout5;
    wire [63:0] dout6;
    wire [63:0] dout7;
    wire [12:0] addra;

    // Instantiate the Unit Under Test (UUT)
    video_memory uut (
        .wr_clk(wr_clk),
        .r_clk(r_clk),
        .wen(wen),
        .line_write(line_write),
        .block_write(block_write),
        .column_select_write(column_select_write),
        .column_data(column_data),
        .addrb(addrb),

```

```

        .dout0(dout0),
        .dout1(dout1),
        .dout2(dout2),
        .dout3(dout3),
        .dout4(dout4),
        .dout5(dout5),
        .dout6(dout6),
        .dout7(dout7),
        .addra(addra)
    );

always #1 wr_clk = ~wr_clk;
always #1 r_clk = ~r_clk;

initial begin
    // Initialize Inputs
    wr_clk = 0;
    r_clk = 0;
    wen = 0;
    line_write = 0;
    block_write = 0;
    column_select_write = 0;
    column_data = 0;
    addrb = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    repeat (30) begin
        repeat (30) begin
            repeat (8) begin
                #40 column_select_write = column_select_write + 1;
            end
            block_write = block_write + 1;
            column_select_write = 0;
        end
        line_write = line_write + 1;
        block_write = 0;
    end
end // initial begin

endmodule // tb_video_memory_v

```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:48:31 05/15/2006
// Design Name:    wireless_memory
// Module Name:    tb_wireless_memory.v
// Project Name:   decoder_final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: wireless_memory
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_wireless_memory_v;

    // Inputs
    reg wr_clk;
    reg [9:0] wr_addr;
    reg [77:0] din;
    reg          wen;
    reg          r_clk;
    reg [4:0] line_read;
    reg [4:0] block_read;
    reg [2:0] inter_row_cnt;

    // Outputs
    wire [103:0] dout;

    // Instantiate the Unit Under Test (UUT)
    wireless_memory uut (
        .wr_clk(wr_clk),
        .wr_addr(wr_addr),
        .din(din),
        .wen(wen),
        .r_clk(r_clk),
        .line_read(line_read),
        .block_read(block_read),
        .inter_row_cnt(inter_row_cnt),
        .dout(dout)
    );

    always #1 wr_clk = ~wr_clk;
    always #1 r_clk = ~r_clk;

    wire [12:0] output_seg[7:0];

```

```

assign output_seg[0] = dout[12:0];
assign output_seg[1] = dout[25:13];
assign output_seg[2] = dout[38:26];
assign output_seg[3] = dout[51:39];
assign output_seg[4] = dout[64:52];
assign output_seg[5] = dout[77:65];
assign output_seg[6] = dout[90:78];
assign output_seg[7] = dout[103:91];

initial begin
    // Initialize Inputs
    wr_clk = 0;
    wr_addr = 0;
    din = 0;
    wen = 0;
    r_clk = 0;
    line_read = 0;
    block_read = 0;
    inter_row_cnt = 0;

    // Wait 100 ns for global reset to finish
    #20;

    // Add stimulus here
    repeat (30) begin
        repeat (30) begin
            repeat (8) begin
                #40 inter_row_cnt = inter_row_cnt + 1;
            end
            block_read = block_read + 1;
            inter_row_cnt = 0;
        end
        line_read = line_read + 1;
        block_read = 0;
    end

end

endmodule

```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    19:44:19 05/13/2006
// Design Name:    rs232_receiverFSM
// Module Name:    tb_receiver_FSM.v
// Project Name:   final_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: rs232_receiverFSM
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_receiver_FSM_v;

    // Inputs
    reg clk;
    reg reset;
    reg cts_b;
    reg rxd;

    // Outputs
    wire [7:0] data;
    wire data_ready;
    wire rts_b;

    // Instantiate the Unit Under Test (UUT)
    rs232_receiverFSM uut (
        .clk(clk),
        .reset(reset),
        .cts_b(cts_b),
        .rxd(rxd),
        .data(data),
        .data_ready(data_ready),
        .rts_b(rts_b)
    );

    always #5 clk = ~clk;
    defparam uut.BAUD_COUNTER = 4;
    defparam uut.HALF_BAUD_COUNTER = 2;
    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 0;
        cts_b = 0;
        rxd = 0;
    end
endmodule

```

```
// Wait 100 ns for global reset to finish  
#100;
```

```
// Add stimulus here
```

```
reset = 1;
```

```
rx_d = 0;
```

```
#460;
```

```
rx_d = 1;
```

```
#50;
```

```
rx_d = 0;
```

```
#150;
```

```
rx_d = 1;
```

```
#250;
```

```
end
```

```
endmodule
```

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    11:50:12 05/15/2006
// Design Name:    rx_shift_reg
// Module Name:    tb_rx_shift_reg.v
// Project Name:   final_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: rx_shift_reg
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb_rx_shift_reg_v;

    // Inputs
    reg clk;
    reg reset;
    reg data_ready;
    reg [7:0] encoded_byte;

    // Outputs
    wire done, state;
    wire [77:0] encoded_data;

    // Instantiate the Unit Under Test (UUT)
    rx_shift_reg uut (
        .clk(clk),
        .reset(reset),
        .data_ready(data_ready),
        .encoded_byte(encoded_byte),
        .done(done),
        .encoded_data(encoded_data),
        .state(state)
    );

    always #5 clk = ~clk;
    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 0;
        data_ready = 0;
        encoded_byte = 0;

        // Wait 100 ns for global reset to finish
        #100;
    end
endmodule

```

```
// Add stimulus here
reset = 1;
#30;
data_ready = 1;
#10;
data_ready = 0;
#30;
data_ready = 1;
#10;
data_ready = 0;
#30;
encoded_byte = 8'hAA;
data_ready = 1;
#10;
data_ready = 0;
#30;
encoded_byte = 8'h01;
data_ready = 1;
#10;
data_ready = 0;
#30;
data_ready = 1;
#10;
data_ready = 0;
#30;
data_ready = 1;
#10;
data_ready = 0;
#30;
data_ready = 1;
#10;
data_ready = 0;
#30;
data_ready = 1;
#10;
data_ready = 0;
#30;
data_ready = 1;
#10;
data_ready = 0;
#30;
data_ready = 1;
#10;
data_ready = 0;
#30;
data_ready = 1;
#10;
data_ready = 0;
```

end

endmodule